



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

University of Alberta

Symbolic Generation of The Dynamic
Equations for Rigid and Flexible Link
Manipulators

by



Luai I. El-Rayyes

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

Edmonton, Alberta

Spring 1990



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-60253-8

micro.

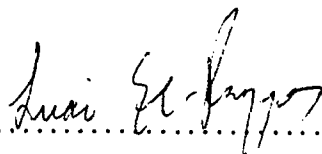
University of Alberta

Release Form

Name of Author: **Luai I. El-Rayyes**
Title of Thesis: **Symbolic Generation of The Dynamic Equations
for Rigid and Flexible Link Manipulators**
Degree: **Master of Science**
Year this degree granted: **1990**

Permission is hereby granted to **The University of Alberta Library** to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly, or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive tracts from it may be printed or otherwise reproduced without the author's written consent.


.....

(Student's signature)

Luai I. El-Rayyes
472 Wanyandi Road
Edmonton, Alberta
T5T 4M6

Date: *26 January 1990*

The University of Alberta
Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled

Symbolic Generation of The Dynamic Equations for Rigid and Flexible Link Manipulators

submitted by

Luai I. El-Rayyes

in partial fulfilment of the requirements for the degree of

Master of Science.

Roger W. Torrance
.....

(Supervisor)

Hong Jun
.....

Jim Fyfe
.....

.....

Date : *25 January 1990*
.....

Dedication

To My Parents

Abstract

The development of computationally efficient dynamic algorithms is a vital part of robotics research. In this thesis, the dynamics of both rigid link manipulators and flexible link manipulators are automatically generated using the symbolic generation technique. A Newton-Euler algorithm for rigid link manipulators was programmed in the symbolic generator *NEDYN*. A modified Lagrangian algorithm for the flexible link manipulators was programmed in the symbolic generator *FLEX*. A post-processor called *CLEAR* is utilized to further simplify and optimize the generated dynamics. Several dynamics problems were treated which include recursive inverse dynamics, closed form inverse dynamics and direct dynamics. The implementation of the symbolic generation techniques results in great reductions in the computational requirements for both rigid link and flexible link manipulators. Furthermore, the computational efficiency of the symbolic generator *NEDYN* was in many cases superior to the most efficient symbolic generators reported in literature to date. For the case of flexible link manipulators the symbolically generated direct dynamics code required only between 2.8% and 5.7% of the computations needed by the generic numerical formulations. Systematic documentation of the computational requirements for the flexible link manipulators is presented in this work. Several test cases were studied to verify both the dynamic models and the correct implementation of the different algorithms in the symbolic generators.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Roger Toogood for his guidance and support throughout this research.

I also would like to acknowledge the financial support provided to me by the department of mechanical engineering in the form of research and teaching assistantships.

Contents

1	Introduction	1
1.1	The Dynamics of Rigid Link Open Chain Robot Manipulators	6
1.1.1	Computational requirements of numerical programming	7
1.1.2	Symbolic Generation of the Dynamics	11
1.2	The dynamics of flexible link manipulators	13
1.3	Outline of this work	16
2	Dynamics of Robot Manipulators	18
2.1.1	Kinematics	19
2.1.2	Dynamics	20
2.2	The Direct Dynamics	24
2.3	The Jacobian Matrix	27
2.4	The Closed Form Dynamics	28
2.4.1	The Inertia Term	29
2.4.2	Centrifugal and Coriolis Term	29

2.4.3	Gravity Term	29
2.4.4	External Force Term	30
2.5	Inverse Dynamic Algorithm for Flexible Link Manipulator	30
2.5.1	Kinematics of The Flexible Links	31
2.5.2	Dynamics of the Flexible Links	35
2.5.3	Computational scheme	40
3	Symbolic Generation	44
3.1	The Symbolic Generators	46
3.2	The <i>CLEAR</i> Post-Processor	53
3.2.1	Removing unused terms	53
3.2.2	Pre-computation of constant terms	54
3.2.3	Renaming of duplicate multiplication terms	56
3.2.4	Trigonometric simplifications	60
3.2.5	Factoring	60
3.2.6	Pre-determined sequence	61
3.2.7	Statistics	61
4	Verifications and Results	65
4.1	Rigid Link Dynamics	65
4.1.1	Inverse Dynamics	66
4.1.2	Closed form dynamics	67

4.1.3	Computational Requirements of Inverse Dynamics	72
4.1.4	Direct Dynamics	79
4.1.5	Computational Requirements of Direct Dynamics	79
4.1.6	Jacobian Matrix	83
4.2	Flexible Link Dynamics	85
4.2.1	Case Studies	85
4.2.2	Verification	110
4.2.3	The Computational Requirements to Calculate the Direct Dynamics for Flexible Link Manipulators	110
5	Summary and Conclusions	118
	References	123
A	Newton-Euler Formulation for Rigid Link Manipulators	129
A.1	Kinematics of the Links	130
A.2	Dynamics of the Links	132

List of Tables

1.1	Computational Requirements of General Purpose Numerical Formulations[42]	9
1.2	Inverse Dynamics Computational Requirements For the Stanford Arm Using Fully Automatic Customized Symbolic Algorithms	14
3.1	List of the symbolic generators	45
3.2	Special cases implemented in the multiplication procedure	48
3.3	Special cases implemented in the addition procedure	49
3.4	The locations of each term in the Fortran code	57
3.5	Common locations for each pair of terms	57
4.1	Inverse Dynamics Computational requirements for Neuman and Murray's ARM generator [29]	76
4.2	Inverse Dynamics Computational Requirements for Six DOF Manipulators Using NEDYN	77
4.3	Inverse Dynamics Computational Requirements for Three DOF Positioning Systems Using NEDYN	78
4.4	Direct Dynamics Computational Requirements for Three DOF Positioning Systems Using NEDYN	81
4.5	Direct Dynamics Computational Requirements for Six DOF Manipulators Using NEDYN	82
4.6	Direct Dynamics Computational requirements for Neuman and Murray's ARM generator [29]	84

4.7	Manipulator Jacobian Matrix Computational Requirements	84
4.8	Link material and geometry parameters used in the test cases	88
4.9	Ratios of the contribution of the first four modes to the static deflection for a link in bending with a point load at the end	88
4.10	Comparison between the theoretical and simulated frequencies for a single flexible link in bending.	88
4.11	Ratios of the contribution of the first four modes to the static deflection for a link in torsion with an external torque at the end	92
4.12	Comparison between the theoretical and simulated frequencies for a single flexible link in torsion.	95
4.13	The definitions of the cases used in the computational efficiency comparisons	112
4.14	The computational requirements of the generator FLEXHB	116
4.15	The computational requirements of the generator FLXDYN	117

List of Figures

- 2.1 Denavit and Hartenberg Parameters 21
- 2.2 Forces and Moments exerted on each link 23
- 2.3 Flowchart for program NEDYN 25
- 2.4 Coordinate Frames and Transformations attached to link i 32

- 3.1 sample segments of the generated Fortran code 52
- 3.2 Fortran code before removing unused terms 55
- 3.3 Fortran code after removing unused terms 55
- 3.4 Fortran code a) before and b) after computations of constant terms 56
- 3.5 Fortran code before renaming of duplicate multiplication terms 58
- 3.6 Fortran code after renaming of duplicate multiplication terms 59
- 3.7 Fortran code a) before and b) after trigonometric simplifications 60
- 3.8 Fortran code a) before and b) after factoring 63
- 3.9 Statistical report produced at the end of *CLEAR* operation 64

- 4.1 Six DOF Stanford manipulator 68
- 4.2 Kinematic State of Joint 2 69
- 4.3 Computed Joint Torques Using Dynamic Code Generated by DYNAM 70
- 4.4 Computed Joint Torques Using Dynamic Code Generated by NEDYN 71
- 4.5 Dynamic Terms Contributions to the Total Torque for Joint 2 of the Stanford Manipulator 73

4.6	Torque for Joint 2 computed by NEDYN	74
4.7	Single flexible link in bending	87
4.8	Response of bending variables of a single flexible link:a) 1st mode b) 2nd mode	89
4.9	Response of bending variables of a single flexible link:a) 3rd mode b) 4th mode	90
4.10	Single flexible link in torsion	92
4.11	Response of torsion variables of a single flexible link: a) 1st mode b) 2nd mode	93
4.12	Response of torsion variables of a single flexible link: a) 3rd mode b) 4th mode	94
4.13	Double flexible compound pendulum	96
4.14	Joint angle responses for double compound flexible pendulum: a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$ c) rigid model	97
4.15	Joint angle responses for double compound flexible pendulum: a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$ c) rigid model	98
4.16	Response of bending variables for link 1 :a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$	99
4.17	Response of bending variables for link 2 :a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$	100
4.18	Two flexible link manipulator	102
4.19	Responses of the manipulator in case 4.1	104
4.20	Responses of the manipulator in case 4.2	106
4.21	Responses of the manipulator in case 4.3	108
4.22	Responses of the manipulator in case 4.4	109

Chapter 1

Introduction

Industrial robots are being increasingly utilized in modern industrial plants. With this growing interest in robot manipulators, robot dynamic analysis has become the target of extensive research in the past few years. This research is aimed at the generation of efficient dynamic equations with the fewest number of arithmetic operations. Efficient robot dynamics are necessary for two main purposes : control and simulation.

When a certain task is required to be performed by the robot, a motion trajectory must be generated to specify the path the robot must follow to execute this task. These trajectories are often generated so as to provide continuity of position, velocity and acceleration. In the case of slow moving robots with constant and small payload, the desired trajectories are programmed in a simple feedback controller. The controller will command the appropriate torque at the joint actuators so that the manipulator can closely follow the planned path. However, the recent trend now is to build faster robots with heavier payloads. For these kinds of robots, simple feedback controllers will not be able to accommodate the dynamic disturbances resulting from

the increased speed and payload. Therefore, more sophisticated controllers which incorporate the robot dynamics into their control schemes are required. For these controllers, an efficient dynamic model is essential because the computations must be performed in real time for on-line control. Furthermore, as the lowest structural resonant frequency of most industrial robots is about 10 Hz, the sampling frequency of the controller must be more than 60 Hz to assure that controller commands will not excite any structural resonances in the manipulator [25, 34].

Simulation plays an important role in the initial design of industrial robots as well as in the planning of tasks for existing robots. During the initial design stage, simulations can provide valuable information regarding the suitability of the structure, the required actuators sizes, the ability of the robot to perform certain tasks under different loading conditions. In addition to this, using simulation, controllers and controlling schemes can be investigated to determine their suitability to perform certain tasks in an acceptable manner. Furthermore, for existing robots and controllers, simulation can provide the expected response of the robot when performing a proposed task, and hence reduce the down time required for programming and testing. Thus, in order to optimize the cost of these simulations, efficient dynamic algorithms are needed.

The dynamic analysis of open chain robot manipulators can be divided into two classes of problem : the inverse dynamics problem and the direct dynamics problem. In the inverse dynamics problem, the joint torques or forces are computed for a set of specified joint positions, velocities and accelerations. The inverse problem is required in the control algorithms. The direct dynamics problem is used to calculate the joint accelerations for specified joint torques or forces, positions and velocities. The direct problem is utilized in the simulation routines where the response of the robot to a set of joint forces or torques is to be determined. Walker and Orin [48] have devised

efficient methods to obtain the direct dynamics from the inverse dynamics. Therefore if an efficient inverse dynamic algorithm is achieved, a direct algorithm can be easily derived from it.

As outlined later in this chapter, several techniques have been implemented to derive the equations of motion. Most of these techniques could be classified as either a Lagrangian method based on Lagrange's equation or a Newton-Euler method based on D'Alembert's principle or they could be a hybrid of the two. Deriving the equations of motion manually using one of these techniques is manageable for manipulators with three or fewer degrees of freedom. For manipulators with a higher number of degrees of freedom, manual derivation becomes very tedious and error prone.

A second alternative to manual derivation is to program these algorithms numerically. Numerical programming has some short falls: the programming itself sometimes becomes so complex that there is an ample opportunity for errors. Furthermore, numerical programming, as will be shown later, is not very efficient in terms of computational requirements. Some of this lack of efficiency is due to the presence of unproductive computations such as multiplication by zero or one, duplicated operations and overlooked simplification opportunities. Also there is some overhead involved in numerical programming such as looping, testing for counters, calling subroutines and passing parameters which causes the slower execution of generic numerical code on computers.

As an alternative to numerical programming, researchers are focusing on the automatic symbolic generation of the equations of motion. The purpose of automatic symbolic generation is to simplify the process of the computer code development and to maximize the efficiency of the produced code. In symbolic generation, mathematical operations involved in the dynamics algorithm are performed utilizing alphanumeric symbols rather than numerical values. For a particular manipulator,

the symbolic generation of its equations of motion is performed only once. The output of these symbolic generation programs (referred to as *generators* throughout the remainder of this work) is a computer code containing the dynamics of the specific manipulator. The generated code is usually called “customized” because it is specifically tailored to calculate the dynamics of a specific manipulator with the minimum number of necessary operations. This customization process greatly improves the efficiency of the dynamic code because all special geometric and mass features of the manipulator are incorporated in the code. These features include the type of joints (translational or rotational), the presence of parallel or intersecting joints and the presence of zero elements in the geometric, mass and inertia parameters.

Robot manipulators can be classified into two categories: the rigid link manipulators and the flexible link manipulators. For a rigid link manipulator, all links and joints are assumed rigid and static deformation and vibration due to different loading conditions are ignored in the dynamic analysis. In fact, this kind assumption is, for all practical purposes, reasonable as most of the present industrial robots are designed with rigid links. These rigid structures reduce the static deformation to very small values. Also, as the rigidity increases, the link vibrations tend to be of high frequencies and small amplitudes. However, there is a price to be paid for this rigidity because as the structure gets heavier, the required actuator torques that are needed to drive the links get larger. This means bigger and heavier actuators and, in turn, larger loads as the actuators are carried by other actuators. In the end, this results in a small useful payload to manipulator mass ratio. Also with the increased masses of links and actuators, speed is compromised. For example, the Excalibur robot has a mass of about 35 kg while its maximum payload capacity is about 3.5 Kg [37]. A more precise manipulator is the Puma 560. This 6 degrees of freedom manipulator has a mass of 55 kg and payload capacity of 2.5 Kg [22]. This results in

a mass to payload ratio of 22 to 1 which indicates that most of the power consumed is wasted just moving the links of the manipulator. With the desire to make lighter and faster robots with higher payload to mass ratio, attention is being made to design new robots that have links with lower structural rigidity and sophisticated control systems to account for structural flexibility. This class of robots is called flexible link manipulators. Light weight manipulators have many advantages over bulky “rigid” link manipulators [45]:

- higher speeds can be achieved.
- less material requirements.
- smaller actuators.
- less power consumption.
- better ability to manoeuvre.
- lower mounting strength and rigidity requirements.
- built-in compliance at the end effector.

However, flexible link manipulators have some serious disadvantages as accuracy in terms of the response of the end effector to the joint controlling commands will be very poor. This poor accuracy is due to the deformation of the links especially when the different loading conditions excite structural oscillations. In order to take advantage of the benefits of this class of manipulators and make them accurate enough for industrial applications, controlling systems must be designed to account for and control the deformations and the vibration modes of the links. Therefore, an efficient and accurate dynamic model is crucial for the purpose of control and also for the purpose of simulation to design the optimum robot configuration and control scheme.

1.1 The Dynamics of Rigid Link Open Chain Robot Manipulators

The dynamic equations of motion are the heart of both control and simulation processes. As it was pointed out, the speed of execution of these dynamic algorithms is of utmost importance.

For an open chain manipulator with N degrees of freedom, the general form of the equations of motion is :

$$\mathbf{H}(\mathbf{q}) \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}(\mathbf{q})^T \mathbf{k} = \boldsymbol{\tau} \quad (1.1)$$

where

- $\mathbf{H}(\mathbf{q})$ $N \times N$ symmetric inertia matrix.
- $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ $N \times N$ matrix specifying centrifugal and coriolis effects.
- $\mathbf{G}(\mathbf{q})$ $N \times 1$ vector specifying gravitational effects.
- $\mathbf{J}(\mathbf{q})$ $6 \times N$ Jacobian matrix specifying the torque (forces) created at each joint due to external load at link N .
- \mathbf{k} 6×1 vector specifying external forces and couples exerted on link N .
- $\boldsymbol{\tau}$ $N \times 1$ vector of actuator torques (forces) at each joint.
- \mathbf{q} $N \times 1$ vector of joint variable (joint angle or offset).
- $\dot{\mathbf{q}}, \ddot{\mathbf{q}}$ $N \times 1$ vector of joint velocity and acceleration respectively.

Several formulations can be utilized to derive the equations of motion. These can be classified into either closed form formulations or recursive formulations [28]. These dynamic formulations have been shown to be equivalent [41], that is, although they possess different formats, they represent exactly the same dynamic behaviour.

The classical Lagrange formulation yields separate kinetic and potential energy terms. The kinetic energy is composed of inertial, centrifugal and coriolis coefficients. The resulting equations separately calculate the different coefficients of equation 1.1 and therefore are referred to as *closed form* dynamic model. A closed form formulation gives better insight into the dynamics of the manipulator and hence it is more useful for designing the control systems.

The recursive formulation involves the implementation of some intermediate quantities to compute the required joint forces or torques. These intermediate quantities sometimes consist of the angular and translational velocities and accelerations and local and total forces and moments. For example in the Newton-Euler formulation, the angular and translational velocities and acceleration (kinematic state) of link i serve as the intermediate quantities which are calculated using the kinematic state of the previous link $i - 1$. Examples of the recursive formulation are the recursive Lagrange formulation [15] and the recursive Newton-Euler formulation [25].

The closed form formulations can be obtained by expanding the recursive formulation; that is by eliminating the intermediate quantities and then extracting the coefficients of the closed form model from the joints torques/forces equations.

1.1.1 Computational requirements of numerical programming

The computational requirements to calculate the inverse dynamics using different numerical techniques, measured by the number of addition and multiplications needed, are shown in table 1.1 [5]. The first entry in the table is Uicker- Kahn Classical Lagrangian formulation. In this formulation (and in all other Lagrangian

formulations), the joint variables q_i are readily utilized as the generalized variables. A coordinate frame is attached to each link. Transformation matrices [5, 34] are used to specify the positions of the different links and the end effector. These transformation matrices are functions of the joint variables q_i . The time derivatives of these transformation matrices specify the motion of the different links. The kinetic and potential energies are formulated in terms of these transformation matrices. As shown in table 1.1, it is clear that the computational requirements of the general purpose Classical Lagrangian formulation developed by Uicker-Kahn are very high as the required number of additions and multiplications are polynomial functions of degree 4 of the number of degrees of freedom. This intensive computational requirement is due to the fact that each joint torque/force is calculated directly and separately from the other joints. In spite of the heavy computational requirements, this closed form formulation is very important as it gives an insight to the dominant dynamic terms for a certain manipulator and hence is useful for the manipulator and controller design engineers.

Due to the extensive computational requirements, simplifying assumptions can be applied to the equations of motion to reduce the intensity of computations. These simplifications include neglecting the coriolis and centrifugal terms, assuming some links to be massless and ignoring some joint offsets. In spite of the resulting reduction in the computational burden due to these assumptions, the dynamic model is not a good representation of the actual system. For example, the coriolis and centrifugal terms are sometimes not negligible even for low speeds [5].

In order to reduce the inefficiency of Uicker-Kahn classical Lagrangian formulation and remove the repetitiveness of computations, Waters [49] re-wrote the Uicker-Kahn algorithm in a more efficient recursive form. Later, this algorithm was refined by Hollerbach [15]. Initially, Hollerbach utilized the standard 4×4 transformation

Table 1.1: Computational Requirements of General Purpose Numerical Formulations[42]

Algorithm	Author	Degrees of Freedom	
		N	6
Classical Lagrange	Uicker-Kahn		
	multiplications	$32N^4 + 86N^3 + 171N^2 + 53N - 128$	67984
	additions	$25N^4 + 66N^3 + 129N^2 + 42N - 96$	51456
Recursive Lagrange	Hollerbach (4×4)		
	multiplications	$830N - 592$	4388
	additions	$675N - 464$	3586
Recursive Lagrange	Hollerbach (3×3)		
	multiplications	$412N - 227$	2195
	additions	$320N - 201$	1719
Newton-Euler	Hollerbach		
	multiplications	$150N - 48$	852
	additions	$131N - 48$	738

matrices and their time derivatives to represent the position and motion of the manipulator. The computational requirements of Hollerbach's 4×4 recursive Lagrangian formulation are shown in table 1.1. It can be seen that this new recursive formulation is a great improvement in efficiency over Uicker-Kahn formulation. The computational requirement has a linear dependency on the number of degrees of freedom. Nevertheless, there are some obvious inefficiencies still present in this algorithm. These inefficiencies arise from the fact that the fourth row of the 4×4 transformation matrix contains only zeros and ones. To overcome this, Hollerbach modified his algorithm by using 3×3 rotational transformation matrix to specify the orientation and a vector to specify the translation. These modifications resulted in great improvement in the efficiency of the algorithm as shown in table 1.1.

The last entry in table 1.1 is the Newton-Euler formulation. As shown in this table, the numerical Newton Euler formulation is the most efficient among all the numerical algorithms. This makes it the most attractive to implement. Silver [41] proved that the recursive Lagrangian formulation and the recursive Newton-Euler formulation are exactly equivalent. What makes the Newton-Euler formulation more efficient is the representation chosen for rotational dynamics. In the Lagrangian formulation, the first and second time derivatives of transformation matrices are used to represent the angular motion of the system. But the transformation matrices have 9 components while only 3 independent components are required to completely specify the orientation of an object. On the other hand, vectors are utilized in the Newton-Euler formation to describe the angular motion which eliminates the redundancies present in the Lagrangian formulation.

1.1.2 Symbolic Generation of the Dynamics

In order to eliminate the unnecessary operations such as multiplications by zero and one or the addition of zero and also to alleviate the computational overhead associated with numerical algorithms, much recent work has been directed at automatically generating the equations of motion utilizing symbolic generation techniques. Besides optimizing the generated codes in terms of efficiency, symbolic generation is an attractive way to simplify the process of code development itself by eliminating tedious manual derivation or numerical programming.

Dillon [12] in 1973 developed the first symbolic generator *OSSAM* which is based on the classical Lagrangian formulation. The main contribution of this generator is its implementation of a symbolic processor designed specifically to address the requirements of robot dynamics. In 1980 Vecchio et al. [46] developed the computational robot dynamics program *DYMIR* to automatically produce the dynamics of manipulators. *DYMIR* is based on the classical Lagrangian formulation and utilizes the symbolic language *REDUCE*. Vecchio et al. used *DYMIR* to generate the complete closed-form dynamics of robots. Luh and Lin [24] in 1981 proposed some simplification procedures by comparing similar algebraic expressions and removing negligible terms. They also proposed the implementation of the recursive Newton-Euler formulation. Unfortunately, these authors did not report the number of computations involved in the generated computer codes. In the early stages of the development of the symbolic generation schemes, concerns were raised that symbolic generation may lead to a greater computational requirement than the numerical programming. Kane and Levinson [18] stated that symbolic generators “tend to lead to computational algorithms involving large numbers of unnecessary arithmetic operations”. They also pointed out the increasing length of the algebraic expressions and the extensive computer memory requirement.

The computational requirements for the inverse dynamics for the Stanford manipulator for different symbolic generators are shown in table 1.2 (part of this data is obtained from Nielan and Kane [32]). Rosenthal and Sherman [38] implemented Kane's method in their symbolic generator. Hussain and Noble utilized the algebraic manipulation system *MACSYMA* to symbolically generate the dynamics based on the classical Lagrange formulation. The drawback of this method was that very long algebraic expressions were produced which resulted in large computational and memory requirements. Due to the apparent inefficiency that accompanies the utilization of general purpose symbolic manipulation languages such as *REDUCE* and *MACSYMA*, researchers are focusing on the development of specialized symbolic manipulation programs for the purpose of generating the dynamics of manipulators. These specialized programs have two advantages: the first is the computer time required to generate the computer codes containing the dynamics is greatly reduced by the elimination of the unnecessary overhead found in the general purpose programs. Secondly, the generated code is more computationally efficient as these specialized programs are designed to take advantage of the simplification opportunities that are inherent in the dynamic equations.

Toogood [42, 43] and Kermack [19] applied Hollerbach's 4×4 recursive Lagrangian formulation [15] in the symbolic generator *DYNAM*. This system has two stages: the dynamic algorithm is implemented in *DYNAM* which produces a unique computer code containing the dynamics of a certain manipulator without any sophisticated simplification. In the second stage the generated code is read by a post processor called *CLEAR* [42] and a series of more advanced simplification procedures are performed on it.

Neuman and ... [28, 29, 30, 31] developed the highly successful symbolic generator *ARM* (A Robot Modeler). Several formulations are implemented in

ARM which include the Newton-Euler formulation among others. Both closed form and recursive form dynamics equations can be generated. *ARM*, which requires a minicomputer to run, consists of two modules: A “composer” which implements the symbolic mathematical operations to produce the dynamic model and a “performer” which performs the operations specified by the composer. By changing the composer, different formulations can be used. The internal algebraic representations can be changed by altering the performer. The composer is written in the *C* programming language and the performer is written in *LISP*.

Recently, Khalil and Kleinfinger [20] implemented a recursive Newton-Euler formulation to produce the inverse dynamics. The resulting dynamics code is very efficient as shown in table 1.2. This high efficiency was attributed partly to the implementation of a new system to describe the manipulators by applying some modification to the Denavit and Hartenberg representation. Furthermore, the recursive Newton- Euler equations were rearranged to allow regrouping of the mass and inertia terms. This permitted the computation of these terms to be performed off-line. The authors did not specify whether these regrouping operation were performed manually or automatically.

1.2 The dynamics of flexible link manipulators

As mentioned earlier, efforts are being directed at the incorporation of link flexibility in the manipulator dynamics. Including flexibility will allow the design of lighter and faster robots with greater useful payloads. The dynamic model of the flexible manipulator consists of two coupled sets of partial differential equations. The first set describes the rigid body motion of links while the second set describes the flexibility of the links. These two sets of partial differential equations are dynamically

Table 1.2: Inverse Dynamics Computational Requirements For the Stanford Arm Using Fully Automatic Customized Symbolic Algorithms

Author	program	year	mult.	add.	total
Rosenthal and Sherman [32]		1983	632	400	1032
Hussain & Noble [17]		1984	5320	1837	7157
Kane and Nielan [18]		1986	772	456	1228
Toogood	DYNAM	1987	331	267	598
Neuman & Murray [29]	ARM	1987	148	185	333
Khalil & Kleinfinger [20]	SYMORO	1987	142	99	241

coupled due to the presence of variable inertia, centrifugal and coriolis terms [2]. Researchers started to consider flexibility as early as 1975 [3]. All these early works were characterized by the implementations of linear models which ignored coupling effects between the rigid and flexible dynamics. Since the demand for light robots is especially important in aerospace applications, early research was aimed at the modelling and control of flexible spacecraft. An example of this is the work done by Singh and Likins [39]. Their model was derived by expanding Kane's method for rigid dynamics. Modal expansion was used to model flexibility.

Recently, more research effort was directed specifically at the dynamic modelling and control of flexible manipulators. Two methods have been used to model the flexible links. These methods are the modal expansion and the finite element method. Sunada and Dubowsky [40] modelled the links of the manipulator using finite element techniques based on the 4×4 recursive Lagrangian formulation. They also employed coordinate reduction techniques to reduce the computational requirements. Usoro [45] also used finite element and Lagrangian methods.

For the previously mentioned methods, no information was reported concerning

the computational requirements of the different formulations.

Book [4] utilized 4×4 transformation matrices to represent the kinematics of both the rotary joint motion and the link deformation. Link deflections were approximated by the summation of assumed modes. Book's algorithm is recursive.

To capitalize on the efficiency of the Newton-Euler formulation, King, Gourishankar and Rink [21, 22] derived the equations of motion using a modified Lagrangian approach. They used angular velocities and 3×3 rotation matrices to represent link kinematics producing a recursive algorithm that is similar to the Newton Euler formulation. Modal expansion was utilized to approximate deflections. They reported that this numerical method is more efficient than Book's [4] method while both methods have the same accuracy as assumptions and approximations were identical.

Similar to rigid link manipulators, symbolic generation of the dynamics of the flexible link manipulators appears to be a good alternative to both numerical programming and manual derivations. When the equations of motion are extended to include the flexibility of the links, the resulting coupled differential equations become too complex to be derived manually. Also, as in the case of rigid link manipulators, the numerical programming incorporates many redundant and unproductive arithmetic operations. Therefore it is logical to seek the aid of symbolic manipulation programs. Most of the work done in symbolic programming has been performed utilizing commercially available symbolic manipulation languages such as *SMP*, *MACSYMA* and *REDUCE*. Examples of this, is the work done by Cetinkunt et al. [6, 7, 8], Nicosia et al. [33] and Tomei [44]. Unfortunately, none of these researchers reported the computational requirements of their algorithms.

Mackay [26] implemented a modified version of Book's algorithm [4] in the symbolic generator *FLXDYN* which is based on the 4×4 recursive Lagrange

formulation and assumed modes. The computational requirements of some cases studied are reported later in this work.

1.3 Outline of this work

In this work, the symbolic generation of inverse and direct dynamics for both rigid link manipulators and flexible link manipulators are treated. The Newton-Euler inverse dynamics formulation of Luh et al. [25] was implemented in the symbolic generator *NEDYN*. This program generates automatically a computer code containing the inverse dynamics of rigid link manipulators. *NEDYN* employs the symbolic manipulation techniques developed by Toogood [42] and Termack [19] for the symbolic generator *DYNAM*. The Newton-Euler formulation was chosen because the numerical Newton-Euler algorithm is one of the most efficient algorithms. It was anticipated therefore that this high efficiency will carry on to the symbolically generated algorithms. *NEDYN* was then modified to symbolically generate the direct dynamics of manipulators utilizing Walker and Orin's technique [48]. *NEDYN* was also modified to symbolically generate the dynamic coefficients found in equation 1.1 such as centrifugal coefficients and coriolis gravitational coefficients. Assembling these coefficients together produced the closed form model of the manipulator. It was also modified to generate the manipulator Jacobian matrix.

The inverse dynamics for flexible link manipulators is programmed in the symbolic generator *FLEX* utilizing King's modified Lagrangian formulation [22]. This formulation utilizes a kinematic representation very similar to the efficient Newton Euler formulation. Additional symbolic generators were derived from *FLEX* to symbolically generate the direct dynamics utilizing Walker and Orin procedures [48].

A post-processor called *CLEAR* is utilized to further optimize the generated

computer code by performing a sequence of simplifying procedures. *CLEAR*, originally developed by Toogood [42], was extensively modified and expanded by the present author.

In chapter two, a very brief outline of the Newton-Euler inverse dynamics formulation of Luh et al. [25] will be presented. Also the Walker and Orin [48] methods to obtain the direct dynamics from the inverse dynamics will be illustrated. The method by which the dynamic coefficients are obtained from the inverse recursive dynamics to formulate the close form dynamics will be explained. King's modified Lagrangian formulation for flexible manipulators will be sketched.

Chapter three will present the technique of symbolic generation, the different generators developed as well as the different simplification techniques as they are implement in the post-processor *CLEAR*.

Chapter four presents the computational requirements of the dynamics generated using the different symbolic generators together with a comparison with other symbolic generators. Some test cases will be presented to verify the correct implementation of the dynamics algorithm in the symbolic generators for the rigid link manipulators. For the flexible link manipulator, some test cases are presented which will indicate the correct implementation of the algorithm in the symbolic generator though they do not rigorously evaluate the accuracy of the original dynamic formulation. Such a verification require comparison with experimental results. Little usable experimental data is available for this.

Finally, chapter five will summarize the findings of this work and present conclusions and recommendations for further work.

Chapter 2

Dynamics of Robot Manipulators

In this chapter, an overview will be presented of the dynamic algorithms implemented in the symbolic generators. Both rigid and flexible link manipulators will be discussed. The Newton-Euler algorithm of Luh et al. [25] for rigid link manipulators is outlined in the first section. This algorithm was implemented in the symbolic generator *NEDYN*. In the second section, the methods of Walker et al. [48] for obtaining the direct dynamics from the inverse dynamics will be presented. The third section presents a new method to deduce the manipulator Jacobian matrix from the inverse dynamics. Algorithms for obtaining other dynamic terms as used in closed form analysis (the inertia, centrifugal and coriolis, gravity, and external force term) are described in the fourth section. In the last section, the main features of the modified Lagrangian algorithm of King et al. [21, 22], implemented in the flexible link dynamic generator *FLEX*, will be discussed. The computational requirements of the various algorithms will be presented in a subsequent chapter, including comparisons with the source algorithms and other symbolic generation schemes.

In this chapter a notation convention is adapted. Vectors are represented by bold

variables, matrices by bold upper case roman letters and other variables are represented by italics. A leading super-script defines the frame of reference and a subscript denotes the link number.

2.1 The Recursive Newton-Euler Inverse Dynamic Algorithm for Rigid Link Manipulators

Referring back to table 1.1 it is apparent that the numerical Newton-Euler algorithm is the most efficient among all the presented numerical formulations. Anticipating that this high efficiency will carry on to symbolic generation, this technique was implemented in the symbolic generator *NEDYN*. The Newton-Euler algorithm implemented is that of Luh, Walker and Paul [25]. In the following sections, the kinematics and dynamics will be discussed.

2.1.1 Kinematics

In this algorithm, a coordinate system is attached to the distal end of each link according to Denavit and Hartenberg convention [34, 11]. Four geometric parameters a_i , d_i , α_i and θ_i are utilized to describe the relative orientation and position of two adjacent link coordinate systems as shown in figure 2.1. The parameter a_i represents the link length, d_i the joint offset, α_i the twist of the link and θ_i the joint angle. Manipulator joints can be classified into two types: revolute (rotational) joints or prismatic (translational) joints. In the case of a revolute joint the variable parameter is the joint angle θ_i and the rest of the parameters are constants while in the case of

a prismatic joint, the variable parameter is the joint offset d_i . The orientation of the coordinate system of link i is specified with respect to the coordinate system of link $i - 1$ by the direction cosine matrix ${}^{i-1}\mathbf{A}_i$:

$${}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i \end{bmatrix} \quad (2.1)$$

while the position of the origin of coordinate system of link i with respect to the origin of the coordinate system of link $i - 1$ is denoted by the vector ${}^i\mathbf{p}_i^*$ as shown in figure 2.2:

$${}^i\mathbf{p}_i^* = \begin{bmatrix} a_i \\ d_i \sin \alpha_i \\ d_i \cos \alpha_i \end{bmatrix} \quad (2.2)$$

The angular and linear velocities, ${}^i\boldsymbol{\omega}_i$ and ${}^i\mathbf{v}_i$, the angular and linear accelerations, ${}^i\dot{\boldsymbol{\omega}}_i$ and ${}^i\dot{\mathbf{v}}_i$ and also the linear acceleration of the center of mass ${}^i\hat{\mathbf{v}}_i$ of the different links are calculated recursively starting at the base link and moving towards the end effector. By giving the base link linear and angular velocities of zero, angular acceleration of zero and linear acceleration equal one g in the z_0 direction, the algorithm includes gravitational effects without explicitly including the weight of each link. The complete kinematics of these links are found in Appendix A.

2.1.2 Dynamics

Equation 1.1 provides the general form of the equation of motion of a manipulator. In the inverse dynamics schemes, the joint forces or moments $\boldsymbol{\tau}_i$ are calculated given

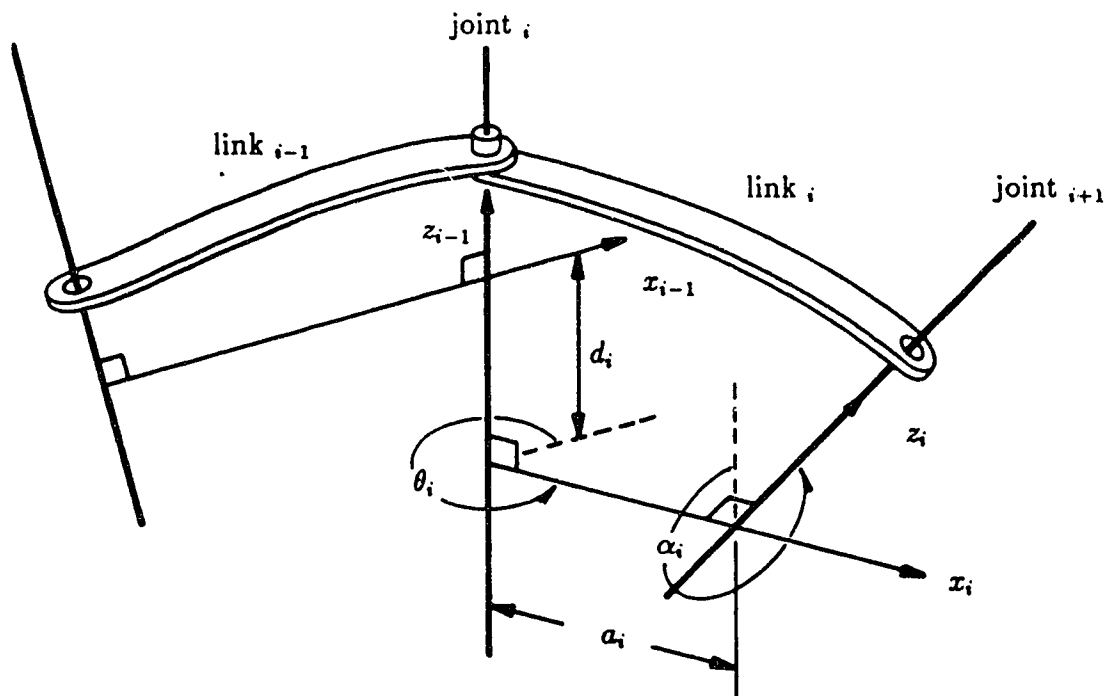


Figure 2.1: Denavit and Hartenberg Parameters

the kinematic state of the manipulator. The Newton-Euler algorithm of Luh et al. [25] is one of the most efficient numerical algorithms to calculate the inverse dynamics. The advantage of Luh et al. formulation over the earlier Newton-Euler formulations is that all the joint forces and torques are referenced in the link's own coordinate frame. According to D'Alembert's principle, the total external vector force ${}^i\mathbf{F}_i$ exerted on link i is given by:

$${}^i\mathbf{F}_i = M_i {}^i\hat{\mathbf{v}}_i \quad (2.3)$$

where M_i is the mass and ${}^i\hat{\mathbf{v}}_i$ is the linear velocity of center of mass of link i . The total external vector moment ${}^i\mathbf{N}_i$ exerted on link i is given by

$${}^i\mathbf{N}_i = {}^i\mathbf{J}_i {}^i\dot{\boldsymbol{\omega}}_i + {}^i\boldsymbol{\omega}_i \times ({}^i\mathbf{J}_i {}^i\boldsymbol{\omega}_i) \quad (2.4)$$

where ${}^i\mathbf{J}_i$ is the inertia matrix of link i about its center of mass referenced to the local coordinate system of the link. Referencing each link's dynamics to its own coordinate system has a great advantage as this eliminates considerable computations that otherwise are necessary to transform all the kinematic and dynamic quantities to the base coordinate system. For example, if the dynamics were reference to the inertial (global) coordinate system, the inertia matrix of the link will change following any change in the configuration of the manipulator during its movement.

Performing equilibrium of forces and moments on link i as shown in figure 2.2 results in

$${}^i\mathbf{f}_i = {}^i\mathbf{A}_{i+1}({}^{i+1}\mathbf{f}_{i+1}) + {}^i\mathbf{F}_i \quad (2.5)$$

$${}^i\mathbf{n}_i = {}^i\mathbf{A}_{i+1}[{}^{i+1}\mathbf{n}_{i+1} + ({}^{i+1}\mathbf{A}_i {}^i\mathbf{p}_i^*) \times {}^{i+1}\mathbf{f}_{i+1}] + ({}^i\mathbf{p}_i^* + {}^i\hat{\mathbf{s}}_i) \times {}^i\mathbf{F}_i + {}^i\mathbf{N}_i \quad (2.6)$$

where ${}^i\mathbf{f}_i$ and ${}^i\mathbf{n}_i$ represent the force and torque vectors respectively exerted on

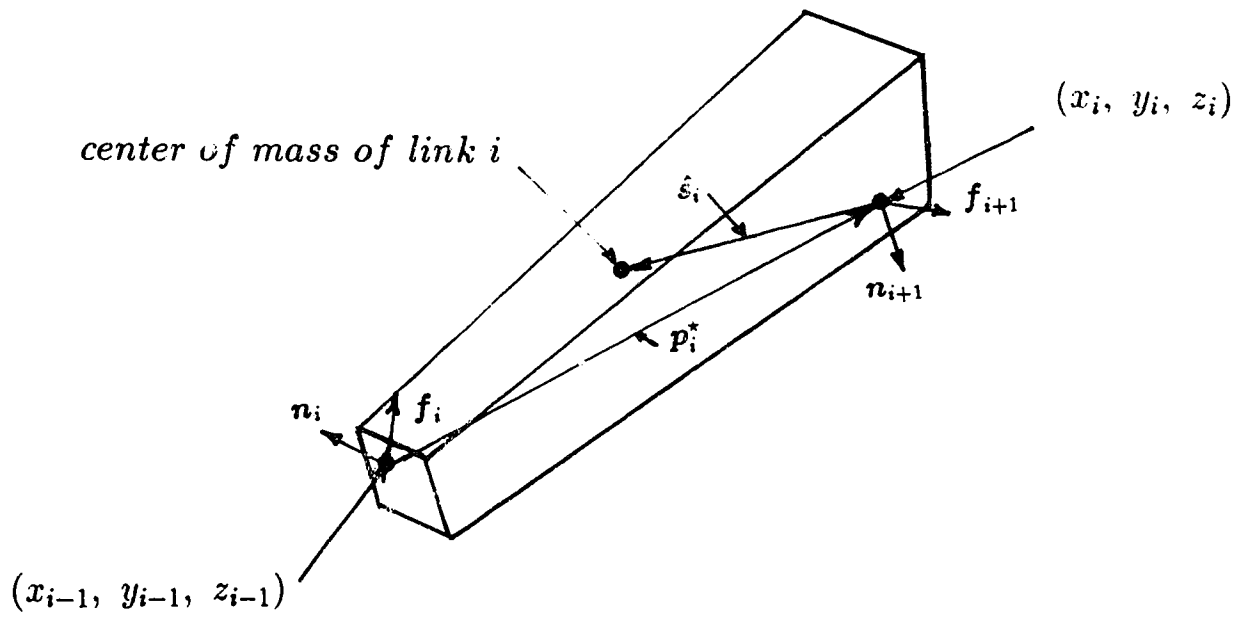


Figure 2.2: Forces and Moments exerted on each link

link i by link $i - 1$. Notice that gravity forces are not included in the balance of forces. Rather, it is included in the kinematic analysis by giving link 0 (base link) an acceleration equal one g upwards.

The resulting joint force or torque τ_i is projection of ${}^i\mathbf{f}_i$ or ${}^i\mathbf{n}_i$ about the \mathbf{z}_{i-1} axis to give

$$\tau_i = \begin{cases} {}^i\mathbf{n}_i^T \cdot ({}^i\mathbf{A}_{i-1} {}^{i-1}\mathbf{z}_{i-1}) & \text{rotational;} \\ {}^i\mathbf{f}_i^T \cdot ({}^i\mathbf{A}_{i-1} {}^{i-1}\mathbf{z}_{i-1}) & \text{translational.} \end{cases} \quad (2.7)$$

A flow-chart of the recursive Newton-Euler algorithm as implemented in *NEDYN* is shown in figure 2.3. The program first generates the link transformation matrices. The forward recursion then generates the kinematic terms beginning at the base link and proceeding outward to the end effector. An external load vector may be applied at the end effector. The backward recursion then proceeds to generate terms representing the interaction forces and moments, and the joint torques until the base link is reached.

2.2 The Direct Dynamics

In the direct dynamic problem, the joint accelerations in equation 1.1 are calculated for specified joint torques or forces, positions and velocities. Four schemes for the direct dynamics solution of 1.1 have been presented by Walker and Orin [48]. Methods 2 and 3 proceed as follows: Equation 1.1 is re-written in the form

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} = \boldsymbol{\tau} - [\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}(\mathbf{q})^T \mathbf{k}] \quad (2.8)$$

or

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} = (\boldsymbol{\tau} - \mathbf{b}) \quad (2.9)$$

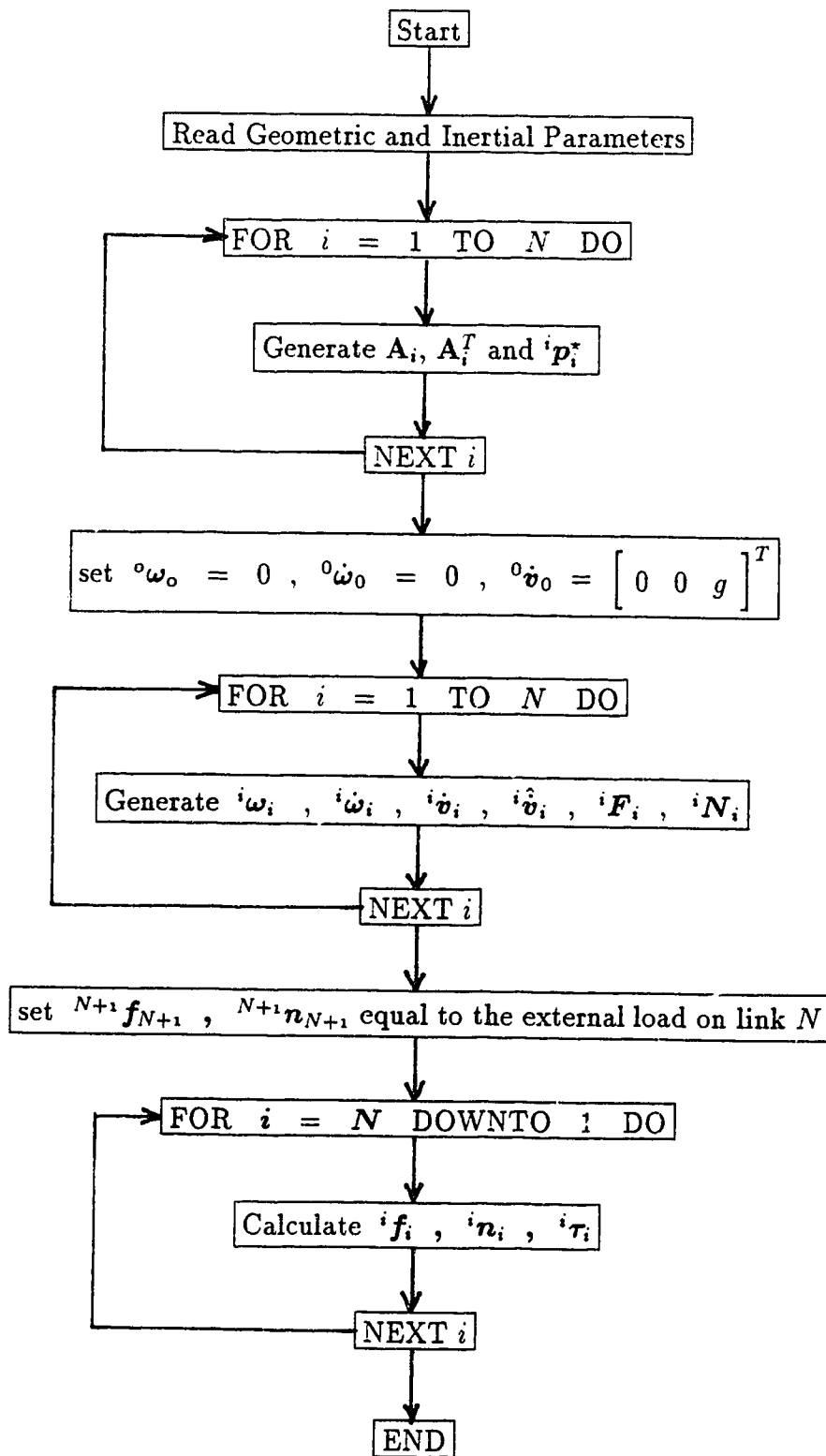


Figure 2.3: Flowchart for program NEDYN

where

$$\mathbf{b} = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{J}(\mathbf{q})^T \mathbf{k} \quad (2.10)$$

is a bias vector. Note that the bias vector can be computed from the inverse dynamics subroutine by setting the joint accelerations to zero.

To compute the inertia matrix, $\mathbf{H}(\mathbf{q})$, all velocity and gravity terms and the payload vector in equation 1.1 are set to zero. Equation 1.1 is then utilized as

$$\mathbf{H}(\mathbf{q}) \mathbf{e}_j = \mathbf{h}_j \quad (2.11)$$

where \mathbf{e}_j is a vector containing all zeros except for row j which contains a one. That is, \mathbf{e}_j is a vector representing a unit acceleration of joint j with all other joint accelerations zero. The result returned by the inverse dynamics is \mathbf{h}_j which is the j^{th} column of the inertia matrix $\mathbf{H}(\mathbf{q})$. Since $\mathbf{H}(\mathbf{q})$ is symmetric, only the terms on and either above or below the diagonal have to be calculated. Walker and Orin's method 2 computes the terms on and below the diagonal. When calculating the j^{th} column of the inertia matrix in this method, the manipulator is considered as having only $N - j + 2$ links with $N - j + 1$ degrees of freedom. Alternatively, Walker and Orin's method 3 computes the terms on and above the diagonal. Method 3 is similar to method 2 in considering only the last $N - j + 2$ links. But in method 3, the last $N - j + 1$ link are regarded as a single rigid body for which a new center of mass and moment of inertia is calculated. With $\mathbf{H}(\mathbf{q})$, \mathbf{b} and $\boldsymbol{\tau}$ known, the system of linear equations can be solved for $\ddot{\mathbf{q}}$. Since $\mathbf{H}(\mathbf{q})$ is a symmetric, positive definite matrix, an efficient algorithm for this solution is Cholesky (LU) decomposition [9].

To produce the direct dynamics it is possible to utilize the inverse dynamic code generated by *NEDYN* and perform the procedures of Walker and Orin's method 2 each time the direct dynamics are required (e.g. each integration step in a simulation program). Nevertheless, it is more efficient to symbolically generate the

direct dynamics as many redundant and unproductive operations can be eliminated. This was done by modifying the inverse dynamics generator *NEDYN* to generate the inertia matrix \mathbf{H} according to either Walker and Orin's method 2 or method 3. The modifications consisted of setting all the velocity and gravity terms to zero and utilizing equation 2.11 and the procedure outlined above to symbolically generate the inertia matrix. The new generator was renamed *HMAT*. Also, *NEDYN* was modified to generate the bias vector \mathbf{b} , by setting the joint accelerations to zero, and renamed *BVECT*. Subsequently, *HMAT* and *BVECT* were combined to form a generator called *HBMAT* which will generate the complete direct dynamics. This merger was performed because the generated codes by *HMAT* and *BVECT* sometimes contain some common operations and hence it is more efficient to combine the generators and the generated codes together.

2.3 The Jacobian Matrix

The Jacobian matrix is very important for the control of manipulators. In order to control the end effector in Cartesian coordinates, the Jacobian matrix and its inverse are required to transform rates and forces from Cartesian coordinates to joint coordinates and visa versa. In order to make it suitable for implementation in real time control schemes, the evaluation of the Jacobian matrix has to be performed on-line and hence very efficient algorithms are needed [34, 36].

It is possible to obtain the Jacobian matrix from the inverse dynamics subroutine in a manner similar to that used to obtain the inertia matrix. This is done by setting all the joint accelerations, velocities and gravity terms to zero in equation 1.1 which can then be re-written as:

$$\mathbf{J}(\mathbf{q})^T \mathbf{e}_i = \mathbf{j}_i \quad (2.12)$$

where \mathbf{e}_i is a (1×6) vector containing all zeros except for row i which contains a one. The result of 2.12 is \mathbf{j}_i which is the i^{th} column of the transpose of the Jacobian matrix $\mathbf{J}(\mathbf{q})^T$ (i^{th} row of $\mathbf{J}(\mathbf{q})$). That is, \mathbf{e}_i is a vector representing a unit external force or moment applied at the end effector in the direction i with all other external forces and moments applied at the end effector set to zero. Indexes 1,2 and 3 represent external forces along \mathbf{x} , \mathbf{y} and \mathbf{z} axes of the end effector respectively, while indexes 4,5 and 6 represent external moments about \mathbf{x} , \mathbf{y} and \mathbf{z} axes respectively. This procedure is repeated six times, with $i = 1 \dots 6$, to evaluate the complete Jacobian matrix.

Calling the inverse dynamics subroutine six times whenever it required to evaluate the Jacobian matrix is not efficient as many duplicate and unproductive operations exist. Therefore it is more efficient to perform the previous process only once. This is done by modifying the inverse dynamics symbolic generator to generate directly a single subroutine for the evaluation of the Jacobian matrix. The computational requirements of the new symbolic generator (named *JACOB*) are discussed in chapter four.

2.4 The Closed Form Dynamics

To formulate the closed form dynamics it is necessary to compute each of the dynamic terms found in equation 1.1 separately. As mentioned earlier, the closed form inverse dynamics, though it is computationally less efficient than the recursive dynamics, is very important to robot design and control engineers as it gives an insight to the dominant dynamic terms for a certain manipulator under given loading conditions. The following dynamic terms are evaluated separately :

- The inertia term $\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}}$

- Centrifugal and coriolis term $C(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}}$
- Gravity term $\mathbf{G}(\mathbf{q})$
- External force term $\mathbf{J}(\mathbf{q})^T \mathbf{k}$

2.4.1 The Inertia Term

The inertia term was basically computed in the direct dynamics. To calculate the inertia term one has only to multiply the inertia matrix $\mathbf{H}(\mathbf{q})$ times the generalized acceleration vector $\ddot{\mathbf{q}}$. Nevertheless, computing the inertia term in this manner is very inefficient because it is computationally intensive to calculate $\mathbf{H}(\mathbf{q})$. Alternatively, the inertia term can be calculated directly from the recursive inverse dynamics by setting all velocity and gravity terms and the payload vector to zero. A more efficient algorithm is produced. *NEDYN* was modified to directly generate the inertia term and was renamed *HTERM*.

2.4.2 Centrifugal and Coriolis Term

The centrifugal and coriolis term is calculated from the recursive inverse dynamics by setting all acceleration and gravity terms and the payload vector in equation 1.1 to zero. *NEDYN* was modified to directly generate the centrifugal and coriolis term and was renamed *CTERM*.

2.4.3 Gravity Term

The gravity term is calculated from the recursive inverse dynamics by setting all acceleration and velocity terms and the payload vector in equation 1.1 to zero.

NEDYN was modified to directly generate the gravity term and was renamed *GTERM*.

2.4.4 External Force Term

The external force term can be calculated by multiplying the transpose of the manipulator Jacobian times payload vector. However it is more efficient to generate it directly from the inverse dynamics by setting all acceleration, velocity and gravity terms 1.1 to zero. Again *NEDYN* was modified to directly generate the external force term and was renamed *KTERM*.

2.5 Inverse Dynamic Algorithm for Flexible Link Manipulator

In the following sections, the dynamic algorithm for flexible link manipulators developed by King [22] will be briefly outlined. The detailed algorithm and its derivations are found in King [22]. Although the efficiency of the Newton-Euler formulation over the different Lagrangian formulations in the case of rigid dynamics is attractive, it is difficult to apply the Newton-Euler approach to flexible links experiencing torsional and flexural vibrations. In order to gain the advantages of both formulations, King utilized the more straightforward Lagrangian formulation to derive the equations of motion while representing the rotational kinematics of the links by angular velocity vectors and coordinate transformations by 3×3 rotational transformation matrices and position vectors. Recursive expressions for the velocities and accelerations were derived using the theories of moving coordinates which is the same process as the Newton-Euler algorithm. Subsequently, the Lagrange's equation

was applied to produce a set of recursive equations which form the non-linear dynamic model of the flexible manipulator. The resulting equations of motion are very similar to the Newton-Euler recursive equations for the rigid link dynamics.

2.5.1 Kinematics of The Flexible Links

To describe the kinematics of a flexible link i , two coordinate frames are utilized as shown in figure 2.4. The first set is the coordinate frame x'_i, y'_i, z'_i which is utilized to describe the gross rigid body motion of the link assuming that the link is not deflected. A convention identical to that of Denavit and Hartenberg used in the kinematics of rigid link manipulators is utilized for flexible link manipulators. The rigid link parameters a_i , d_i , α_i and θ_i are employed to describe the relative position and orientation of the origin of this "rigid" frame with respect to the origin of the coordinate frame $x_{i-1}, y_{i-1}, z_{i-1}$ of link $i - 1$ by the vector p_i^* and direction cosine matrix ${}^{i-1}A_i$. Only revolute joints were considered both here and in King's algorithm [22].

The second set is the coordinate frame x_i, y_i, z_i . This frame is attached to the end of the deflected link and constitutes the true coordinate frame of the link. To describe the deflections of the link along its axis and also to describe the position and orientation of the "true" coordinate frame relative to the "rigid" coordinate frame, additional kinematic parameters are required. These parameters consist of the instantaneous flexural and torsional deflections of the link along its axis. The lateral deflections in the transverse y'_i and z'_i directions of the link i at a distance η from its proximal end and at time t are represented by $w_{y_i}(\eta, t)$ and $w_{z_i}(\eta, t)$ respectively. In this algorithm, longitudinal vibrations in the x'_i direction are neglected. $\phi_{y_i}(\eta, t)$ and $\phi_{z_i}(\eta, t)$ represent the rotational angles of the link cross-section about y'_i and z'_i axes

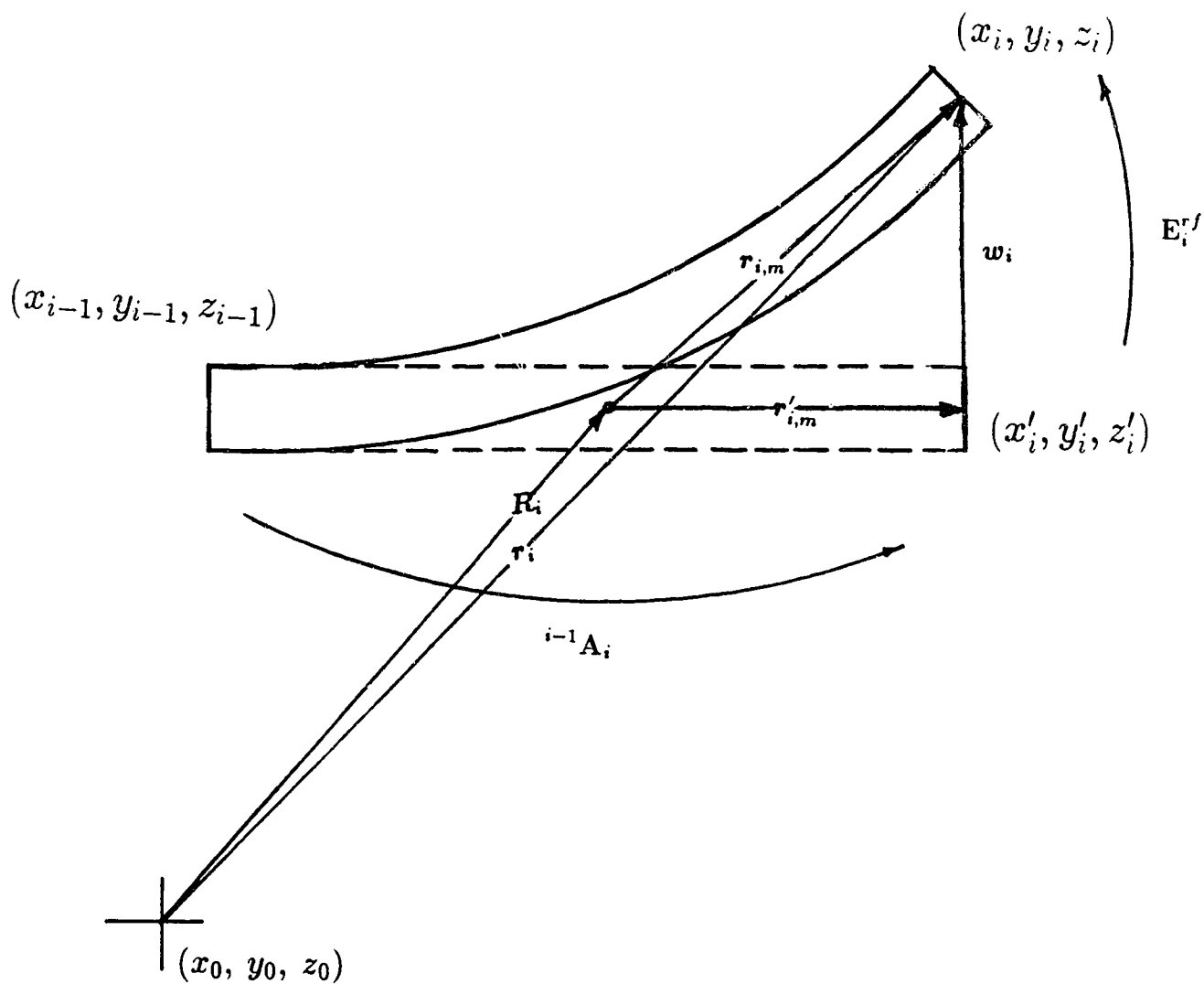


Figure 2.4: Coordinate Frames and Transformations attached to link i

due to the link i bending deflections. $\phi_{xi}(\eta, t)$ represents the angle of torsion about the x'_i axis.

A homogenous transformation matrix \mathbf{E}_i^{rf} is utilized to describe the orientations of the origin of the x_i, y_i, z_i frame relative to the origin of the "rigid" x'_i, y'_i, z'_i frame. If it is assumed that the angles of rotation due to link deflections are small so that

$$\sin(\phi) \approx \phi$$

and

$$\cos(\phi) \approx 1$$

then the following transformation matrix is obtained

$$\mathbf{E}_i^{rf} = \begin{bmatrix} 1 & -\phi_{zi} & \phi_{yi} \\ \phi_{zi} & 1 & -\phi_{xi} \\ -\phi_{yi} & \phi_{xi} & 1 \end{bmatrix} \quad (2.13)$$

where

$$\phi_{xi} = \phi_{xi}(a_i, t)$$

$$\phi_{yi} = \phi_{yi}(a_i, t)$$

$$\phi_{zi} = \phi_{zi}(a_i, t)$$

The transformation that defines the orientation of the end effector coordinate frame x_N, y_N, z_N with respect to the base can be written in the form :

$$\mathbf{T}_N = {}^0\mathbf{A}_1 \mathbf{E}_1^{rf} {}^1\mathbf{A}_2 \mathbf{E}_2^{rf} \dots {}^{N-1}\mathbf{A}_N \mathbf{E}_N^{rf} \quad (2.14)$$

In this algorithm, the assumed modes method is used to calculate the deflections of the link. The total flexural and torsional deflections are approximated by the

finite summation of products of m_i time varying generalized variables $\delta(t)$ and space dependent shape functions $\Delta(\eta)$ and $\Phi(\eta)$ which can be written as:

$$\phi_{xi}(\eta, t) = \sum_{k=1}^{m_i} \delta_{xik}(t) \Phi_{xik}(\eta) \quad (2.15)$$

$$\phi_{yi}(\eta, t) = - \sum_{k=1}^{m_i} \delta_{zik}(t) \Phi_{yik}(\eta) \quad (2.16)$$

$$\phi_{zi}(\eta, t) = \sum_{k=1}^{m_i} \delta_{yik}(t) \Phi_{zik}(\eta) \quad (2.17)$$

$$w_{yi}(\eta, t) = \sum_{k=1}^{m_i} \delta_{yik}(t) \Delta_{yik}(\eta) \quad (2.18)$$

$$w_{zi}(\eta, t) = \sum_{k=1}^{m_i} \delta_{zik}(t) \Delta_{zik}(\eta) \quad (2.19)$$

or

$${}^i w_i = \sum_{k=1}^{m_i} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \delta_{yik} & 0 \\ 0 & 0 & \delta_{zik} \end{bmatrix} \begin{bmatrix} 0 \\ \Delta_{yik} \\ \Delta_{zik} \end{bmatrix} \quad (2.20)$$

$${}^i \phi_i = \sum_{k=1}^{m_i} \begin{bmatrix} \delta_{xik} & 0 & 0 \\ 0 & \delta_{yik} & 0 \\ 0 & 0 & \delta_{zik} \end{bmatrix} \begin{bmatrix} \Phi_{xik} \\ -\Phi_{yik} \\ \Phi_{zik} \end{bmatrix} \quad (2.21)$$

where $\Phi_{xik}(\eta)$, $\Phi_{yik}(\eta)$, $\Phi_{zik}(\eta)$, $\Delta_{yik}(\eta)$ and $\Delta_{zik}(\eta)$ are k^{th} mode shape function of the link i , m_i is the number of assumed modes per direction of flexibility and δ_{xik} , δ_{yik} and δ_{zik} are the generalized time varying coordinates.

For the “rigid” coordinate frames x'_i , y'_i , z'_i , recursive expressions for the linear and rotational velocities ${}^0 v_i^r$ and ${}^0 \omega_i^r$ and the linear and rotational accelerations ${}^0 \dot{v}_i^r$ and ${}^0 \dot{\omega}_i^r$ are written which calculate these quantities serially starting at the base link and proceeding to the last link. Similar expressions are written to calculate the linear and

rotational velocities ${}^0\boldsymbol{v}_i$ and ${}^0\boldsymbol{\omega}_i$; and also the linear and rotational accelerations ${}^0\dot{\boldsymbol{v}}_i$ and ${}^0\dot{\boldsymbol{\omega}}_i$. The derivation of these expressions is similar to the corresponding derivation in the Newton-Euler rigid dynamic formulations with the inclusion of the parameters describing the flexibility of the links. As an example, the expressions to calculate ${}^i\dot{\boldsymbol{v}}_i^r$ and ${}^i\dot{\boldsymbol{v}}_i$ are

$$\begin{aligned} {}^i\dot{\boldsymbol{v}}_i^r &= {}^i\mathbf{A}_{i-1} [{}^{i-1}\dot{\boldsymbol{v}}_{i-1}] + {}^i\dot{\boldsymbol{\omega}}_i^r \times {}^i\boldsymbol{p}_i^* + {}^i\boldsymbol{\omega}_i^r \times ({}^i\boldsymbol{\omega}_i^r \times {}^i\boldsymbol{p}_i^*) \\ {}^i\dot{\boldsymbol{v}}_i &= \mathbf{E}_i^{rfT} [{}^i\dot{\boldsymbol{v}}_i^r + {}^i\dot{\boldsymbol{\omega}}_i^r \times {}^i\boldsymbol{w}_i + {}^i\boldsymbol{\omega}_i^r \times ({}^i\boldsymbol{\omega}_i^r \times {}^i\boldsymbol{w}_i) + 2({}^i\boldsymbol{\omega}_i^r \times {}^i\dot{\boldsymbol{w}}_i) + {}^i\ddot{\boldsymbol{w}}_i] \end{aligned}$$

where

$${}^i\dot{\boldsymbol{w}}_i = \sum_{k=1}^{m_i} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \dot{\delta}_{yik} & 0 \\ 0 & 0 & \dot{\delta}_{zik} \end{bmatrix} \begin{bmatrix} 0 \\ \Delta_{yik} \\ \Delta_{zik} \end{bmatrix} \quad (2.22)$$

and

$${}^i\ddot{\boldsymbol{w}}_i = \sum_{k=1}^{m_i} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \ddot{\delta}_{yik} & 0 \\ 0 & 0 & \ddot{\delta}_{zik} \end{bmatrix} \begin{bmatrix} 0 \\ \Delta_{yik} \\ \Delta_{zik} \end{bmatrix} \quad (2.23)$$

where ($\dot{}$) and ($\ddot{}$) denote the first and second time derivatives as seen in the rigid coordinate frame. Notice that the equation which calculates ${}^i\dot{\boldsymbol{v}}_i^r$ is identical to the expression used in the Newton-Euler formulation for rigid link manipulator (refer to Appendix A). The complete set of the expressions to calculate the kinematics are found in reference [22].

2.5.2 Dynamics of the Flexible Links

In King's algorithm, it was assumed that the links are slender with high aspect ratios. These links can be considered as Euler-Bernoulli beams which (in the absence

of external transverse loads) comply to the following partial differential equation:

$$\frac{\partial^2}{\partial \eta^2} \left(EI(\eta) \frac{\partial^2 \mathbf{w}(\eta, t)}{\partial \eta^2} \right) + \mu(\eta) \frac{\partial^2 \mathbf{w}(\eta, t)}{\partial t^2} = 0 \quad (2.24)$$

where

- $\mathbf{w}(\eta, t)$ the lateral deflection of the link at time t
at a distance η from its proximal end along its axis
- $EI(\eta)$ link's flexural stiffness
- $\mu(\eta)$ mass density (mass per unit length at location η)

The mode shape functions used in the present work are the eigenfunctions of equation 2.24 assuming clamped-free boundary conditions. The choice of the appropriate boundary conditions that applies to the links of manipulator is a very difficult task complicated by the fact that these boundary conditions change during the motion of the manipulator. Nevertheless, a number of researchers have indicated that using clamped-free boundary condition resulted in a better approximation in the modelling of flexible manipulators. An example of this is the work performed by Hasting and Book [14] in which results from experimental data obtained from a flexible arm were compared to data obtained from a linear dynamic computer model utilizing both clamped-free and pinned-free boundary conditions. They concluded that the application of clamped free boundary condition is far more accurate than the pinned-free boundary condition. Furthermore, King [22] showed that results using the finite element method to model deflections gave results very close to those using the assumed modes with clamped free boundary condition.

For the torsional deflections, the following partial differential equation is utilized:

$$\frac{\partial^2 \phi_x(\eta, t)}{\partial t^2} = \frac{G_i}{\rho_i} \frac{\partial^2 \phi_x(\eta, t)}{\partial \eta^2} \quad (2.25)$$

where

- $\phi_x(\eta, t)$ the angular displacement of the link about the x axis
at time t and at location η from its proximal end,
 G_i link's shear modulus,
 ρ_i mass per unit volume.

The eigenfunctions of equation 2.25 assuming also a clamped-free boundary condition were used in the present work.

To derive the equation of motion, the Lagrangian approach is applied. Referring to figure 2.4, the kinetic energy of the link i can be written as follows:

$$\mathbf{KE}_i = \frac{1}{2} \int \dot{\mathbf{r}}_i \cdot \dot{\mathbf{r}}_i dm + \frac{1}{2} \int \rho I_{xi} \dot{\phi}_{xi} \cdot \dot{\phi}_{xi} d\eta \quad (2.26)$$

where I_{xi} is the polar moment of inertia of the links's cross section area about the x' axis. Noting that

$$\begin{aligned} \mathbf{r}_i &= \mathbf{R}_i + \mathbf{r}_{i,m} \\ \mathbf{r}_{i,m} &= \mathbf{r}'_{i,m} + \mathbf{w}_i \end{aligned}$$

and defining the following parameters:

$$\mathbf{e}_{ik} = \int \Delta_{ik} dm \quad (2.27)$$

$$\begin{aligned} \mathbf{c}_i &= \int {}^0\mathbf{w}_i dm \\ &= \sum_{k=1}^{m_i} [\delta_{ik}] \mathbf{e}_{ik} \end{aligned} \quad (2.28)$$

$$\begin{aligned} \dot{\mathbf{c}}_i &= \int {}^0\dot{\mathbf{w}}_i dm \\ &= \sum_{k=1}^{m_i} [\dot{\delta}_{ik}] \mathbf{e}_{ik} \end{aligned} \quad (2.29)$$

$$\dot{\mathbf{h}}_i = \int (\mathbf{r}'_{i,m} \times {}^0 \dot{\mathbf{w}}_i) dm \quad (2.30)$$

$$\dot{\mathbf{a}}_i = \int ({}^0 \mathbf{w}_i \times {}^0 \dot{\mathbf{w}}_i) dm \quad (2.31)$$

$$\begin{aligned} \dot{\mathbf{b}}_i = & \frac{1}{2} \int ({}^0 \dot{\mathbf{w}}_i \cdot {}^0 \dot{\mathbf{w}}_i) dm + \\ & \frac{1}{2} I_{xi} \rho \int \dot{\phi}_{xi} \cdot \dot{\phi}_{xi} d\eta \end{aligned} \quad (2.32)$$

After some manipulations the kinetic energy can be written as follows:

$$\begin{aligned} \mathbf{KE}_i = & \frac{1}{2} M_i ({}^0 \dot{\hat{\mathbf{v}}}_i^r \cdot {}^0 \dot{\hat{\mathbf{v}}}_i^r) + \frac{1}{2} ({}^0 \boldsymbol{\omega}_i^r \cdot \mathbf{I}_i {}^0 \boldsymbol{\omega}_i^r) + \\ & \dot{\mathbf{b}}_i + {}^0 \dot{\hat{\mathbf{v}}}_i^r \cdot [({}^0 \boldsymbol{\omega}_i^r \times \mathbf{c}_i) + \dot{\mathbf{c}}_i] + {}^0 \boldsymbol{\omega}_i^r \cdot (\dot{\mathbf{h}}_i + \dot{\mathbf{a}}_i) \end{aligned} \quad (2.33)$$

where \mathbf{I}_i is the link inertia tensor which can be calculated as follows:

$$\mathbf{I}_i^r = \text{trace}(\mathbf{J}_i^r) \mathbf{X} - \mathbf{J}_i^r \quad (2.34)$$

$$\mathbf{I}_i^{r,f} = (\text{trace}(\mathbf{J}_i^{r,f}) \mathbf{X} - \mathbf{J}_i^{r,f}) + (\text{trace}(\mathbf{J}_i^{r,f}) \mathbf{X} - \mathbf{J}_i^{r,f})^T \quad (2.35)$$

$$\mathbf{I}_i^f = \text{trace}(\mathbf{J}_i^f) \mathbf{X} - \mathbf{J}_i^f \quad (2.36)$$

$$\mathbf{I}_i = \mathbf{I}_i^r + \mathbf{I}_i^{r,f} + \mathbf{I}_i^f \quad (2.37)$$

where

\mathbf{X} = identity tensor

$$\mathbf{J}_i^r = \int [\mathbf{r}'_{i,m}] \cdot [\mathbf{r}'_{i,m}]^T dm \quad (2.38)$$

$$\mathbf{J}_i^{r,f} = \int [\mathbf{r}'_{i,m}] \cdot [{}^0 \mathbf{w}_i]^T dm \quad (2.39)$$

$$\mathbf{J}_i^f = \int [{}^0 \mathbf{w}_i] \cdot [{}^0 \mathbf{w}_i]^T dm \quad (2.40)$$

The total kinetic energy of the manipulator will then be

$$\mathbf{KE}_{total} = \sum_i \mathbf{KE}_i \quad (2.41)$$

The potential energy of link i will take the form

$$\mathbf{PE}_i = \frac{1}{2} \int \left\{ G_i I_{xi} \left(\frac{\partial \phi_{xi}}{\partial \eta} \right)^2 + E_i I_{yi} \left(\frac{\partial \phi_{yi}}{\partial \eta} \right)^2 + E_i I_{zi} \left(\frac{\partial \phi_{zi}}{\partial \eta} \right)^2 \right\} d\eta \quad (2.42)$$

Notice that the gravitational potential energy is not included in the previous expression but accounted for, as in the Newton-Euler algorithm, by giving the base of the first link an upward acceleration equal to $1 g$.

The total elastic potential energy of the manipulator will be

$$\mathbf{PE}_{total} = \sum_i \mathbf{PE}_i \quad (2.43)$$

Lagrange's equations are then applied to obtain the equations of motion. This results in two sets of coupled differential equations. The first set governs the gross rigid body motion of the manipulator's joints. Knowing that the elastic potential energy is independent of the generalized joint variables q_j , the first set can be obtained from

$$\frac{d}{dt} \left[\frac{\partial \mathbf{KE}_{total}}{\partial \dot{q}_j} \right] - \frac{\partial \mathbf{KE}_{total}}{\partial q_j} = \tau_j^r \quad (2.44)$$

The second set describes the vibrations of the links and is obtained by

$$\frac{d}{dt} \left[\frac{\partial \mathbf{KE}_{total}}{\partial \dot{\delta}_{yik}} \right] - \frac{\partial \mathbf{KE}_{total}}{\partial \delta_{yik}} + \frac{\partial \mathbf{PE}_{total}}{\partial \delta_{yik}} = \tau_{yik} \quad (2.45)$$

where δ_{yik} is the k^{th} generalized flexibility variable in the transverse \mathbf{y} direction. Similar expressions are written to calculate τ_{zik} and τ_{xik} . By the proper choice of the modal functions, these generalized deflection forces will be zero.

After substituting the expressions for the kinetic energy and potential energy in equation 2.44 and manipulating of terms as described in King [22], the Lagrangian

equation for the joint can be written as follows:

$$\begin{aligned}
\tau_j^r = & \sum_{i=j}^N \left\{ \left[\frac{\partial^0 \hat{\mathbf{v}}_i^r}{\partial \dot{q}_j} \right] \cdot \left[M_i^0 \hat{\mathbf{v}}_i^r + {}^0 \dot{\boldsymbol{\omega}}_i^r \times \mathbf{c}_i + \right. \right. \\
& {}^0 \boldsymbol{\omega}_i^r \times ({}^0 \boldsymbol{\omega}_i^r \times \mathbf{c}_i) + 2 ({}^0 \boldsymbol{\omega}_i^r \times \dot{\mathbf{c}}_i) + \ddot{\mathbf{c}}_i \left. \right] + \\
& \left[\frac{\partial^0 \boldsymbol{\omega}_i^r}{\partial \dot{q}_j} \right] \cdot \left[\mathbf{I}_i {}^0 \dot{\boldsymbol{\omega}}_i^r + \dot{\mathbf{I}}_i {}^0 \boldsymbol{\omega}_i^r + {}^0 \boldsymbol{\omega}_i^r \times \mathbf{I}_i {}^0 \boldsymbol{\omega}_i^r + \right. \\
& \left. \mathbf{c}_i \times {}^0 \hat{\mathbf{v}}_i^r + {}^0 \boldsymbol{\omega}_i^r \times (\dot{\mathbf{h}}_i + \dot{\mathbf{a}}_i) + \ddot{\mathbf{h}}_i + \ddot{\mathbf{a}}_i \right] \left. \right\} \quad (2.46)
\end{aligned}$$

and the equation of flexibility is written in the following way:

$$\begin{aligned}
\tau_{yik} = & \sum_{i=j}^N \left\{ \left[\frac{\partial^0 \hat{\mathbf{v}}_i^r}{\partial \dot{\delta}_{yik}} \right] \cdot \left[M_i^0 \hat{\mathbf{v}}_i^r + {}^0 \dot{\boldsymbol{\omega}}_i^r \times \mathbf{c}_i + \right. \right. \\
& {}^0 \boldsymbol{\omega}_i^r \times ({}^0 \boldsymbol{\omega}_i^r \times \mathbf{c}_i) + 2 {}^0 \boldsymbol{\omega}_i^r \times \dot{\mathbf{c}}_i + \ddot{\mathbf{c}}_i \left. \right] \\
& + \left[\frac{\partial^0 \boldsymbol{\omega}_i^r}{\partial \dot{\delta}_{yik}} \right] \cdot \left[\mathbf{I}_i {}^0 \dot{\boldsymbol{\omega}}_i^r + \dot{\mathbf{I}}_i {}^0 \boldsymbol{\omega}_i^r + {}^0 \boldsymbol{\omega}_i^r \times \mathbf{I}_i {}^0 \boldsymbol{\omega}_i^r + \right. \\
& \left. \mathbf{c}_i \times {}^0 \hat{\mathbf{v}}_i^r + {}^0 \boldsymbol{\omega}_i^r \times (\dot{\mathbf{h}}_i + \dot{\mathbf{a}}_i) + \ddot{\mathbf{h}}_i + \ddot{\mathbf{a}}_i \right] \\
& \left. + \sigma_{yik} \right\} \quad (2.47)
\end{aligned}$$

where

$$\begin{aligned}
\sigma_{yik} = & {}^0 \hat{\mathbf{v}}_i^r \cdot \frac{\partial \dot{\mathbf{c}}_i}{\partial \dot{\delta}_{yik}} + {}^0 \dot{\boldsymbol{\omega}}_i^r \cdot \left[\frac{\partial \dot{\mathbf{h}}_i}{\partial \dot{\delta}_{yik}} + \frac{\partial \dot{\mathbf{a}}_i}{\partial \dot{\delta}_{yik}} \right] \\
& + {}^0 \boldsymbol{\omega}_i^r \cdot \left[\frac{d}{dt} \left(\frac{\partial \dot{\mathbf{a}}_i}{\partial \dot{\delta}_{yik}} \right) - \frac{\partial \dot{\mathbf{a}}_i}{\partial \dot{\delta}_{yik}} \right] + \frac{d}{dt} \left(\frac{\partial \dot{\mathbf{b}}_i}{\partial \dot{\delta}_{yik}} \right) \\
& - \frac{1}{2} {}^0 \boldsymbol{\omega}_i^r \cdot \frac{\partial \mathbf{I}_i}{\partial \dot{\delta}_{yik}} {}^0 \boldsymbol{\omega}_i^r + \sum_{l=1}^{m_i} \delta_{yil} K_{yilk} \quad (2.48)
\end{aligned}$$

2.5.3 Computational scheme

The inertial forces and moments are given by:

$${}^i \mathbf{F}_i = M_i^i \hat{\mathbf{v}}_i^r + {}^i \dot{\boldsymbol{\omega}}_i^r \times {}^i \mathbf{c}_i + {}^i \boldsymbol{\omega}_i^r \times ({}^i \boldsymbol{\omega}_i^r \times {}^i \mathbf{c}_i) + 2 ({}^i \boldsymbol{\omega}_i^r \times \dot{\mathbf{c}}_i) + \ddot{\mathbf{c}}_i \quad (2.49)$$

$$\begin{aligned}
{}^i N_i &= {}^i I_i \dot{\omega}_i^r + \dot{I}_i \omega_i^r + ({}^i \omega_i^r \times {}^i I_i \omega_i^r) + {}^i c_i \times {}^i \hat{v}_i^r \\
&\quad + {}^i \omega_i^r \times ({}^i h_i + {}^i \dot{a}_i) + {}^i F_i \bar{a}_i
\end{aligned} \tag{2.50}$$

Equations 2.49 and 2.50 are deduced from equation 2.46 following the procedures described in Silver [41].

As in the Newton-Euler rigid dynamics, the interaction forces and moments between links can be expressed in the following way:

$${}^i f_i = E_i^{rf} {}^i A_{i+1} {}^{i+1} f_{i+1} + {}^i F_i \tag{2.51}$$

$$\begin{aligned}
{}^i n_i &= E_i^{rf} {}^i A_{i+1} {}^{i+1} n_{i+1} + [{}^i p_i^* + {}^i w_i] \times E_i^{rf} {}^i A_{i+1} {}^{i+1} f_{i+1} + [{}^i p_i^* + {}^i \hat{s}_i] \times {}^i F_i + {}^i N_i
\end{aligned} \tag{2.52}$$

and the joint torques are given by

$$\tau_i = z_0 \cdot {}^{i-1} A_i {}^i n_i \tag{2.53}$$

The flexibility equations can be deduced in a similar way from equation 2.47 to give:

$$\tau_{xik} = \sigma_{xik} + \Phi_{xik} \cdot E_i^{rf} {}^i A_{i+1} {}^{i+1} n_{i+1} \tag{2.54}$$

$$\tau_{yik} = \sigma_{yik} + \Phi_{yik} \cdot E_i^{rf} {}^i A_{i+1} {}^{i+1} n_{i+1} + \Delta_{yik} \cdot E_i^{rf} {}^i A_{i+1} {}^{i+1} f_{i+1} \tag{2.55}$$

$$\tau_{z ik} = \sigma_{z ik} + \Phi_{z ik} \cdot E_i^{rf} {}^i A_{i+1} {}^{i+1} n_{i+1} + \Delta_{z ik} \cdot E_i^{rf} {}^i A_{i+1} {}^{i+1} f_{i+1} \tag{2.56}$$

where

$$\Phi_{xik} = \begin{bmatrix} \Phi_{xik} \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned}
\Phi_{yik} &= \begin{bmatrix} 0 \\ \Phi_{yik} \\ 0 \end{bmatrix} \\
\Phi_{zik} &= \begin{bmatrix} 0 \\ 0 \\ \Phi_{zik} \end{bmatrix} \\
\Delta_{yik} &= \begin{bmatrix} 0 \\ \Delta_{yik} \\ 0 \end{bmatrix} \\
\Delta_{zik} &= \begin{bmatrix} 0 \\ 0 \\ \Delta_{zik} \end{bmatrix}
\end{aligned}$$

Detailed derivations and proofs of this algorithm are given in King [22].

This algorithm was implemented in the symbolic generator *FLEX* which generates the inverse dynamics of flexible link manipulators. *FLEX* was then modified to generate the direct dynamics according to Walker and Orin [48] method 2 described earlier. This resulted in two more generators : *FLEXH* which generates the inertia matrix and *FLEXB* which generates the bias vector. Subsequently, *FLEXH* and *FLEXB* were combined in one generator *FLEXHB* which generates the complete direct dynamics.

In this chapter a brief description of the dynamic algorithms implemented in

the symbolic generators was presented. The computational efficiencies and the verification of the symbolic generators implementing these algorithms are given in chapter four.

Chapter 3

Symbolic Generation

As a part of this work, several symbolic generation programs were developed to automatically generate expressions, terms and relations involved in the dynamic analysis and simulation of robot manipulators. The output of these programs is a unique compiler-ready Fortran code representing one of several possible algorithms for the manipulator under study. A list of these generating programs is found in table 3.1. These programs are written in Pascal and run on a microcomputer. Since the entire process is automatic, the required user input is only the geometric and inertia parameters of the manipulator. The generation process is very fast, taking between 1 to 5 seconds¹ to generate the code for the inverse or direct dynamics of a manipulator. In this chapter, the mechanism of symbolic generation and how it is employed to generate the equations of dynamics for robots will be explained. Also, the algorithms that perform the post-processing to optimize the generated code as implemented in *CLEAR* will be presented.

¹depending on the complexity of the manipulator, the type of problem whether direct or inverse, and the presence of flexible links

Table 3.1: List of the symbolic generators

Program	Algorithm	Dynamic Model		Generated Dynamics	Formulation
		Flexible Links	Rigid Links		
NEDYN	Newton-Euler		•	inverse dynamics	recursive
HMAT	Newton-Euler		•	inertia matrix	recursive
BVECT	Newton-Euler		•	bias vector	recursive
HBMAT	Newton-Euler		•	inertia matrix + bias vector	recursive
GTERM	Newton-Euler		•	gravity term	closed form
HTERM	Newton-Euler		•	inertia term	closed form
CTERM	Newton-Euler		•	centrifugal & coriolis term	closed form
KTERM	Newton-Euler		•	external force term	closed form
JACOB	Newton-Euler		•	Jacobian matrix	
FLEX	Lagrangian	•	•	inverse dynamics	recursive
FLEXH	Lagrangian	•	•	inertia matrix	recursive
FLEXB	Lagrangian	•	•	bias vector	recursive
FLEXHB	Lagrangian	•	•	inertia matrix + bias vector	recursive

3.1 The Symbolic Generators

A library of symbolic manipulation procedures was developed. These procedures perform the algebraic operations that are required in the dynamic analysis of robot manipulators. In addition to simple arithmetic operations such as addition, subtraction and multiplication of two quantities, these operations involve some more complicated operations such as vector and matrix algebra. This library is shared by all the generation programs. Using these procedures, the Newton-Euler inverse dynamics algorithm [25] is programmed in the generator *NEDYN*, the Newton-Euler direct dynamics algorithm [48] is programmed in the generators *HMAT*, *BVECT* and *HBMAT*, and the closed form formulation where the dynamic terms are generated separately is generated by the generators *GTERM*, *HTERM*, *CTERM* and *KTERM*. The Lagrangian algorithm in a Newton-Euler like formulation for a flexible link manipulator [22] is programmed in *FLEX* (inverse) and *FLEXB*, *FLEXH* and *FLEXHB* (direct) generators.

For all the generators, the output is Fortran code having a very simple structure which is only a series of assignment statements (see sample code in figure 3.1). Conversion of this code to any other computer languages is easily accomplished using a standard text editor or filter program.

The “customized” code which is unique to the manipulator being studied, is generally more efficient in terms of computational speed than its numerically programmed generic counterpart. This gained efficiency is due mainly to the following factors :

- Unproductive arithmetic operations such as multiplication by zero or one are eliminated. This feature is very powerful when performing mathematical

operations on partially filled matrices and vectors which are frequently encountered in the dynamics of robots.

- Unlike numerical programming, the generated code contains no looping, incrementing, calling for external subroutines, testing for counters . . . etc. These are some “intangible” advantages of the symbolically generated code which makes its gained efficiency over a numerical code much greater than what might be indicated by examining only the savings obtained in the number of required arithmetic operations (multiplications or additions).
- By implementing symbolic generation it is possible to take advantage of particular geometric and mass parameters for each robot (e.g. links with zero length, parallel joint axes, sparse center of gravity vectors...etc.). This “customization” process reduces the computational requirements of the generated code by avoiding unnecessary operations and exposing some trigonometric identities.
- The generated code can be further optimized by performing some post-processing on it to remove most of the unproductive and redundant computations.

In symbolic generation all variables (this includes all scalars and vector and matrix elements) are represented and stored in memory as alphanumeric strings, as opposed to real or integer numbers. Mathematical operations are performed by string manipulation of these variables rather than floating point arithmetic as in numerical programming. Two basic procedures (subroutines) were written to perform the addition and the multiplication of string symbols. The procedure that performs the multiplication of two strings consists mainly of the concatenation of the strings

Table 3.2: Special cases implemented in the multiplication procedure

<i>Variable</i> ₁	<i>Variable</i> ₂	<i>Result</i>
<i>STR1</i>	0	0
<i>STR1</i>	1	<i>STR1</i>
<i>STR1</i>	-1	- <i>STR1</i>
- <i>STR1</i>	-1	<i>STR1</i>
<i>STR1</i>	<i>STR2</i>	<i>STR1</i> * <i>STR2</i>
- <i>STR1</i>	<i>STR2</i>	- <i>STR1</i> * <i>STR2</i>
<i>STR1</i>	- <i>STR2</i>	- <i>STR1</i> * <i>STR2</i>
- <i>STR1</i>	- <i>STR2</i>	<i>STR1</i> * <i>STR2</i>

with the insertion of the multiplication sign “*” in between. Table 3.2 lists some of the special cases this procedure implements. It should be noted here that variables *STR1* and *STR2* could be single strings or multiple strings related together by one or more multiplications. Also the two variables could be in any order.

A similar procedure was developed to perform the addition of two strings which is mainly the concatenation of the strings with the insertion of the addition sign “+” or subtraction sign “-” in between. This procedure implements the special cases outlined in table 3.3. Again, *STR1* and *STR2* could be simple² strings or compound strings³.

These two basic procedures were the building blocks for a library of procedures that perform vector and matrix algebra. This library includes procedures for the following operations :

²strings representing single variables

³strings each representing more than one variable related by arithmetic operations

Table 3.3: Special cases implemented in the addition procedure

<i>Variable₁</i>	<i>Variable₂</i>	<i>Result</i>
<i>STR1</i>	0	<i>STR1</i>
<i>STR1</i>	$-STR1$	0
<i>STR1</i>	$-STR2$	$STR1 - STR2$
<i>STR1</i>	<i>STR2</i>	$STR1 + STR2$

- The dot and cross product of vectors.
- The addition and subtraction of vectors.
- Scalar times a vector.
- Matrix multiplication, addition, and subtraction.
- Scalar times a matrix.

To alleviate the problem of increasing string length and also to reduce the amount of redundant computations, an elementary process of string simplification was implemented. This simplification scheme was developed by Kermack [19] as a solution for the rapidly increasing string length in the symbolic generator *DYNAM*. When a string is formed which contains one or more additions or more than a specified number of multiplications⁴, a new dummy variable is generated which takes the form of "Z####" where #### is an automatically incremented counter. This dummy variable replaces the string in memory and an assignment statement defining this variable is appended to the Fortran code. For example pre-multiplying a vector *B* by a matrix *A*, where

⁴six multiplications in the current implementation.

$$A = \begin{bmatrix} C1 & 0 & -S1 \\ S1 & 0 & C1 \\ 0 & -1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} BX \\ BY \\ Z0003 \end{bmatrix}$$

results in a vector C which is stored in the memory as

$$C = \begin{bmatrix} Z0005 \\ Z0006 \\ -BY \end{bmatrix}$$

and the following Fortran statements are appended to the generated code :

$$Z0005 = C1 * BX - S1 * Z0003$$

$$Z0006 = S1 * BX + C1 * Z0003$$

This simplification procedure is similar to the function "LAYER" which maintains a "symbol list" in the algebraic manipulation language *MACSYMA*. The major difference is that in *MACSYMA* the symbolic list is modified during the generation phase while in the current system no modifications are performed in this phase. Furthermore, the *MACSYMA* function "CLEAR" will automatically perform a procedure called "replacement layering" which is the replacement of duplicated strings in the symbol list with a previously defined symbol. In the present system, this type of simplification is performed in a post-processor called *CLEAR* which is described in the next section.

The required user input to the generators is limited to a file containing robot geometry parameters (e.g. number and type of links, lengths, offsets, twist angles ...etc.) and mass and inertia parameters (e.g. masses, center of mass vectors and moment of inertia tensors). For the flexible link generators, the user must

also designate the flexible links and their stiffness, the directions of flexibility, the number of assumed modes to approximate deflections. The generators are interactive programs. The user is also given the options to read the robot parameters from an existing file, to input them manually and then save them, or to alter an existing parameter file. The user is given the option of including a payload vector and the choice of the coordinate frame in which this vector is described.

The output of the generators is a compiler-ready Fortran subroutine. This Fortran subroutine consists of three main parts as shown in figure 3.1. The first part is a declaration section where all the strings that identify the mass and geometric constants are assigned to their numeric values. Some of these values, such as masses and lengths, are obtained from the input parameter file. Other parameters such as flexibility constants for flexible links are calculated by the generating program automatically. Note that these constants are assigned names that start with a lower case letter. This convention was adopted in order to make it easy for the post-processor *CLEAR* to identify them as constants. The second part consists of a series of assignment statements defining the 'Z####' terms described earlier. These terms constitute the bulk of the Fortran subroutine. It should be mentioned that these terms do not have any dynamic significance. They are only a tool to solve memory problems and reduce computational requirements. For the 6 degrees of freedom Stanford arm, about 197 of these 'Z####' are generated for the inverse rigid link dynamics computation using the generator *NEDYN*. Using the generator *FLEX* to generate the inverse dynamics code for a 7 degrees of freedom flexible link manipulator ⁵, about 324 of these 'Z####' are produced. The third part of the Fortran subroutine contains a series of statements which assign the resultant forces or moments to their corresponding output vectors.

⁵one rigid link and two flexible links each having two directions of flexibility

```

SUBROUTINE NEDYN(TH,THD,THDD,LD,RES)
IMPLICIT REAL * 4(A - Z)
REAL TH(6),THD(6),THDD(6),LD(6),RES(6)

  g      = 9.81
  a1     = 0.00000
  C1     = COS(TH(1))
  S1     = SIN(TH(1))
  QD1    = THD(1)
  QF1    = THDD(1)
  m1     = 9.2900
  sz1    = -0.110500
  j11    = 0.159722
  j31    = 0.068155
  :      :      :
  Z0001  = -C2 * QD1 * QD2 - S2 * QF1
  Z0002  = -S2 * QD1 * QD2 + C2 * QF1
  Z0003  = -C2 * QD1 * C2 * QD1 - QD2 * QD2
  Z0004  = -C2 * QD1 * C2 * QD1 - S2 * QD1 * S2 * QD1
  Z0005  = -QD2 * QD2 - S2 * QD1 * S2 * QD1
  Z0006  = -Z0002 - QD2 * S2 * QD1
  :      :      :
  Z0193  = S2 * Z0184 - C2 * Z0186
  Z0194  = -spyl * m1 * QD1 * QD1 * sz1 + spz1 * m1 * g
  Z0195  = -j21 * QF1 - spz1 * m1 * QF1 * sz1
  Z0196  = Z0194 + Z0192
  Z0197  = Z0195 + Z0193
  :      :      :
  RES(1)  = -Z0197
  RES(2)  = Z0185
  RES(3)  = Z0166
  :      :      :

RETURN
END

```

Figure 3.1: sample segments of the generated Fortran code

3.2 The *CLEAR* Post-Processor

After the generation procedure, the generated code is further simplified and optimized in a post-processing program called *CLEAR*. This program was originally developed by Toogood [42, 43] and then modified by the present author where some structures were revised and new functions were added. A text file containing the generated Fortran code is read by *CLEAR* and a series of simplification procedures are performed automatically on this file. These procedures include removing of unused terms, trigonometric simplification, renaming of duplicate multiplication terms, factoring and pre-computation of constant terms (usually involving inertia and mass properties). On an 80386-based microcomputer, the processing time required to perform the simplification depends on the length of the generated code and varies from a few seconds for robots with simple geometries up to a few minutes for complex robots with a large number of degrees of freedom. Using dynamic memory allocation, *CLEAR* can accommodate 3200 text lines. For rigid manipulators with six degrees of freedom, fewer than 400 lines are required while about 450 lines are needed for a 7 degrees of freedom flexible link manipulator. In the following sections, some more detailed description of the functions of *CLEAR* will be presented.

3.2.1 Removing unused terms

The removal of unused terms is one of the original and most powerful functions in *CLEAR*. Some unproductive assignment statements are present in the original generated Fortran code which do not contribute to the calculation of the final results. For example, when a certain variable is multiplied by a zero it will disappear from subsequent Fortran assignment statements and all the assignment statements that are devoted solely to calculate this variable can be deleted without altering the final

results. The removal of unused terms in *CLEAR* works as follows: starting at the end of the file and working to the top, the $Z####$ term of left hand side of each assignment statement is extracted and the following statements in the text file are scanned for the presence of this term. If the term is located then this statement is saved; otherwise, the statement is erased from the file. For example in figure 3.2 the term $Z0195$ is present in one of the following assignment statements and therefore it was retained as shown in figure 3.3. On the other hand, the terms $Z0196$ and $Z0198$ were deleted from the file because they were not used in the subsequent statements.

By utilizing this procedure, usually about 10% to 30% of the statements and computations in the file are eliminated.

3.2.2 Pre-computation of constant terms

The pre-computation of constant terms is one of the new procedures added to *CLEAR*. In the generated codes, mathematical operations involving constant terms are sometimes encountered. These constants often represent mass, inertia and geometrical parameters, or combinations of these, that are fixed for the manipulator and do not vary with the positions of the different joints. To ease the process of identification of these simplification opportunities, constant terms are always saved in the lower case alphanumeric string form while the upper case form is utilized for the variable terms. *CLEAR* extracts these terms, performs the mathematical operation (either multiplication or addition), assigns the result to an automatically generated $x###$ term and then substitutes back the $x###$ term in the original statement.

As an example, the assignment statement that defines $Z0017$ in figure 3.4 contains some constant terms ($j32, j22$) which represent moments of inertia. *CLEAR* will factor out the other terms, generate the $x###$ term and substitute it back. By

```

      :
Z0195  =  -j21 * QF1 - spz1 * m1 * QF1 * sz1
Z0196  =  Z0194 + Z0192
Z0197  =  Z0195 + Z0193
Z0198  =  spy1 * m1 * QF1 * sz1 + Z0185
RES(1) =  -Z0197
RES(2) =  Z0185
RES(3) =  Z0166

```

Figure 3.2: Fortran code before removing unused terms

```

      :
Z0195  =  -j21 * QF1 - spz1 * m1 * QF1 * sz1
Z0197  =  Z0195 + Z0193
RES(1) =  -Z0197
RES(2) =  Z0185
RES(3) =  Z0166

```

Figure 3.3: Fortran code after removing unused terms

utilizing this function of *CLEAR* most of the computations that can be executed off-line are performed and hence a reduction in the number of on-line computations is achieved.

$$\begin{aligned}
 \text{a) } Z0017 &= QD2 * j32 * C2 * QD1 - C2 * QD1 * j22 * QD2 \\
 \text{b) } C \quad x001 &= j32 - j22 \\
 x001 &= 0.026343 \\
 Z0017 &= C2 * QD1 * QD2 * x001
 \end{aligned}$$

Figure 3.4: Fortran code a) before and b) after computations of constant terms

3.2.3 Renaming of duplicate multiplication terms

Locating and renaming of duplicate multiplication of the same two terms is another powerful function of *CLEAR* developed by the present author. When two terms (e.g. *C2* and *QD1* in figure 3.5) are multiplied together in more than one location in the code, a new *Y###* term (*Y001* in figure 3.6) is generated and inserted into the appropriate position in the code. The two terms (i.e. *C2*QD1*) are replaced in all other locations with the *Y###* term as shown in figure 3.6. Therefore, instead of performing a certain multiplication operation several times throughout the code, it needs to be done only once. This usually reduces the number of multiplication required by up to 20%.

This procedure works as follows : starting at the top of the text file, each assignment statement is factored to its basic terms. For example the first part of the assignment statement defining *Z0001* in figure 3.1 is factored to produce the terms *C2*, *QD1* and *QD2*. The rest of the file is scanned to search for the occurrence of these terms (terms detected are shown in bold face in figure 3.5). When one of

these terms is found, the location is stored in an array as shown in table 3.4. This location code is a combination of the line number and the position⁶ of the term in the line. Then the arrays containing the location codes of all the terms in the first part of the statement are intersected to determine the common locations of each pair of the terms (i.e. the locations where the two terms are multiplied together). For example, the locations of the terms *C2* and *QD1*, which are found in the first and second rows of table 3.4 respectively, are intersected to give the common locations as shown in the first row of table 3.5. Finally, the pair of terms that are multiplied together most frequently (i.e. *C2* and *QD1* as shown in table 3.5) are replaced by the *Y###* term. The procedure is repeated for the rest of the terms including the new *Y###* term. As a result of the extent of processing due to this algorithm, renaming of duplicate terms requires the most time of all *CLEAR* functions.

Table 3.4: The locations of each term in the Fortran code

Term	Locations														
<i>C2</i>	255	261	265	270	272	285	287	306	308	318	561				
<i>QD1</i>	254	256	259	261	270	272	274	283	285	287	293	306	308	318	...
<i>QD2</i>	254	259	261	270	274	282	287	291	306	321	323	325			

Table 3.5: Common locations for each pair of terms

Term 1	Term 2	Common locations	Count
<i>C2</i>	<i>QD1</i>	261 270 272 285 287 306 308 318	8*
<i>C2</i>	<i>QD2</i>	261 270 287 306	4
<i>QD1</i>	<i>QD2</i>	254 259 261 270 274 287 306	7

After this procedure, the Fortran code will appear as shown in figure 3.6.

⁶By position, it is meant whether the term is found in the first part of the assignment statement before the addition or subtraction sign or in the second part.


```

Z0001 = -C2 * QD1 * QD2 - QF1 * S2
Z0002 = -QD1 * QD2 * S2 + C2 * QF1
Z0004 = -QD1 * QD1
Z0006 = -Z0002 - QD1 * QD2 * S2
Z0011 = Z0001 + C2 * QD1 * QD2
Z0012 = Z0006 * dl2 - S2 * g
Z0013 = Z0011 * dl2 + C2 * g
Z0014 = Z0006 * sy2 + Z0012
Z0016 = Z0011 * sy2 + Z0013
Z0017 = C2 * QD1 * QD2 * r001
Z0018 = C2 * QD1 * QD1 * S2 * r002
Z0019 = QD1 * QD2 * S2 * r003
Z0020 = Z0001 * j12 + Z0017
Z0021 = QF2 * j22 + Z0018
Z0022 = Z0002 * j32 + Z0019
Z0025 = -QD2 * QD2 - QD1 * QD1 * S2 * S2
Z0027 = QF2 - C2 * QD1 * QD1 * S2
Z0029 = -Z0001 + C2 * QD1 * QD2
:

```

Figure 3.5: Fortran code before renaming of duplicate multiplication terms

```

Y'001  =  C2 * QD1
Y'002  =  Y'001 * QD2
Z0001  =  -Y'002 - QF1 * S2
Y'003  =  QD1 * S2
Y'004  =  Y'003 * QD2
Z0002  =  -Y'004 + C2 * QF1
Z0004  =  -QD1 * QD1
Z0006  =  -Z0002 - Y'004
Z0011  =  Z0001 + Y'002
Z0012  =  Z0006 * dl2 - S2 * g
Z0013  =  Z0011 * dl2 + C2 * g
Z0014  =  Z0006 * sy2 + Z0012
Z0016  =  Z0011 * sy2 + Z0013
Z0017  =  Y'002 * x001
Y'005  =  Y'001 * Y'003
Z0018  =  Y'005 * x002
Z0019  =  Y'004 * x003
Z0020  =  Z0001 * j12 + Z0017
Z0021  =  QF2 * j22 + Z0018
Z0022  =  Z0002 * j32 + Z0019
Z0025  =  -QD2 * QD2 - Y'003 * QD1 * S2
Z0027  =  QF2 - Y'005
Z0029  =  -Z0001 + Y'002
      ⋮

```

Figure 3.6: Fortran code after renaming of duplicate multiplication terms

3.2.4 Trigonometric simplifications

Opportunities for trigonometric simplifications often arise in the generated Fortran code especially if the manipulator has some parallel revolute joint axes. *CLEAR* searches for a number of trigonometric identities such as

$$\begin{aligned}\pm \sin(\theta_1) * \sin(\theta_1) \pm \cos(\theta_1) * \cos(\theta_1) &= \pm 1 \\ \cos(\theta_1) * \cos(\theta_2) \mp \sin(\theta_1) * \sin(\theta_2) &= \cos(\theta_1 \pm \theta_2) \\ \pm \cos(\theta_1) * \sin(\theta_2) + \cos(\theta_2) * \sin(\theta_1) &= \sin(\theta_1 \pm \theta_2)\end{aligned}$$

and performs the simplification. Examples of such simplifications are shown in figure 3.7. The program detects the simplification opportunities automatically.

$$\begin{aligned}a) \quad Z0004 &= -C2 * QD1 * C2 * QD1 - S2 * QD1 * S2 * QD1 \\ Z0102 &= C4 * Z0099 * S5 + Z0099 * C5 * S4 \\ b) \quad S45 &= SIN(TH(4) + TH(5)) \\ Z0004 &= -QD1 * QD1 \\ Z0102 &= Z0099 * S45\end{aligned}$$

Figure 3.7: Fortran code a) before and b) after trigonometric simplifications

3.2.5 Factoring

Factoring is one of the original functions of *CLEAR*. By using this procedure, the program searches for opportunities to extract some common factors in order to reduce the amount of computations as shown in figure(3.8).

3.2.6 Pre-determined sequence

The previous procedures can be performed in any order although a preset automatic sequence can be initiated. By experimenting with the order of these operations, a certain sequence was found to perform the greatest simplifications. This pre-determined sequence is in the following order:

1. Removing unused terms
2. Trigonometric identities
3. Pre-computation of operations involving constants
4. Renaming of duplicate multiplication terms
5. Factoring

3.2.7 Statistics

CLEAR also provides a tally of arithmetic operations required for each Fortran code. At the end of simplification runs, *CLEAR* provides a report containing a count of the number of operations and assignment statements at the end of each simplification procedure. For example, after processing the inverse dynamics code for the Stanford arm, *CLEAR* will produce a statistical report as shown in figure 3.9. The number of arithmetic operation required by the numerical Newton- Euler algorithm to compute the inverse dynamics for the six DOF Stanford manipulator is about 984 multiplications and 1125 additions which adds to a total of 2109 arithmetic operations. From this report, it can be noted that the symbolically generated source code (S) had a total of 561 arithmetic operation before processing. This means that the computational requirements of the symbolically generated code for this

particular example is only about 27 % of computational requirements of the original numeric algorithm even before performing any simplification by the post-processor *CLEAR*. After the removal of the unused terms, the total number of operations was reduced to 433 with a saving of about 23 % . Trigonometric (T) simplification cut 6 operations, pre-computation of operation involving constants (C) cut 80 and renaming of duplicate terms (D) cut 38 arithmetic operations. For this particular case, factoring (F) did not reduce the computational requirements. The total processing by *CLEAR* reduced the total number of arithmetic operations from 571 operations to 309 which means a reduction of 252 operations or about 45 %. A more detailed analysis of the computational efficiencies of the symbolically generated dynamics is found in chapter four.

In this chapter the basics of symbolic generation and the way it is employed to generate codes containing the dynamics for manipulators was presented. Also, the simplification procedures that optimize the generated code as they are implemented in the post-processor *CLEAR* were explained. In the following chapter a comparison of the computational efficiency between this system and other systems will be presented together with some test cases for verification.

$$\begin{aligned} a) \quad Z0009 &= -Z0002 * sy2 - Y004 * sy2 \\ b) \quad Z0009 &= sy2 * (-Z0002 - Y004) \end{aligned}$$

Figure 3.8: Fortran code a) before and b) after factoring

Chapter 4

Verifications and Results

This chapter will present test cases which verify the dynamic models and demonstrate the computational efficiencies of the symbolic generators developed as a part of this work. The first section will discuss the generated dynamics for rigid link manipulators. The second section will examine the generated dynamics for the flexible link manipulators.

4.1 Rigid Link Dynamics

This section presents several test cases used to verify the correct implementation of the recursive Newton Euler technique of Luh et al. [25] in the inverse dynamic symbolic generator *NEDYN* and in the closed form and direct dynamics generators which were derived from *NEDYN*. In addition to this, the computational efficiencies of the dynamic codes produced by *NEDYN* and the other generators are compared to recently reported results in the literature for some common industrial manipulators.

4.1.1 Inverse Dynamics

The first stage of verifying the correct implementation of the recursive Newton-Euler algorithm in the symbolic generator *NEDYN* consisted of generating inverse dynamics codes for simple one and two degrees of freedom linkages. These were a single link pendulum and a double link pendulum. The generated dynamics codes were compared to manually derived equations of motion for these cases. The result was complete agreement between the manually derived dynamics and the symbolically generated ones.

In the second stage, inverse dynamics codes symbolically generated by *NEDYN* were verified against dynamics codes generated by *DYNAM* which was developed by Toogood [42] and Kermack [19]. *DYNAM* is based on Hollerbach's 4×4 recursive Lagrangian formulation. *DYNAM* underwent rigorous verification firstly by cross checking it against a numerically programmed version of the recursive Lagrangian algorithm and secondly by comparing its generated dynamics codes with manually derived equations of motion for a number of test cases as documented in [43]. Comparison of the results produced by the dynamic codes generated by *NEDYN* and the codes generated by *DYNAM* showed complete agreement. This conforms with the fact that all of the different dynamic formulations are actually equivalent [41].

As an example of the second stage verification, inverse dynamic codes for the six degrees of freedom Stanford manipulator (see figure 4.1) were generated using both *DYNAM* and *NEDYN*. A motion trajectory was created and fed into the inverse dynamics codes in order to calculate the required joint torques that are needed to sustain this motion trajectory. Figure 4.2 shows a typical position, velocity and acceleration profile supplied to each joint (in this case joint 2) to calculate the joint

torque. The other 5 joints were given similar kinematic states but with different ranges of joint movement.

The joint torques corresponding to these kinematic states computed using the dynamic codes produced by *DYNAM* and *NEDYN* are shown in Figures 4.3 and 4.4 respectively.

It can be seen from these figures that the computed torques using the two symbolic generators are the same. The calculated torques by the two methods agreed to the fifth decimal place. The very small discrepancy is due to round off errors which results from the different representation of the inertia term in Newton- Euler and Lagrangian dynamics. For example, in the Newton-Euler algorithm the input moments of inertia parameters are calculated about the center of mass of the link while in the Lagrangian algorithm the input moments of inertia parameters are computed about the link's coordinate system.

4.1.2 Closed form dynamics

As mentioned earlier in chapter two, in order to calculate the closed form dynamics, each of the dynamics terms such as the inertia term and the gravitational term is calculated separately and independently. The verification of these dynamic terms was performed by comparing the sum of the individual dynamic terms for a given manipulator at a given kinematic state to the values of torques calculated by the recursive inverse dynamics for the same manipulator and kinematic state. The sum of these terms should equal the torque calculated by the recursive inverse dynamics.

As an example, the closed form dynamics of the Stanford manipulator were generated and evaluated for a kinematic state similar to one presented in figure 4.2. The computed dynamic terms as well as their sum for joint 2 are plotted in figure 4.5.

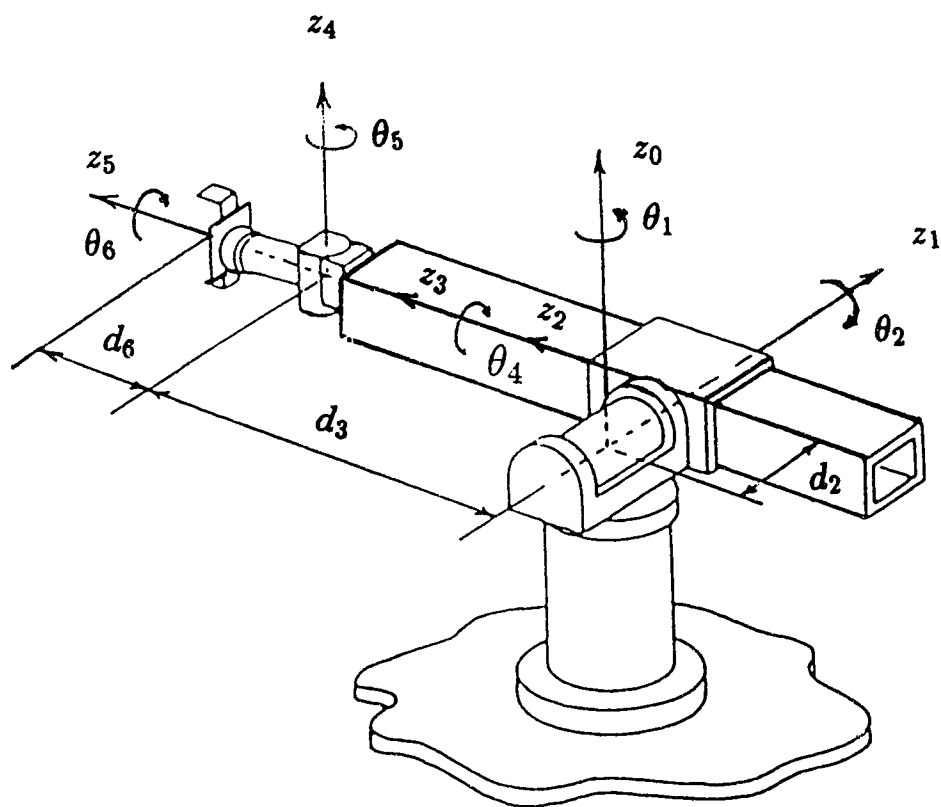


Figure 4.1: Six DOF Stanford manipulator

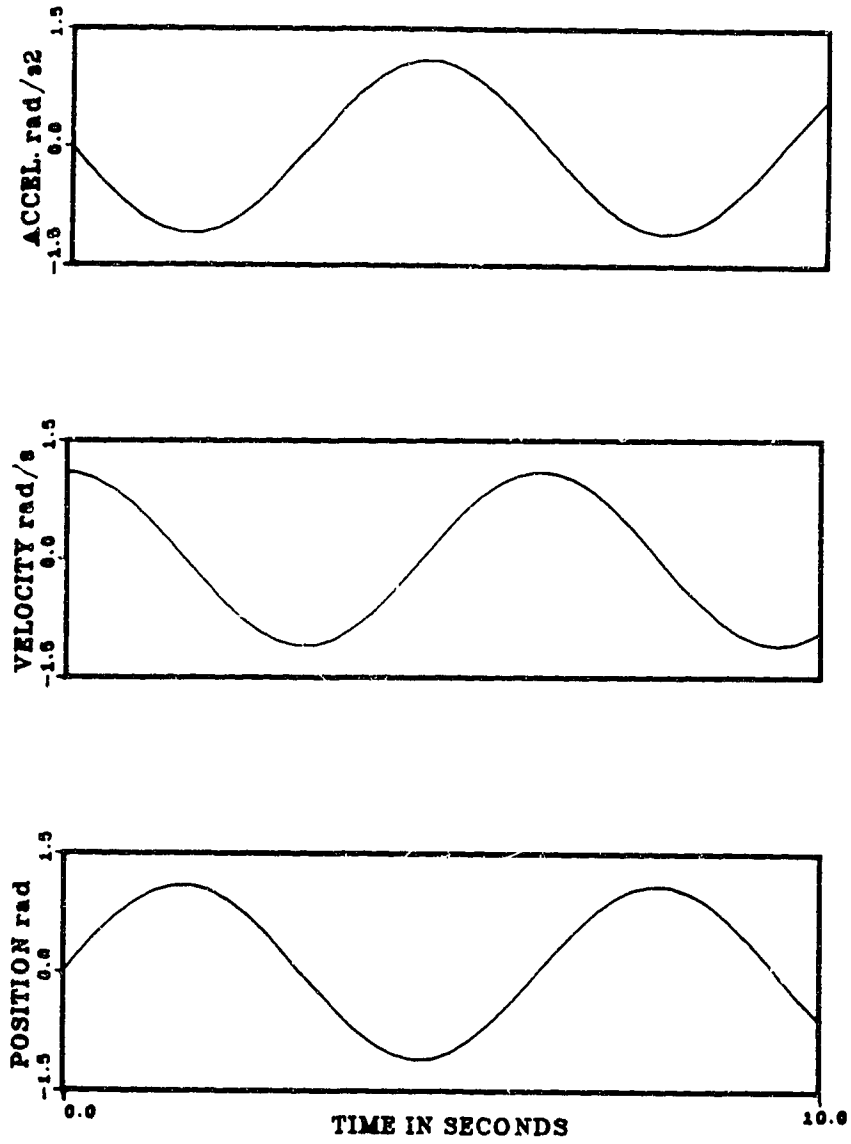


Figure 4.2: Kinematic State of Joint 2

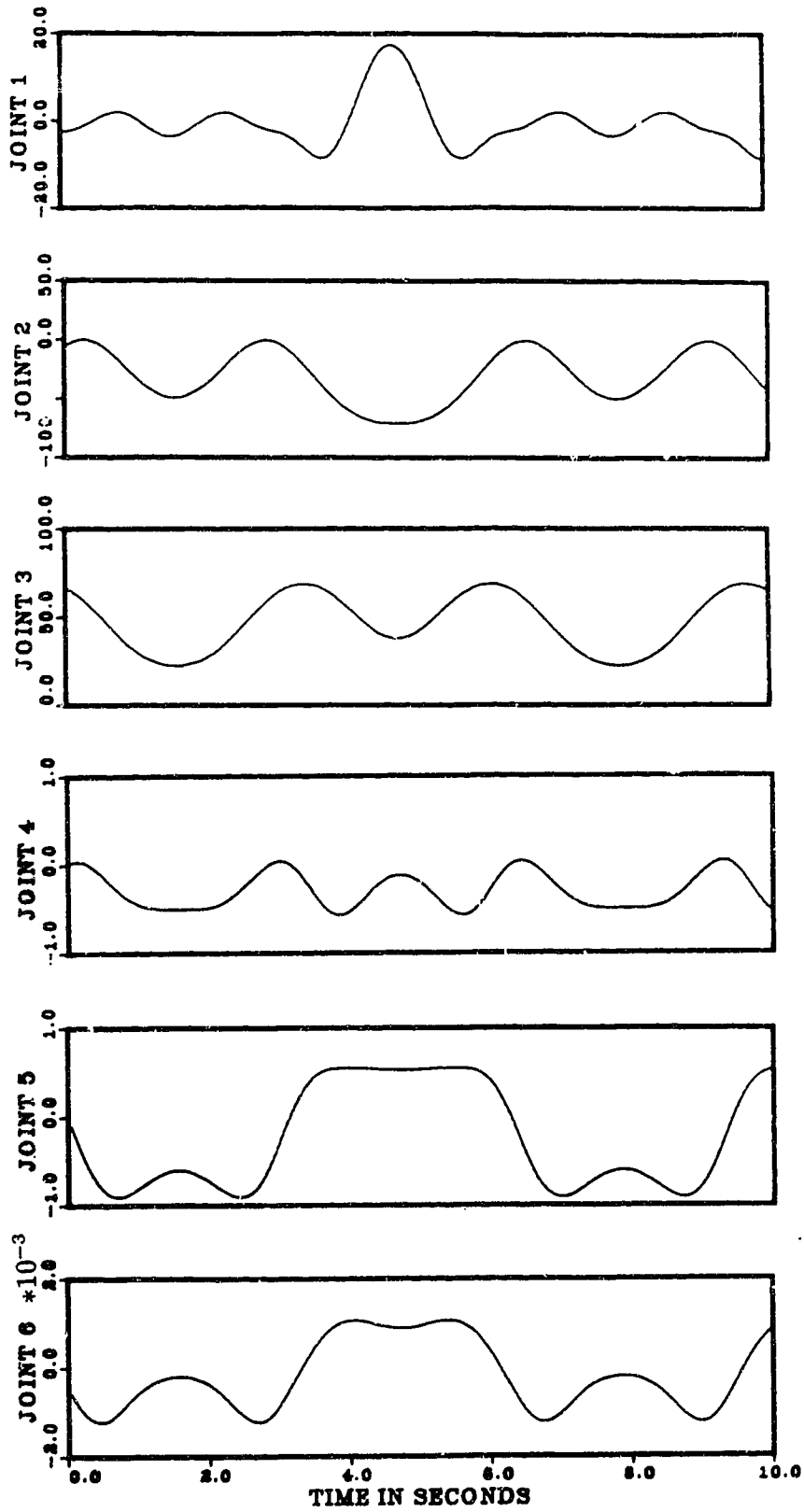


Figure 4.3: Computed Joint Torques Using Dynamic Code Generated by DYNAM

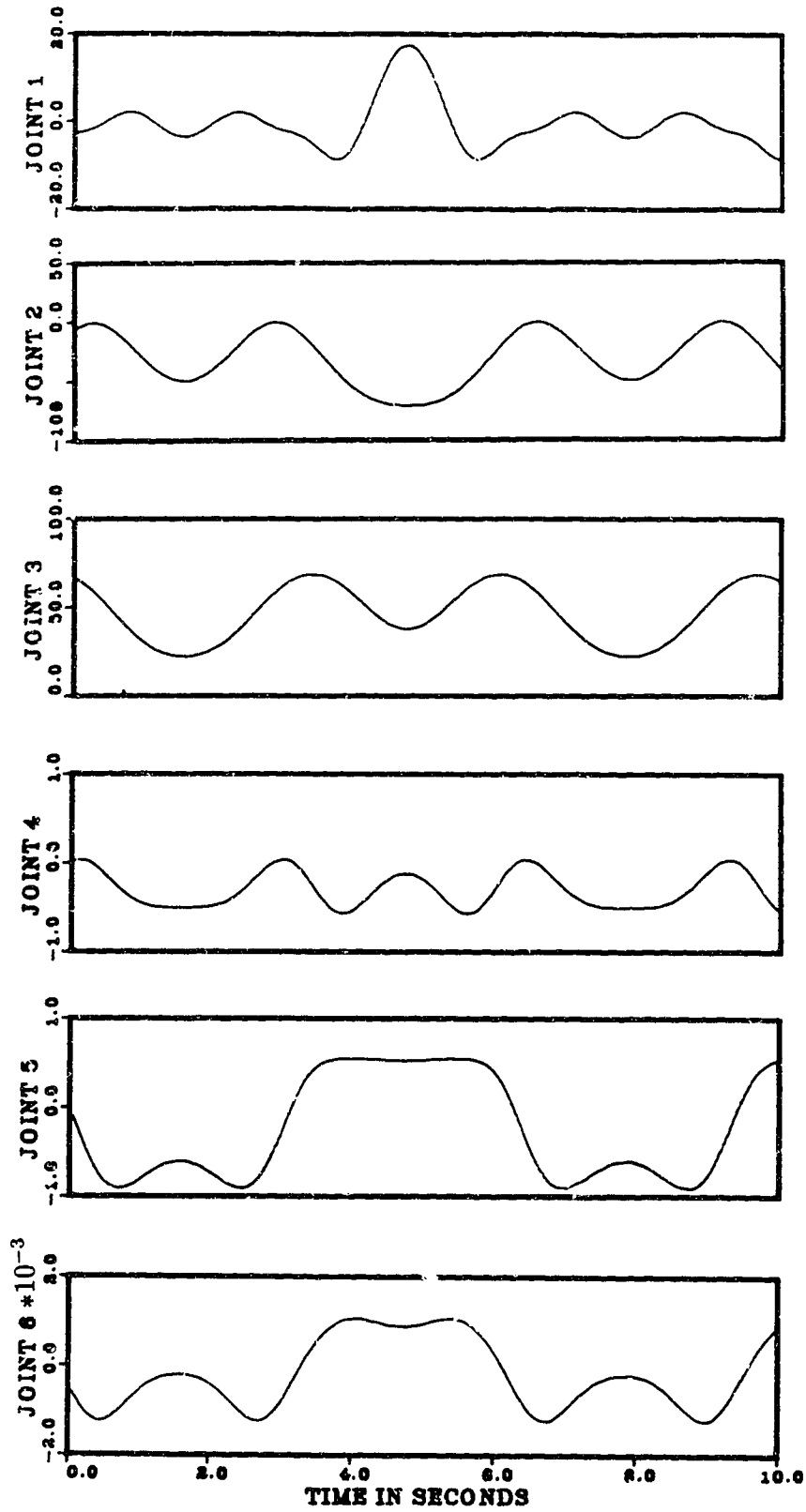


Figure 4.4: Computed Joint Torques Using Dynamic Code Generated by NEDYN

The torques for joint 2 calculated for the same kinematic state using the recursive inverse dynamic generator *NEDYN* are shown in figure 4.6.

It is evident from these plots that the sum of the dynamic terms is identical to the torque calculated from the recursive inverse dynamic algorithm. This agreement suggests the correctness of the calculation of the individual dynamic terms.

It is noticed that the gravity term contribution to total torque at the second joint¹ is very high which is expected as link 2 is very massive. It is also observed that the gravity term contribution to the torque is zero when all the joint angle positions are zeros (i.e. the manipulator is stretching upwards.) and hence the weight of the last five links at this position won't exert any moment on joint 2. The gravity term approaches its maximum value when link 2 is closest to its horizontal position. It is also noticed that the inertia term for joint 2 is maximum when the acceleration of joint 2 is maximum. In calculating the external force term, a constant payload vector consisting of a force of $[5, 5, 5]^T$ N and a moment of $[5, 5, 5]^T$ Nm was applied to the end effector. It is observed that the torque required to offset the external payload is much smaller than the torque required to compensate for the gravitational forces. This is an unfortunate cost that must be paid when designing manipulators to have massive and rigid structures.

4.1.3 Computational Requirements of Inverse Dynamics

As mentioned earlier, one of the biggest advantages of using symbolic generation techniques is to produce dynamics code with the minimal number of arithmetic operations. The usefulness of a particular generator is therefore measured by the computational efficiency of the dynamics code as well as the ease of producing this

¹link 2 is the shoulder for the Stanford manipulator.

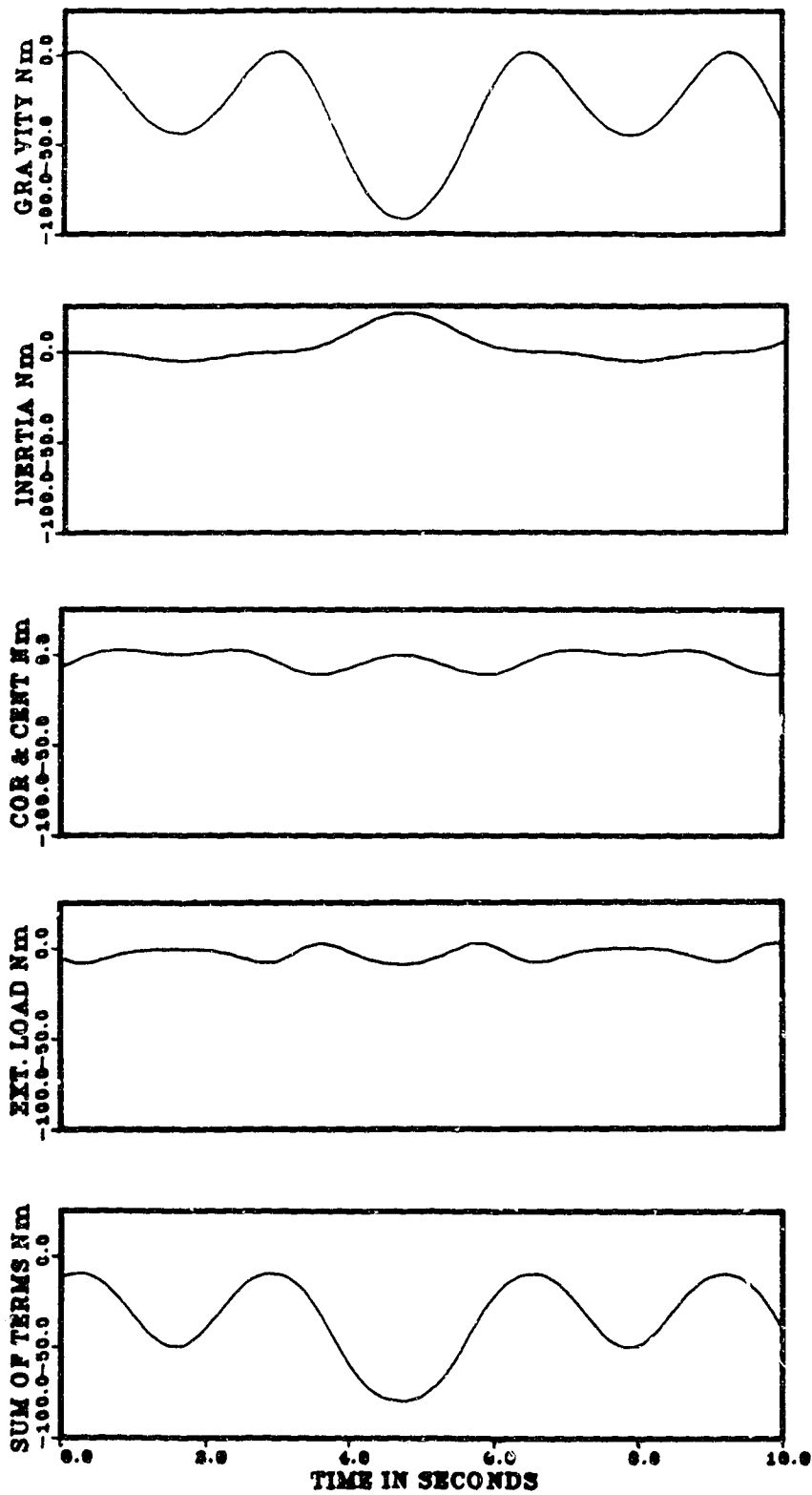


Figure 4.5: Dynamic Terms Contributions to the Total Torque for Joint 2 of the Stanford Manipulator

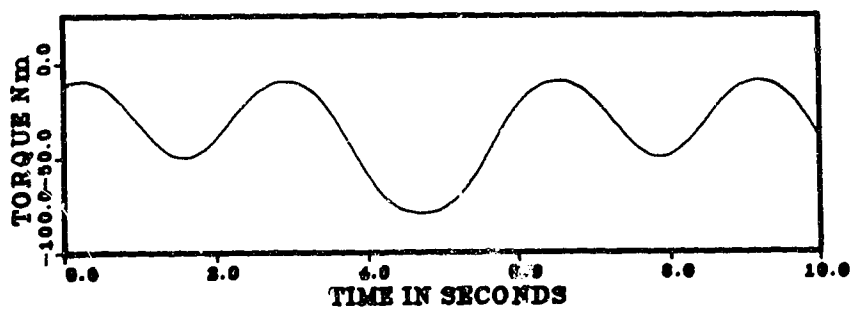


Figure 4.6: Torque for Joint 2 computed by NEDYN

code. Researchers have employed some “benchmarks”, which are commonly used manipulators, to compare the relative efficiency of the different symbolic generation schemes. To illustrate the computational efficiency of the symbolic generator *NEDYN* and the generators for the direct dynamics and closed form dynamics, a comparison between its efficiency and the efficiency of the highly successful symbolic generator *ARM* [27, 28, 29, 30, 31] will be presented. Four benchmark manipulators are utilized to compare the computational efficiency. The manipulators used are the six degrees of freedom Stanford manipulator, the three degrees of freedom Stanford positioning system, the six degrees of freedom Puma 560 manipulator and the three degree of freedom Puma 560 positioning system. Table 4.1 illustrates the computational requirements of the code produced by the *ARM* generator. It is noted from this table that the closed form formulation in *ARM* is the most efficient for the 3 DOF freedom manipulators while the recursive formulation shows the better efficiency for the 6 DOF manipulators. Another note from this table is that the computational requirements vary considerably for different geometries even if the number of degrees of freedom is the same.

Tables 4.2 and 4.3 present the computational requirements of the recursive and closed form inverse dynamics for the 6 DOF and 3 DOF manipulators respectively using *NEDYN*. For the 6 DOF manipulators the dynamic code generated by *NEDYN* is more efficient than the code produced by *ARM* for both the recursive form and the closed form. For the 6 DOF Stanford manipulator 309 and 600 arithmetic operations are required to compute the recursive and the closed form dynamics respectively using *NEDYN*, whereas it takes 333 and 741 arithmetic operations to perform equivalent operations using the *ARM* generator. Similarly, for the 6 DOF Puma 560 manipulator, *NEDYN* requires 358 and 733 arithmetic operations to perform the recursive and closed form inverse dynamics respectively compared to 398 and 813

Table 4.1: Inverse Dynamics Computational requirements for Neuman and Murray's ARM generator [29]

Manipulator	Algorithm	Mult.	Add.	Total
Six-DOF Stanford	closed form	432	309	741
	recursive	185	148	333
Three-DOF Stanford	closed form	40	24	64
	recursive	57	47	104
Six-DOF Puma 560	closed form	461	352	813
	recursive	224	174	398
Three-DOF Puma 560	closed form	55	42	97
	recursive	99	75	174

operations required by *ARM*. For the three degrees of freedom positioning systems *NEDYN* recursive formulations are still more computationally efficient than *ARM*. However, *ARM* produced more computationally efficient closed form inverse dynamics for the three degrees of freedom positioning systems.

It is worth noting also from tables 4.2 and 4.3 that the coriolis and the centrifugal term is the most expensive to calculate in the closed form dynamics yet its contribution to the total joint torque is not as significant as the contribution of the gravity term for example. Therefore, depending on the operating speed of the manipulator, the coriolis and centrifugal term should be the first to be ignored if the computational facilities of the system cannot accommodate the calculation of all of the dynamic terms at a sufficiently fast rate.

Table 4.2: Inverse Dynamics Computational Requirements for Six DOF Manipulators Using NEDYN

Manipulator	Generator	Dynamics Generated	Payload Vector	Mult.	Add.	Total
Stanford	NEDYN	inverse recursive	no	174	135	309
	NEDYN	inverse recursive	yes	174	141	315
	GTERM	gravity term		56	35	91
	HTERM	inertia term		110	77	187
	CTERM	cent. & coriolis		165	113	278
	KTERM	external force		28	16	44
	Total closed form				359	241
Puma 560	NEDYN	inverse recursive	no	204	154	358
	NEDYN	inverse recursive	yes	206	162	368
	GTERM	gravity term		69	43	112
	HTERM	inertia term		133	90	223
	CTERM	cent. & coriolis		193	134	327
	KTERM	external force		44	27	71
	Total closed form				439	294

Table 4.3: Inverse Dynamics Computational Requirements for Three DOF Positioning Systems Using NEDYN

Manipulator	Generator	Dynamics Generated	Payload Vector	Mult.	Add.	Total
Stanford	NEDYN	inverse recursive	no	49	37	86
	NEDYN	inverse recursive	yes	52	44	96
	GTERM	gravity term		7	2	9
	HTERM	inertia term		27	17	44
	CTERM	cent. & coriolis		41	19	60
	KTERM	external force		6	5	11
	Total closed form				81	43
Puma 560	NEDYN	inverse recursive	no	204	154	358
	NEDYN	inverse recursive	yes	206	162	368
	GTERM	gravity term		16	8	24
	HTERM	inertia term		46	28	74
	CTERM	cent. & coriolis		66	39	105
	KTERM	external force		14	9	23
	Total closed form				142	84

4.1.4 Direct Dynamics

The direct dynamics produced by the symbolic generators *HMAT*, *BVECT* and *HBMAT* were verified by establishing closure between the inverse and the direct dynamics. To prove closure, the inverse dynamics code for a certain manipulator was used to calculate the joint torques corresponding to a given kinematic state (joint positions, velocities and accelerations). The computed joint torques together with joint positions and velocities were then utilized in the direct dynamic code (see equation 2.9) to solve for joint accelerations. The resulting joint accelerations should equal the joint accelerations that were initially employed to calculate the joint torques. A number of test cases were performed including one case for the 6 DOF Stanford manipulator. All of the test cases showed complete closure.

Furthermore, simulations were performed for small cases utilizing the symbolically generated dynamics codes and were compared with simulations that employed manually derived equations of motion. The result was complete agreement between the simulations utilizing the symbolically generated dynamics and simulations using manually derived dynamics codes.

Also, inertia matrices obtain by Walker and Orin's method 2 and method 3 (see section 2.2) were cross checked and found to provide identical results.

4.1.5 Computational Requirements of Direct Dynamics

This section will present results demonstrating the computational efficiency of the code produced by *NEDYN* and its derivatives. The computational efficiency of the direct dynamics code produced by *NEDYN* will also be compared with the dynamic code produced by *ARM*.

The direct dynamics code can be generated using three different methods. The first method is to produce the inertia matrix utilizing Walker and Orin's method 2 (see section 2.2) using the symbolic generator *HMAT* and the bias vector using *BVECT* separately. The second method is to generate the inertia matrix and the bias vector simultaneously in order to combine some of the common computations of the two routines. This is done in the symbolic generator *HBMAT*. Moreover, *HBMAT* is capable of producing the inertia matrix using either Walker and Orin method 2 or method 3. Tables 4.4 and 4.5 show the arithmetic requirements of the dynamics computation using the various generators for the 3 DOF and 6 DOF manipulators, respectively. It should be noted that the direct dynamics codes are in a recursive form.

From these tables it can be seen that combining the generation of the bias vector and the inertia matrix in one routine results in a slightly more efficient dynamic code than by separately generating the bias vector and the inertia matrix. Although the gained efficiency is small in terms of the number of saved arithmetic operations, there is a benefit in reducing the number of trigonometric function calls² which are very computationally extensive. The merits of using Walker and Orin's method 3 instead of method 2 are mixed. For the 3 DOF manipulators, method 2 gives better results for all the test cases. For the 6 DOF manipulators the most efficient method is dependent on the geometry of the manipulator. For example, the complete direct dynamics for the 6 DOF Stanford arm requires a total of 590 arithmetic operations if method 2 is used and 601 arithmetic operations when using method 3. On the other hand the calculation of the direct dynamics for the 6 DOF Puma 560 manipulator requires 768 arithmetic operation if calculated by method 2 and only 680 operations if method 3 utilized. This is true in spite of the indication that the Stanford arm,

²10 instead of 20 for the 6 DOF Stanford manipulator

Table 4.4: Direct Dynamics Computational Requirements for Three DOF Positioning Systems Using NEDYN

Manipulator	Generator	Dynamics Generated	Mult.	Add.	Total
Stanford	HMAT	Inertia matrix (method 2)	18	13	31
	BVECT	Bias vector	45	23	68
	Total	HMAT & BVECT	63	36	99
	HBMAT	Inertia & Bias (method 2)	62	35	97
	HBMAT	Inertia & Bias (method 3)	85	55	140
Puma 560	HMAT	Inertia matrix (method 2)	41	27	68
	BVECT	Bias vector	68	41	109
	Total	HMAT & BVECT	109	68	177
	HBMAT	Inertia & Bias (method 2)	107	68	175
	HBMAT	Inertia & Bias (method 3)	132	92	224

Table 4.5: Direct Dynamics Computational Requirements for Six DOF Manipulators Using NEDYN

Manipulator	Generator	Dynamics Generated	Mult.	Add.	Total
Stanford	HMAT	Inertia matrix (method 2)	187	122	309
	BVECT	Bias vector	169	117	286
	Total	HMAT & BVECT	356	239	595
	HBMAT	Inertia & Bias (method 2)	352	238	590
	HBMAT	Inertia & Bias (method 3)	345	256	601
Puna 560	HMAT	Inertia matrix (method 2)	258	183	441
	BVECT	Bias vector	195	136	331
	Total	HMAT & BVECT	453	319	772
	HBMAT	Inertia & Bias (method 2)	449	319	768
	HBMAT	Inertia & Bias (method 3)	396	284	680

which includes a prismatic joint and an additional offset (d_2), appears to be a more geometrically complex manipulator. The presence of a prismatic joint may have reduced the computational requirements for the Stanford manipulator.

Table 4.6 shows the computational requirements to calculate the direct dynamics using code generated by *ARM*. For the 3 DOF manipulators, it is clear that closed form direct dynamic codes generated by *ARM* are superior to the recursive forms, including the present results, in terms of computational efficiency. The recursive direct dynamics code generated by *HBMAT* method 2 is slightly more efficient than the recursive dynamics code generated by *ARM*. For 6 DOF manipulators *ARM* recursive codes are more efficient than the closed form. For the 6 DOF Stanford manipulator *ARM* generated recursive direct dynamic code contained only 553 arithmetic operation while it takes the code generated by *HBMAT* 590 arithmetic operation to perform the same calculations. For the 6 DOF Puma 560 manipulator, the recursive dynamic code generated by *HBMAT* contained 680 arithmetic operations which is slightly more efficient than the 686 operations reported for *ARM*.

4.1.6 Jacobian Matrix

The Jacobian matrix for the 6 DOF Puma 560 manipulator was generated using the symbolic generator *JACOB* which is a modified version of *NEDYN*. The generated matrix was verified against a manually derived matrix for the same manipulator by Leahy et al. [23]. Table 4.7 lists the computational requirements to compute the Jacobian matrix using different methods. From this table it is clear that the algorithm implemented to generate the Jacobian matrix in *JACOB* is not the most efficient method to generate this matrix. Nevertheless it offers an interesting method to easily obtain the Jacobian matrix from the inverse dynamics, thereby avoiding the

Table 4.6: Direct Dynamics Computational requirements for Neuman and Murray's ARM generator [29]

Manipulator	Algorithm	Mult.	Add.	Total
Six-DOF Stanford	closed form	402	280	682
	recursive	309	244	553
Three-DOF Stanford	closed form	33	17	50
	recursive	62	48	110
Six-DOF Puma 560	closed form	430	320	750
	recursive	392	294	686
Three-DOF Puma 560	closed form	46	33	79
	recursive	135	98	233

Table 4.7: Manipulator Jacobian Matrix Computational Requirements

Manipulator	Author	sin/cos	Mult.	Add.	Total
Puma 560	Leahy et al. [23]	12	66	30	99
Puma 560	El-Rayyes (<i>JACOBE</i>)	10	92	49	141
General 6 DOF	Orin and Schrader [36]	10	93	24	117
Simplified Puma 560	Paul et al. [35]	11	46	21	67

complexity of manual derivations.

4.2 Flexible Link Dynamics

In this section the process of verifying the correct implementation of King's [22] algorithm for the flexible manipulators in the symbolic generator *FLEX* will be presented. In the absence of experimental data, rigorous verification of this model is very difficult to achieve. Nevertheless, several test cases were studied which implied the correctness of both the model and the symbolic generation. The geometric and inertial parameters of the flexible links used in the first three test cases are found in table 4.8.

4.2.1 Case Studies

In the following test cases simulations were performed to verify the direct dynamic code produced by the symbolic generators *FLEXH* and *FLEXB*. The system of linear equations 2.9 are solved for the generalized accelerations by Cholesky(LU) decomposition [9] and then integrated using the fourth order Runge-Kutta scheme.

Case 1: Single Flexible Link in Bending

This test case was devised to examine the bending response of a single flexible link to an external step function point force exerted at the free end of the link as shown in figure 4.7. The mode shape functions implemented in this model are the eigenfunctions of the Euler-Bernoulli equation for a beam in bending assuming clamped-free boundary condition. The displacement response of this link to a general dynamic loading can be calculated by the superposition of a number of assumed modes. For this case the flexural deflection was approximated using the first four modes. If damping was introduced and the vibrations were let to die away, it

was possible to calculate the static response of a link using the dynamic model. (Certainly not the most efficient way to calculate it!) *FLEXH* and *FLEXB* were used to symbolically generate the inertia matrix and the bias vector for this test case. The response of the link to a step function point load of 10 N, applied at the end of the link, was simulated. The static deflection of the tip of the link as a result of this load calculated from strength of material considerations is 0.01637 m. The simulated response of the first four generalized deflection variables are shown in figures 4.8 and 4.9. In these figures it is noticed that the generalized variables are vibrating about a mean value which corresponds to the mode contribution to the static deflection of the link. If damping is introduced these vibrations will die out and the generalized variables will reach a value corresponding to the static deflection. To obtain the contribution of each mode to the total displacement, the value of each of the generalized deflection variables has to be multiplied by the value of the corresponding shape function. By comparing the static displacement with the expressions for the dynamic response of the different assumed modes, it is possible to calculate the contribution of each of the modes to the total deflection [10] for that particular loading condition. The theoretical values of the ratio of the contribution of the first four modes divided by the static deflection for a clamped free beam with a concentrated external force acting at the end are found in table 4.9. From this table it can be seen that the first mode contributes about 97 % of the total deflection. Table 4.9 also presents the ratios of the simulated deflections divided by the theoretical static deflection for the four assumed modes. From this table it can be seen that the simulated ratios are virtually the same as the theoretical ones considering round off and truncation errors encountered in numerical integration. Furthermore, table 4.10 shows nearly a perfect agreement in the comparison between the simulated frequencies for the first four assumed modes and theoretical natural frequencies. This test case

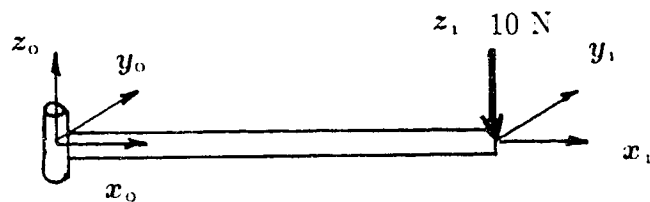


Figure 4.7: Single flexible link in bending

proves that the flexibility parameters for a single link such as stiffness and inertia are modelled correctly in the symbolic generator.

The integration time step used was 0.1 ms. If the integration time step size was increased, the amplitude of the vibrations start to decay experiencing what might be called “numerical damping”. This phenomena is more evident in the higher modes of vibrations which have higher frequencies. This numerical damping is attributed to truncation errors that are associated with numerical integration. For example, mode 4 has a frequency of about 388 Hz which corresponds to a period of 2.6 ms. Using a rough rule of thumb, the integration time step should be less than $\frac{1}{10}$ of the period or less than 0.26 ms. Therefore, by using a time step of 0.1 ms, numerical complications were not evident.

Table 4.8: Link material and geometry parameters used in the test cases

Link parameters	
Length	1.00 m
Radius	0.06 m
Mass	0.5 Kg
Youngs Modulus	2×10^{11} Pa
Modulus of Rigidity	8.3×10^{10} Pa
Xsectional Area	1.1309734×10^{-4} m ²
I_{xx}	2.035752×10^{-9} m ⁴
I_{yy} & I_{zz}	1.017876×10^{-9} m ⁴

Table 4.9: Ratios of the contribution of the first four modes to the static deflection for a link in bending with a point load at the end

mode	theoretical/static	simulated/static
1	0.97069	0.97070
2	0.02472	0.02472
3	0.00315	0.00315
4	0.00082	0.00082

Table 4.10: Comparison between the theoretical and simulated frequencies for a single flexible link in bending.

mode	theoretical frequency Hz	simulated frequency Hz
1	11.29	11.29
2	70.76	70.80
3	198.1	198.3
4	388.3	388.2

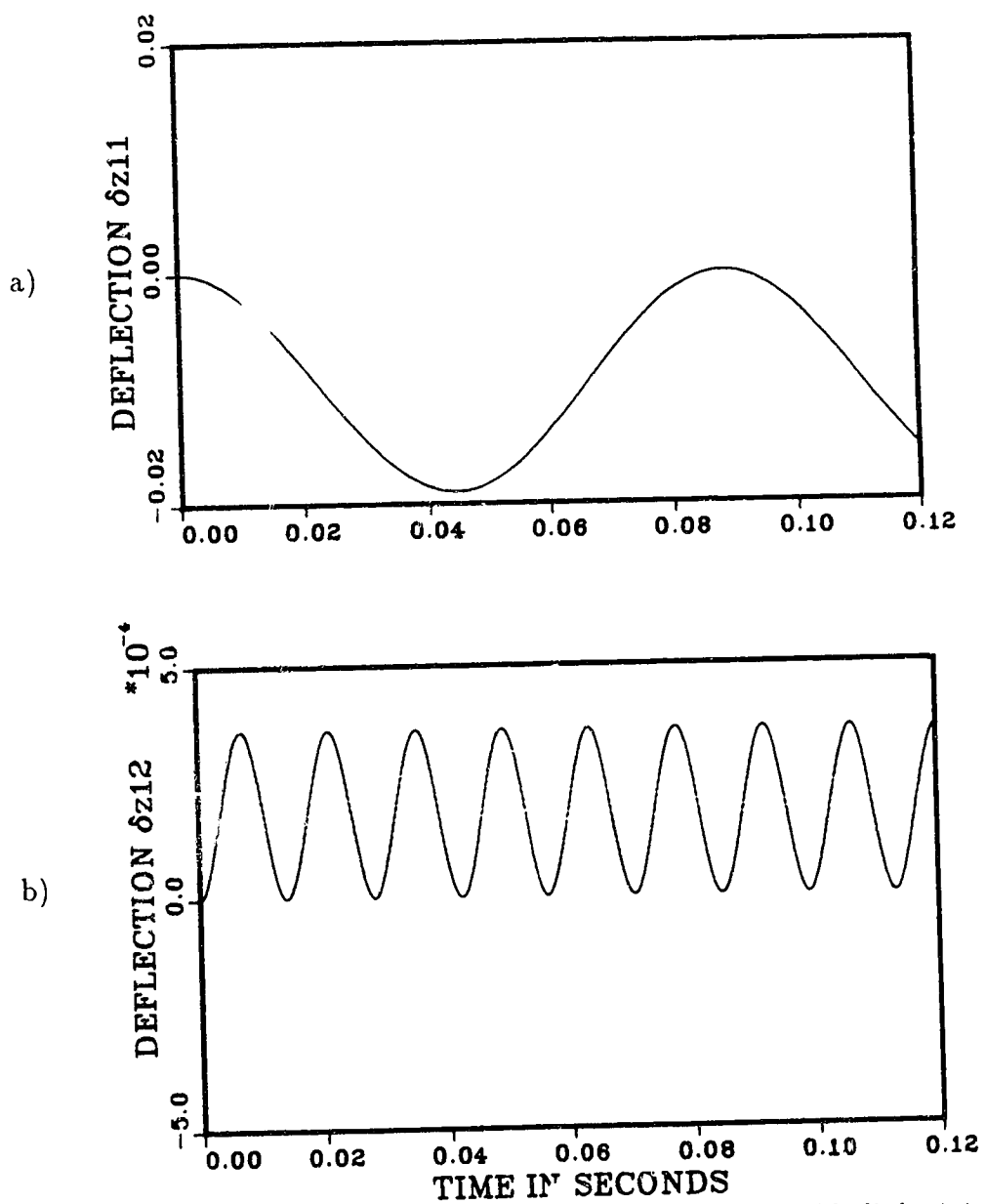


Figure 4.8: Response of bending variables of a single flexible link: a) 1st mode
b) 2nd mode

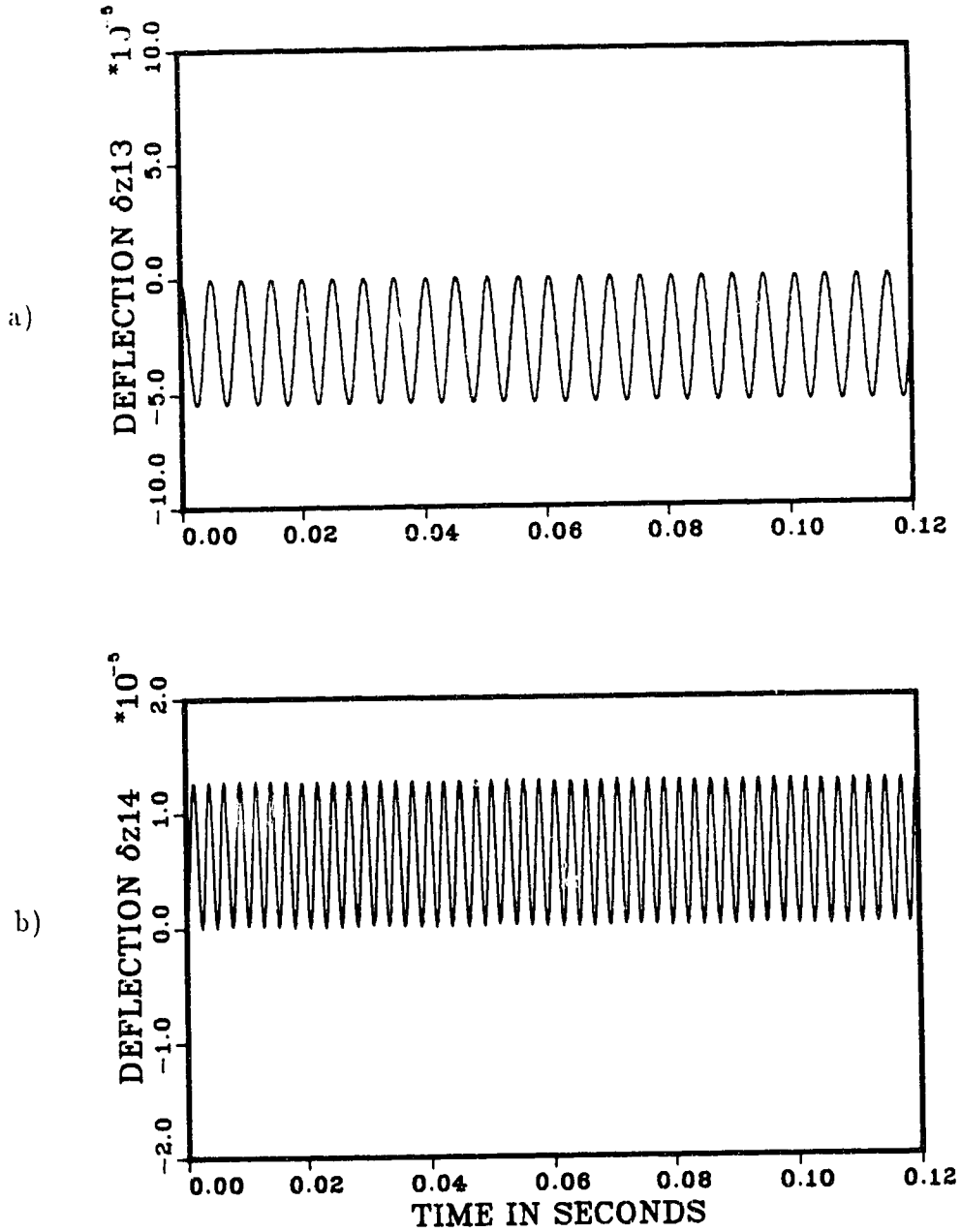


Figure 4.9: Response of bending variables of a single flexible link: a) 3rd mode
b) 4th mode

Case 2: Single Flexible Link in Torsion

Similar to case 1, this test case was devised to verify the response of a single flexible link when it is loaded in torsion as shown in figure 4.10. The mode shape functions utilized to model torsion are the eigenfunctions of equation 2.25 for a beam in torsion assuming clamped-free boundary condition. The torsional deflection for this test case was approximated by the first four assumed modes. This test case was simulated using dynamic codes generated by the symbolic generators. The simulated responses of the generalized variables to a step external torque of 12 Nm and an integration time step of 0.01 ms are shown in figures 4.11 and 4.12. The theoretical values of the ratio of the contribution of the first four modes divided by the static torsional deflection for a clamped free beam with an external torque acting at the end are found in table 4.11. It is noticed that the first mode contributes about 81 % of the total deflection. The ratios of the simulated response of the different modes divided by the theoretical static deflection of 0.0710196 rad are almost identical to the theoretical ratios as shown in table 4.11. Furthermore, the simulated frequencies were very close to the theoretical ones as shown in table 4.12. The period of mode 4 is 0.13 ms. Using the “time step less than $\frac{1}{10}$ of the period” rule of thumb, a time step of 0.01 should be adequate. However, a small error is noted for mode 4 and even for mode 2. This leads to the conclusion that the time step should be less than $\frac{1}{30}$ of the natural frequency as in test case 1 where integration errors were smaller. Finally, this test case gives the confidence that torsion for a single flexible link is modelled correctly.

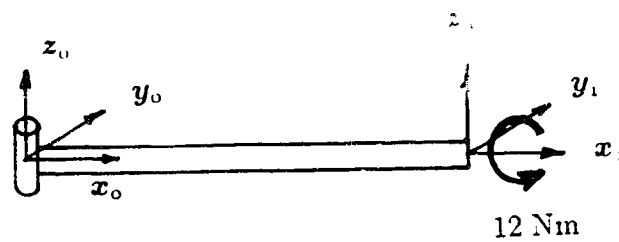


Figure 4.10: Single flexible link in torsion

Table 4.11: Ratios of the contribution of the first four modes to the static deflection for a link in torsion with an external torque at the end

mode	theoretical/static	simulated/static
1	0.8106	0.8108
2	0.0901	0.0901
3	0.0324	0.0324
4	0.0165	0.0165

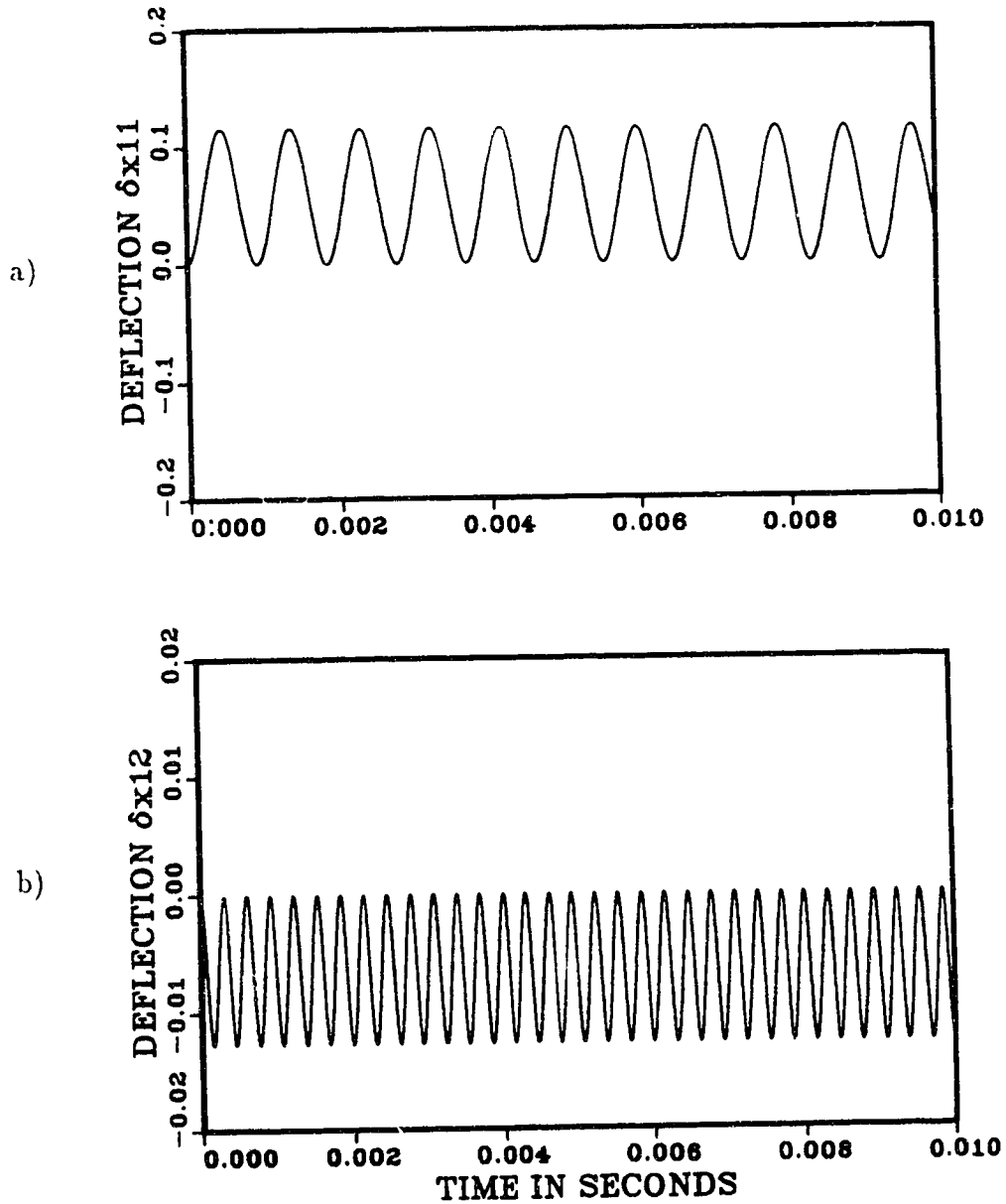


Figure 4.11: Response of torsion variables of a single flexible link: a) 1st mode
b) 2nd mode

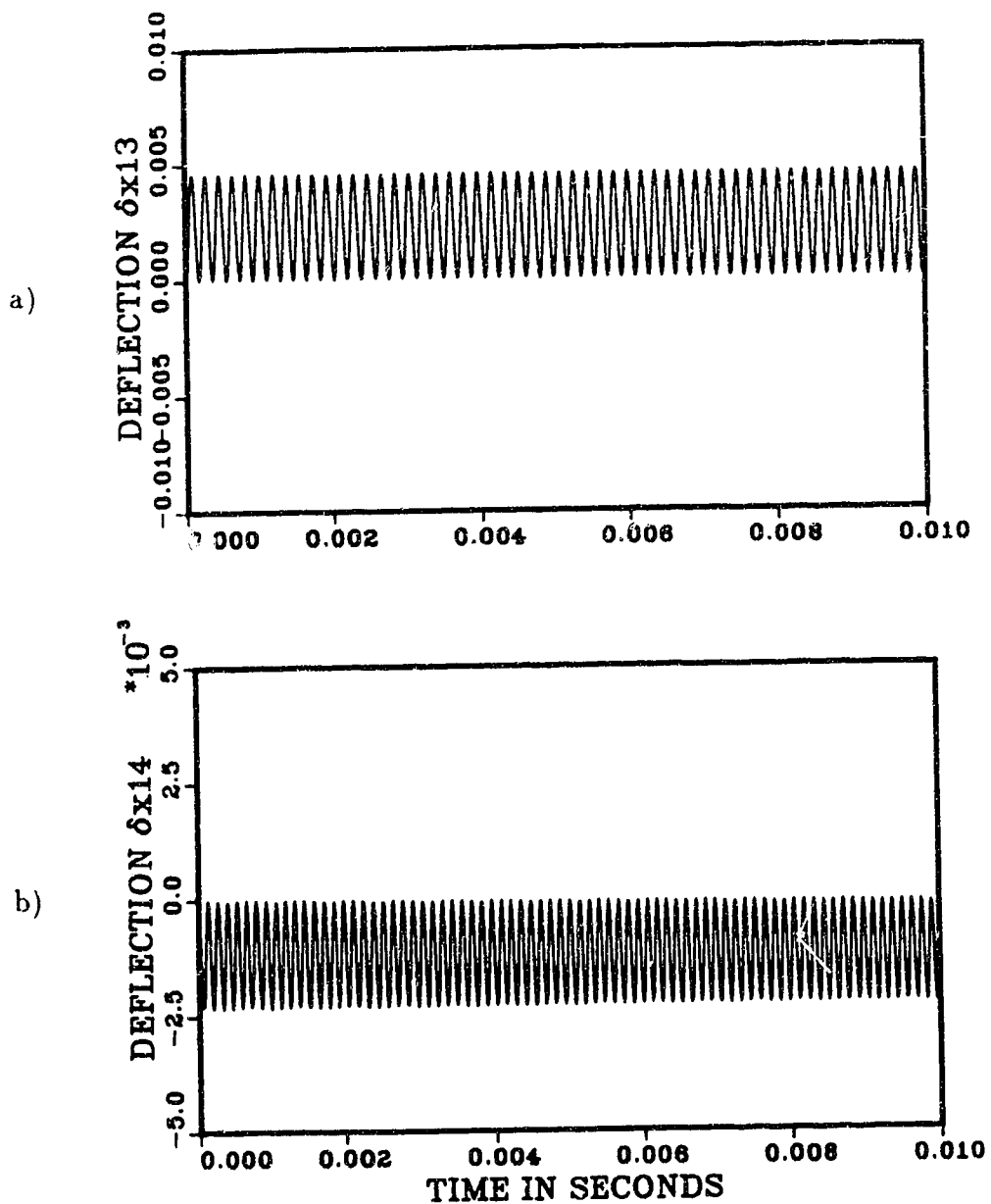


Figure 4.12: Response of torsion variables of a single flexible link: a) 3rd mode
b) 4th mode

Table 4.12: Comparison between the theoretical and simulated frequencies for a single flexible link in torsion.

mode	theoretical frequency Hz	simulated frequency Hz
1	1083	1083
2	3250	3252
3	5416	5414
4	7582	7591

Case 3: Double Compound Flexible Pendulum in Bending

This test case was designed to test the correct modeling of the rigid body dynamics and consists of a double compound pendulum with flexible links and a mass attached to the end of the second link as shown in figure 4.13. The equations of motion were first derived using the rigid link dynamics generator *HBMAT*. A second set of the equations of motion were generated using the flexible link dynamics generator *FLEXHB* and assuming flexible links with a stiffness (EI) equal to 10.178 Nm^2 . The second set were used to perform a simulation using the same initial conditions and integration time step. A third set of equations of motion similar to second set were generated assuming a link stiffness of 101.78 Nm^2 . Using these equations, simulations were performed using an integration time step of 0.5 ms and initial positions of -1.75 and -0.50 for the first and the second joints respectively. The response of the joint angles in the three simulations are shown in figure 4.14 for the first joint and in figure 4.15 for the second joint.

From these figures it is noticed that gross motion of the flexible links with a stiffness 10.178 Nm^2 (a in figures 4.14 and 4.15) resembles the motion of the completely rigid links (c in figures 4.14 and 4.15) with the superposition of vibrations

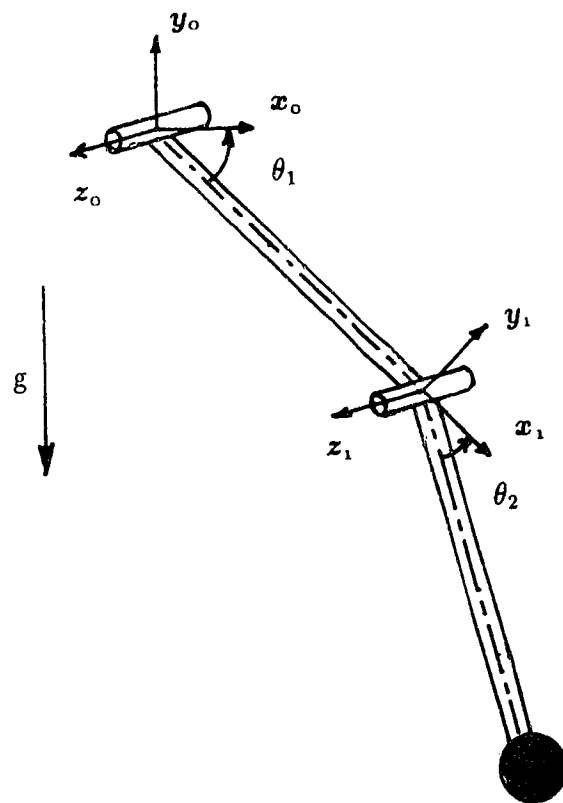


Figure 4.13: Double flexible compound pendulum

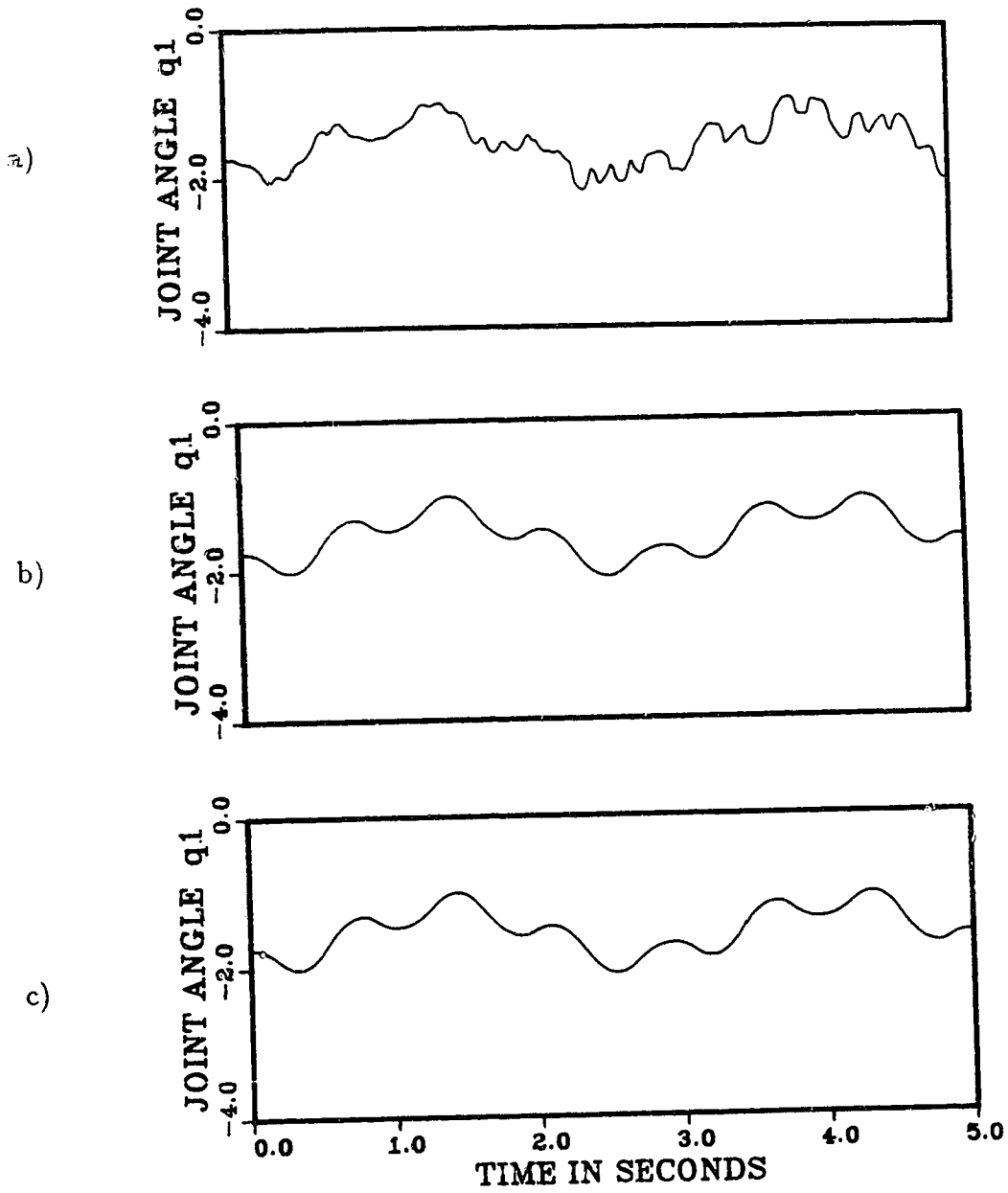


Figure 4.14: Joint angle responses for double compound flexible pendulum: a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$ c) rigid model

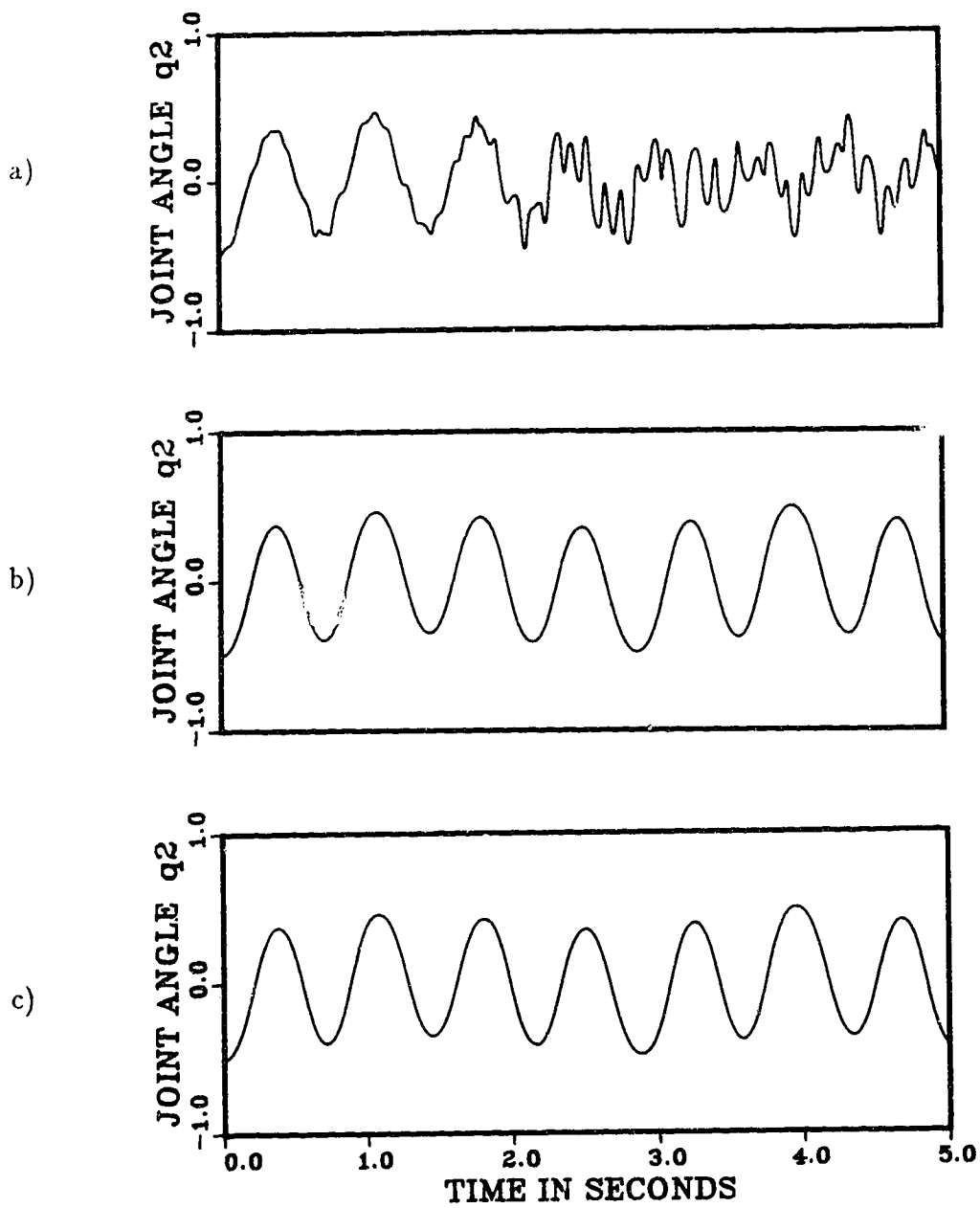


Figure 4.15: Joint angle responses for double compound flexible pendulum: a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$ c) rigid model

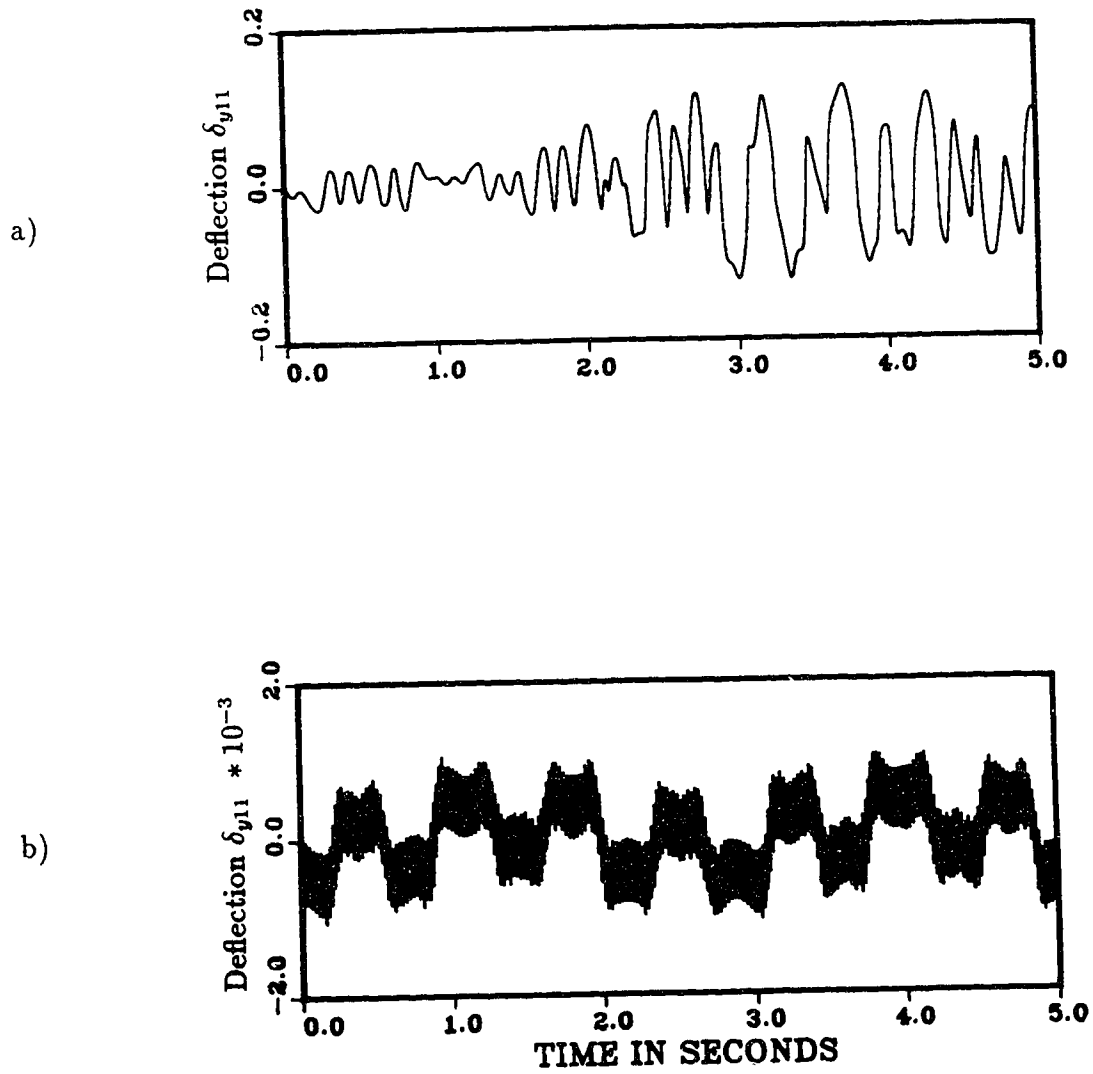


Figure 4.16: Response of bending variables for link 1 :a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$

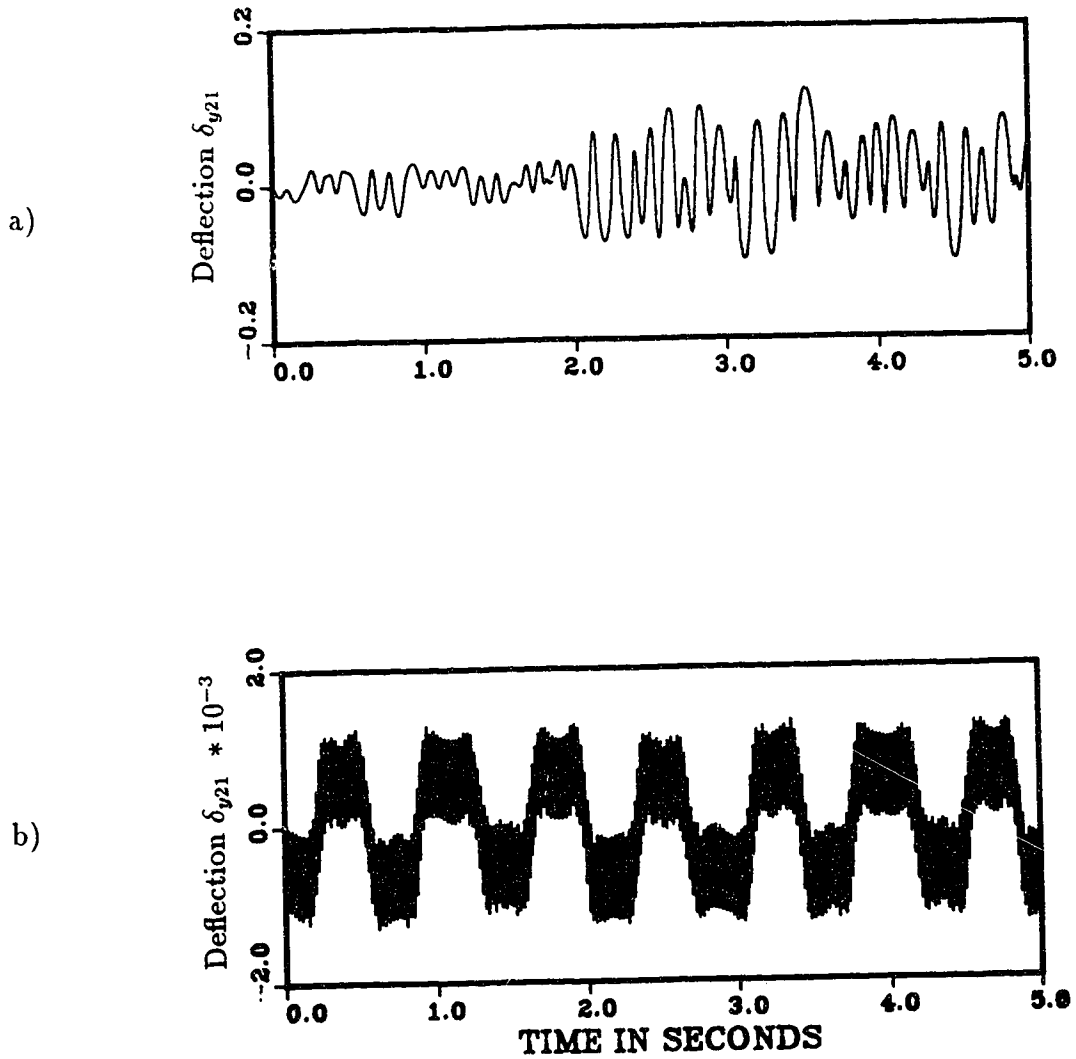


Figure 4.17: Response of bending variables for link 2 :a) $EI=10.178 \text{ Nm}^2$ b) $EI=101.78 \text{ Nm}^2$

of higher frequencies accounting for the vibrations of the links. When the links were given the higher stiffness of 101.78 Nm^2 , the response of the joint angle (b in figures 4.14 and 4.15) approached the response of the completely rigid links. As the stiffness increases, the vibration of the links occurs at higher frequencies and lower amplitudes as shown in figures 4.16 and 4.17. This test case indicated that rigid body motion is modelled correctly in the symbolic generator.

Case 4: Double Flexible Link Manipulator

This test case was performed to test the correct coupling of the various modes of vibrations of the different links. The manipulator used is a two flexible link manipulator shown in figure 4.18. The parameters of the two links are the same as in table 4.8 except for the mass of the link which is assigned a value of 1 Kg. Joint masses of 2 Kg were added to each link. Flexibility was modelled using one assumed mode per direction of flexibility. Several simulations were performed using the direct dynamic code symbolically generated by *FLEXHB* assuming different initial conditions. In these simulations, the gravitational term was ignored and the integration time step used was 1 ms. During these simulations the joints were made free to rotate by supplying zero torques to the joints.

Case 4.1

The first simulation was performed with the initial values of both the first joint and the second joint angles set to zero. The links were made flexible in the two directions of bending³. The deflection variable δ_{y21} was given an initial value of

³bending in the plane defined by the x and y axes of the link and bending in the plane defined by the x and z axes

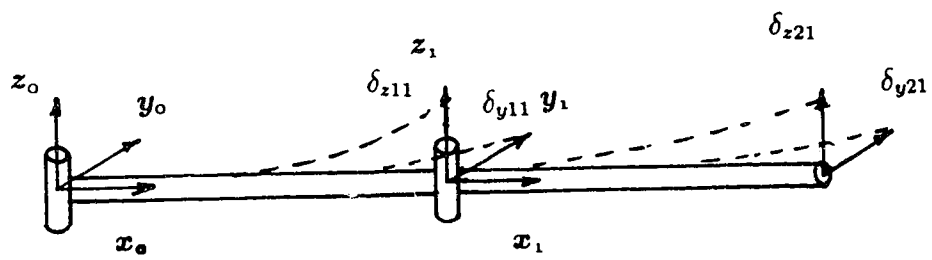


Figure 4.18: Two flexible link manipulator

0.002724 which is equivalent to a bending deflection of 5.448 mm in the xy plane. The system was then released with no torques supplied to any joint. The response of the deflection variables and joint angles are found in figure 4.19. Bending in the xy plane for the second link represented by the deflection variable δ_{y21} back drives the second joint q_2 with the same frequency but 180° out of phase. Bending in the xy plane for the first link represented by the deflection variable δ_{y11} also back drives the first joint with the same frequency and also 180° out of phase. The bending vibration of the second link in the xy plane induced bending vibration in the xy plane of the first link but it did not induce vibrations in the xz planes of either the first or the second link. The induced vibrations in the first link have the same frequency as the second link but the amplitude is different. This simulation indicates the correct coupling between the vibrations in the xy plane and the other modes of vibration in the previous links. The simulated frequency of oscillation of mode δ_{y21} was about 31.2 Hz which is closer to the theoretical natural frequency of the second link assuming pinned-free boundary condition model (≈ 35.01 Hz) than the model assuming clamped-free boundary condition (≈ 7.98 Hz). Although clamped-free mode shapes were implemented in the symbolic generator, the simulated frequency came closer to pinned-free frequency as the link was not actually clamped but pinned in this direction of vibration.

Case 4.2

The second simulation is similar to the first one as the initial values for the joint angles are set to zero and the links are made flexible in the same directions. The deflection variable δ_{z21} which represents the bending deflection in the xz plane is given an initial value of 0.002724 which is equivalent to a deflection of 5.448 mm. The response of the different variable are found in figure 4.20. The bending vibration

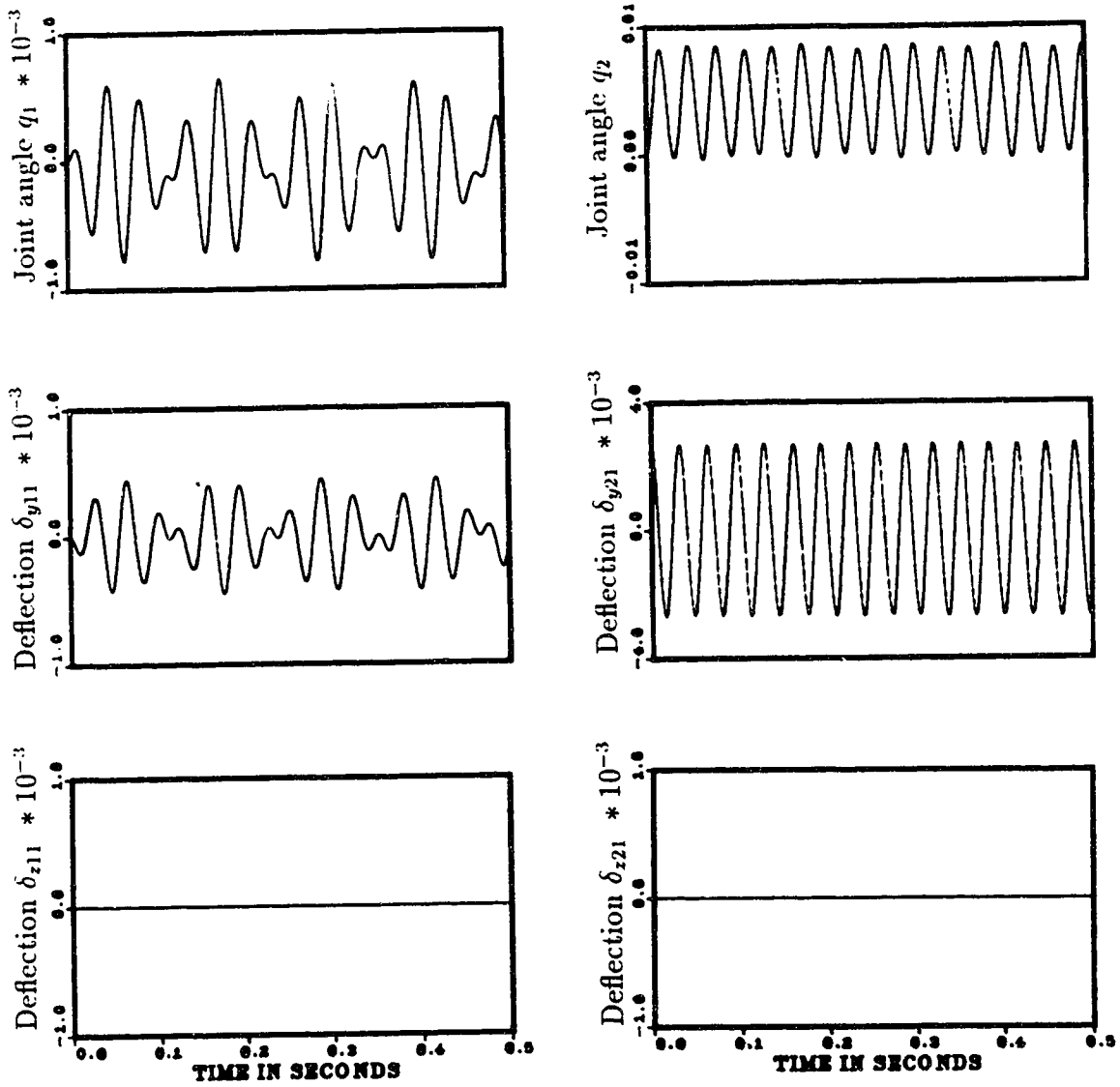


Figure 4.19: Responses of the manipulator in case 4.1

in the xz plane of the second link excited vibration in the xz plane of the first link with the same frequency. As expected, the vibrations in the xz plane did not excite any vibrations in the xy planes in either link nor did it excite vibrations in the joint angle as the directions of the initial vibrations are parallel to the joint axes. The simulated frequency of oscillations of mode δ_{z21} is about 11.3 Hz which is closer to the natural frequency of the link assuming clamped-free boundary condition (7.98 Hz) than the frequency assuming pinned-free boundary condition (35.01 Hz).

Case 4.3

In the third simulation, the first joint angle was set to zero and the second joint angle was set to 90° . The first link is made flexible in torsion in addition to the two directions of bending. The second link is made flexible in the two directions of bending. The deflection variable δ_{z21} which represents the bending deflection in the xz plane is given an initial value of 0.002724 which is equivalent to a deflection of 5.448 mm. The response of the different variables are found in figure 4.21. The bending vibration in the xz plane of the second link excited bending vibration of the same frequency in the xz plane of the first link and also excited torsional vibrations about the x axis of the first link. As the bending vibration of the second link exerts strong torques on the first link in this configuration, the amplitude of the torsional deflections⁴ of the first link (0.0058 rad) is significant. The first link is vibrating in the xz plane⁵ with two frequencies. The lower frequency (≈ 4 Hz) can be attributed to the natural frequency of the first link⁶. The higher frequency (≈ 38 Hz), which is superimposed on the lower frequency, could be attributed to the excitation by the

⁴represented by the generalized deflection variable δ_{x11}

⁵represented by the generalized deflection variable δ_{z11}

⁶approximated as a beam clamped in one end and has a concentrated mass in the other end

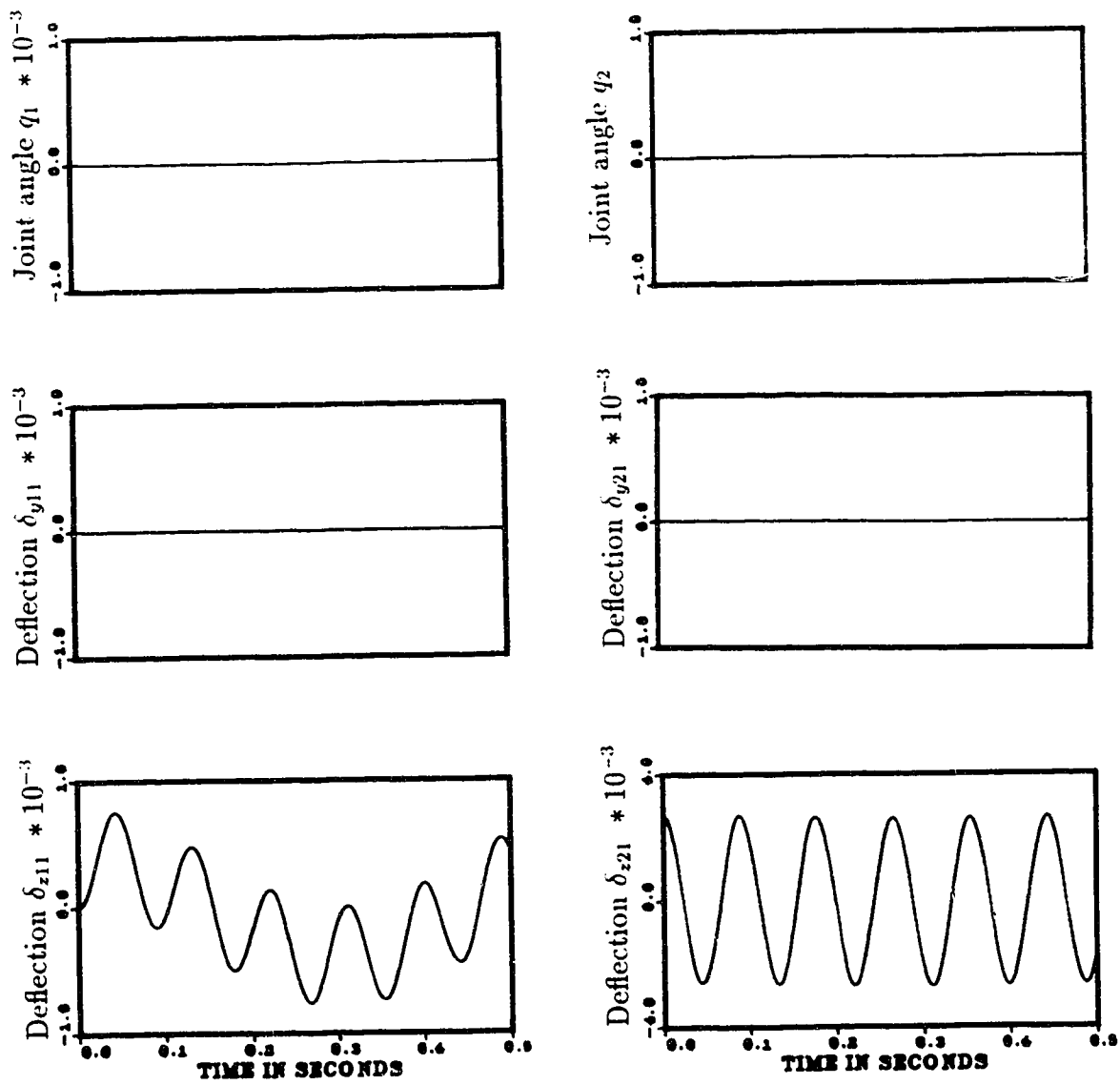


Figure 4.20: Responses of the manipulator in case 4.2

vibrating mode δ_{z21} of the second link. Very small vibrations (two orders of magnitude smaller) are induced in the xy planes of both the first and the second link. The joint angles are vibrating in the same manner as vibrations in the xy plane of both links but also with very small amplitudes.

Case 4.4

This test case geometry was identical to the previous test case but the system was excited differently. The first link was given an initial torsion of 0.002 rad and then released. The response of the different variables are shown in figure 4.22. The torsional vibrations excited the same modes of vibrations as in the previous test case. For example, it excited bending vibrations in the xz plane of the second link with the same frequencies as the previous test case. It also excited bending in the xz plane of the first link in the same manner as the previous test case. The amount of induced bending is smaller than the previous test case since the amplitude of the exciting torsional vibrations was lower.

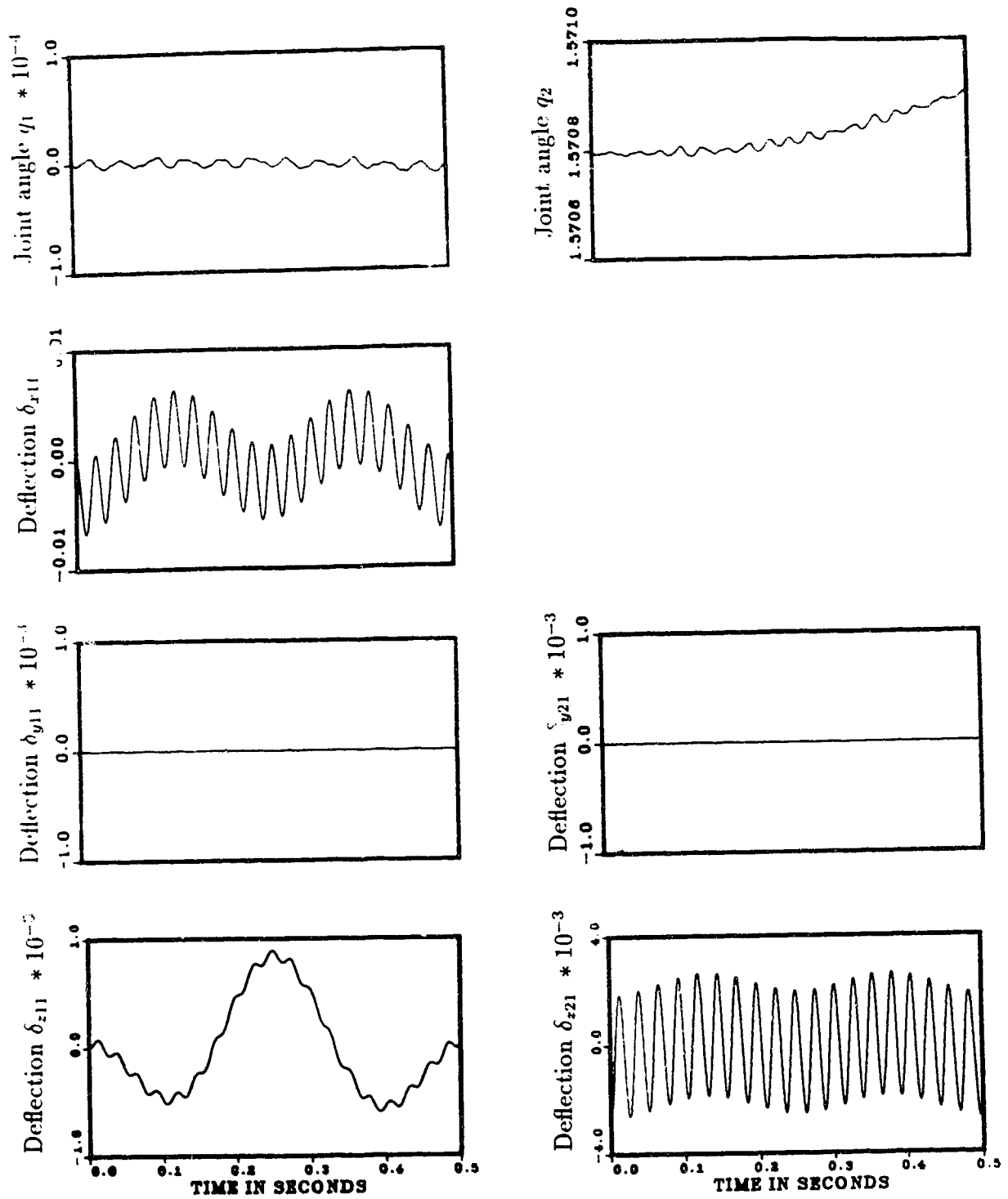


Figure 4.21: Responses of the manipulator in case 4.3

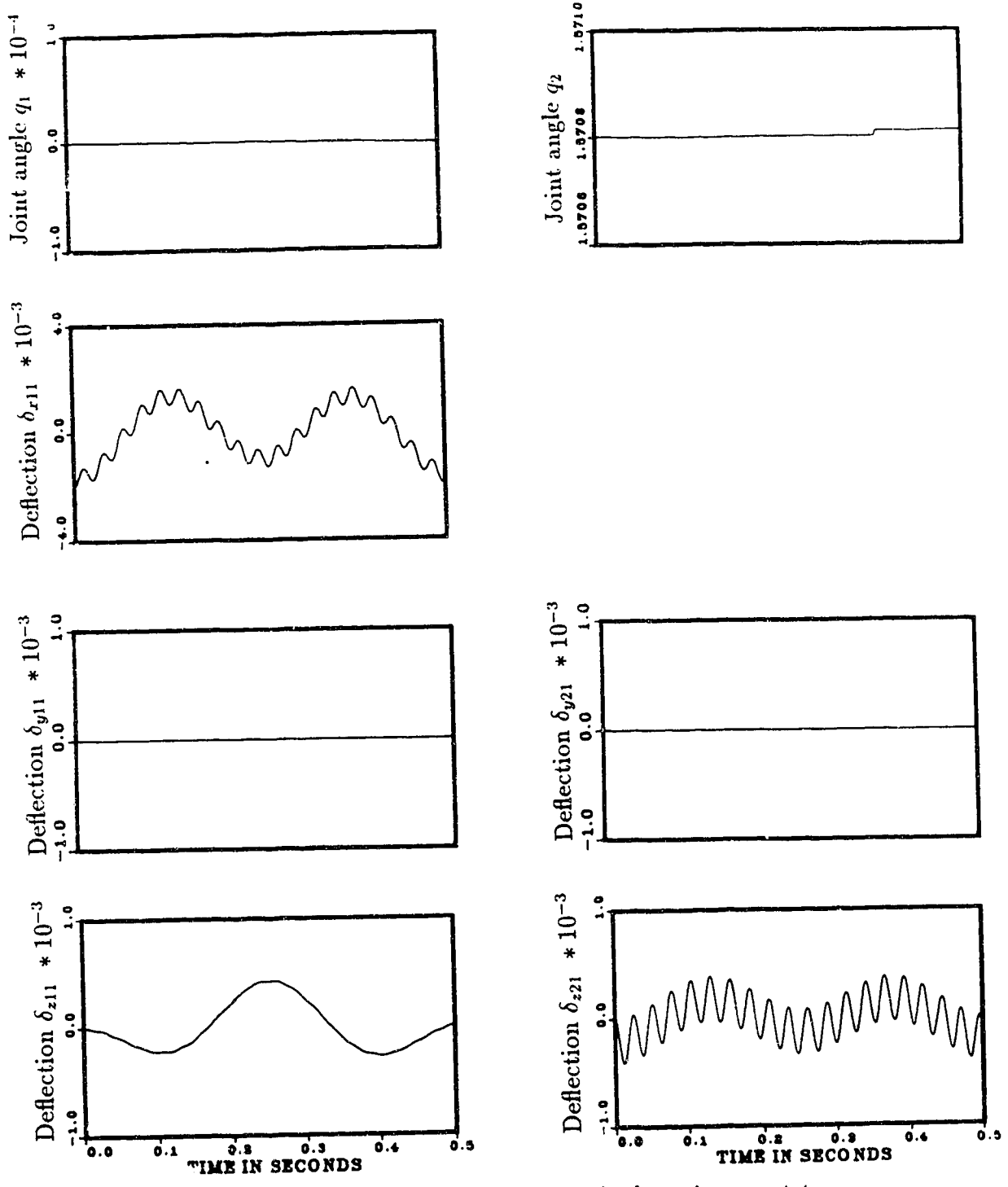


Figure 4.22: Responses of the manipulator in case 4.4

4.2.2 Verification

In summary, verification of the correct implementation of flexible link manipulator dynamics in the symbolic generator *FLEX* was performed first by comparing the performance of the symbolically generated dynamics to the analytically derived dynamics as demonstrated in the first two test cases. In the third test case it was proved that the flexible link dynamics converge to rigid link dynamics as the stiffness of the links increases. In the fourth test case the coupling between the various modes of vibrations of the different links and also between the flexible modes and rigid link motion were examined and found reasonable. This is not considered a rigorous proof but rather an indication of the correctness of both the dynamic model and the symbolic generation. Furthermore, the dynamics of test cases three and four were generated using Mackay's symbolic generator *FLXDYN* [26] which is based on Book's algorithm [4] that uses the recursive Lagrangian algorithm. Employing the same simplifying assumptions, the manipulator dynamics generated by *FLXDYN* were utilized to perform the same simulations as in test cases three and four. The outcomes of these simulations were identical (neglecting very small numerical errors) to the simulations performed using dynamic code generated by *FLEXHB*. Although almost totally different methods were used to generate the dynamics, the results of the simulations were in a very close agreements.

4.2.3 The Computational Requirements to Calculate the Direct Dynamics for Flexible Link Manipulators

As mention earlier, the advantage sought of the symbolic generation is the low computational requirements to calculate the dynamics. It is not only a matter of reducing the cost of simulation by making the dynamics more efficient, but it is also

a question of the feasibility of doing the simulation in the first place. As it will be illustrated later, employing symbolic generation techniques can considerably reduce the computational requirements to calculate the dynamics.

Several test cases were examined to determine the computational requirements for the direct dynamics code symbolically generated by *FLEXHB*. The computational requirements are compared to the computational requirements of a numerically programmed version of King's algorithm. Comparisons are also made between the computational requirements of the direct dynamics code symbolically generated by *FLEXHB* and the direct dynamics code produced using Mackay's generator *FLXDYN* and also with a numerically programmed version of Book's algorithm. Table 4.13 lists the different test cases employed in the comparison which vary in the complexity from 6 DOF to 14 DOF. The second column in the table specifies the number of links. The third column specifies the flexural freedom(s) of the links. "R" denotes that the link is rigid, "BB" denotes flexibility in the two directions of bending and "T" denotes flexibility in torsion. For example "R-TBB-BB" means that the first link is rigid, the second link is flexible in the two directions of bending in addition to torsion and the third link is flexible in the two directions of bending. The fourth column in the table specifies the number of assumed modes used to approximate each direction of flexibility. The fifth column lists the number of joint masses the manipulator possesses which makes the manipulator slightly more complex. The last column indicates whether or not the gravity term is included in the computations.

Tables 4.14 and 4.15 present the computational requirements to calculate the direct dynamics for the test cases presented in table 4.13 using *FLEXHB* and Mackay's *FLXDYN* symbolic generators respectively. The computational requirements of the symbolically generated code and an estimate of the computational requirements of the generic numerical code are presented for each test case. This estimate was

Table 4.13: The definitions of the cases used in the computational efficiency comparisons

Case	Number of Links N	Directions of Flexibility L	Number of Modes M	Joint Mass	Degrees of freedom $N + (L * M)$	Gravity Term Included
A	2	4 (BB-BB)	1	0	6	YES
B	2	4 (BB-BB)	1	0	6	NO
C	2	4 (BB-BB)	1	2	6	YES
D	2	4 (BB-BB)	1	2	6	NO
E	2	5 (TBB-BB)	1	0	7	YES
F	2	5 (TBB-BB)	1	0	7	NO
G	2	5 (TBB-BB)	1	2	7	YES
H	2	5 (TBB-BB)	1	2	7	NO
I	2	4 (BB-BB)	2	2	10	NO
J	2	5 (TBB-BB)	2	2	12	NO
K	3	4 (R-BB-BB)	1	0	7	YES
L	3	4 (R-BB-BB)	1	2	7	YES
M	3	5 (R-TBB-BB)	1	0	8	YES
N	3	5 (R-TBB-BB)	1	2	8	YES
O	3	4 (R-BB-BB)	2	2	11	YES
P	3	5 (R-TBB-BB)	2	2	13	YES
Q	6	4 (R-BB-BB-R-R-R)	1	0	10	YES
R	6	4 (R-BB-BB-R-R-R)	2	0	14	YES

obtained by counting the number of arithmetic operations in the original algorithm processed by the symbolic generator. These numbers include the number of arithmetic operations needed to calculate the bias vector and the inertia matrix. The number of arithmetic operations presented for the symbolically generated dynamics is the number of operations the dynamic code contained after being simplified by the post-processor *CLEAR* described in chapter three.

The first observation from table 4.14 is that the *FLEXHB* generated dynamics code is always more efficient, by a great margin, than the numerically programmed generic dynamics. It is noticed that the symbolically generated routine requires only between 2.8% and 5.7% of the amount of computations required by the generic numerical formulation. This means the symbolically generated customized dynamics are between 17.5 and 35.7 times more efficient than the generic numerical dynamics. The amount of saving is highly dependent on the inertial and geometric complexity of the manipulator under study. This observation is valid also for Mackay's *FLXDYN* symbolic generator. The dynamics generated by this generator requires only between 2.9% and 6.1% of the computations needed in Book's algorithm if programmed numerically.

Comparing the computational requirements of the generator *FLEXHB* with Mackay's generator *FLXDYN*, it can be concluded that *FLEXHB* is more efficient in computing the dynamics for these test cases. *FLEXHB* needed only between 50 % and 80 % of the arithmetic computations contained in dynamics codes generated by *FLXDYN*. The amount of saving is dependent on the configuration of the manipulator under study. The *FLEXHB* higher efficiency is due to the more efficient algorithm of King where the equations of motion are written in a Newton-Euler like formulation. This further proves that the higher efficiency of the Newton-Euler algorithm carries on to the flexible link dynamics from rigid dynamics. By examining the computational

requirements of the numerical algorithms of both Book and King, it is observed that numerically programmed version of King's algorithm is more efficient than Book's numerical algorithm.

From table 4.14 it is noticed that approximating vibrational deflections by including a higher number of assumed modes has a smaller effect on the computational burden than increasing the number of links. For example, adding an extra 5 DOF by approximating each direction of flexibility by two assumed modes (test case J) instead of one assumed mode as in (test case H) added 533 extra arithmetic operation or 107 operation for each extra degree of freedom. By examining the remaining test cases presented, it can be noticed that increasing the number of assumed modes used to approximate deflections will add between 85 and 130 arithmetic operations for each added mode of deflection.

Introducing torsion in the manipulator model increases the computational requirements to calculate the dynamics by a greater amount than by increasing the number of modes of flexure. For example, adding torsion increases the computational requirements of the previous test cases (e.g. going from case A to E, B to F, C to G, D to H and K to M) by about 180-200 arithmetic operations.

Introducing more links is the most computationally expensive. In the previous examples, the addition of an extra rigid link added between 400 and 500 arithmetic operations to the computational requirements.

Another interesting comparison in the computational requirements is between flexible manipulators and the same manipulators but assuming rigid links. Two extreme cases are examined. The first case is case A which is a two link manipulator (see figure 4.18). Assuming rigid links, the manipulator will have only 2 DOF instead of the 6 DOF if flexibility of links were considered. The rigid link model will reduce the computational requirements for this case from a total of 563 mathematical operations

to only 30. From these numbers it can be concluded that considering the flexibility of the links for this particular case will add considerably to the computational burden. However, we must keep in mind that the number of degrees of freedom were tripled when going from the rigid model to the flexible model. The second case is case Q which is a 6 link manipulator similar to the Puma 560 rigid manipulator except that links 2 and 3 were made flexible in both directions of flexure. To calculate the direct dynamics for the 6 DOF Puma 560 manipulator 768 arithmetic operations⁷ are needed compared to 2038 operation required for the 10 DOF case Q where flexibility is considered. The penalty of including flexibility in this case is not as heavy as the first case, due to the fact that the number of degrees of freedom are increased only by about 67 %. Therefore, modelling the flexibility of only the links that are likely to be susceptible to vibrations may result in dynamics codes that can be executed with reasonable speed.

A note worth mentioning is that for the same number of degrees of freedom, the computational requirements of the *FLEXHB* generated codes are slightly less than the computational requirement of rigid manipulators with the same number of degrees of freedom. For example for cases A through D in table 4.14 which have 6 DOF, the number of arithmetic operations required range between 559 and 636 operations. For the case of 6 DOF rigid link manipulators presented in table 4.5 the number of computations that are required to calculate the same dynamics range between 590 and 772 operations.

⁷using Walker and Orin's method 2

Table 4.14: The computational requirements of the generator FLEXHB

Case	DOF	Numerical			Symbolic			% Of Numerical Algorithm
		Mult.	Add.	Total	Mult.	Add.	Total	
A	6	5522	6004	11526	337	226	563	4.9
B	6	5522	6004	11526	336	223	559	4.9
C	6	9304	9952	19256	372	267	639	3.3
D	6	9304	9952	19256	372	264	636	3.3
E	7	6540	7146	13686	443	306	749	5.5
F	7	6540	7146	13696	441	302	743	5.4
G	7	10862	11668	22530	482	354	836	3.7
H	7	10862	11668	22530	480	350	830	3.7
I	10	17108	18152	35260	553	424	977	2.8
J	12	20966	22410	43376	767	596	1363	3.1
K	7	8468	9116	17584	566	398	964	5.5
L	7	12790	13628	26418	635	483	1118	4.2
M	8	9785	10587	20372	675	482	1157	5.7
N	8	14647	15663	30310	752	576	1328	4.4
O	11	21900	23184	45084	878	695	1573	3.5
P	13	26356	28080	54436	1104	876	1980	3.6
Q	10	20546	21836	42382	1172	866	2038	4.8
R	14	31414	33204	64618	1691	1286	2977	4.6

Table 4.15: The computational requirements of the generator FLXDYN

Case	DOF	Numerical			Symbolic			% Of Numerical Algorithm
		Mult.	Add.	Total	Mult.	Add.	Total	
A	6	9982	9960	19942	579	368	947	4.8
B	6	9982	9960	19942	569	360	929	4.7
C	6	9982	9960	19942	582	380	962	4.8
D	6	9982	9960	19942	574	372	946	4.7
E	7	12724	12701	25425	640	416	1056	4.2
F	7	12724	12701	25425	627	406	1033	4.1
G	7	12724	12701	25425	644	428	1072	4.2
H	7	12724	12701	25425	632	418	1050	4.1
I	10	21106	20920	42026	918	602	1520	3.6
J	12	29720	29504	59224	1026	679	1705	2.9
K	7	14067	14092	28159	1096	722	1818	6.5
L	7	14067	14092	28159	1141	803	1944	6.9
M	8	17145	17189	34334	1262	830	2092	6.1
N	8	17145	17189	34334	1313	913	2226	6.5
O	11	26023	25900	51923	1848	1257	3105	6.0
P	13	35309	35196	70505	2112	1461	3573	5.1
Q	10	30602	30688	61290	2202	1516	3718	6.1
R	14	N/A*	-	-	-	-	-	-

* Not available

Chapter 5

Summary and Conclusions

The production of computationally efficient dynamics for manipulators is very essential in both the initial robot design stage and later in the operational stage. In the design stage the expected behaviour of the robot under different conditions is simulated to optimize the design parameters of the robot. In the operational stage simulation can provide verification of the ability of the robot to perform certain tasks. Furthermore, as the current trend is in the direction of employing faster robots, control algorithms are being developed which incorporate the dynamics of the manipulator in their control schemes to account for the dynamic coupling that is associated with faster moving robots.

Several methods can be used to model the dynamics of manipulators. In chapter two, a description was given of the recursive Newton-Euler method of Luh et al. [25] which is one of the most efficient methods to derive the dynamics for rigid link manipulators. Flexible link manipulators were also treated in this work. The algorithm implemented for these manipulators was King's [22] modified Lagrangian scheme which is written in a Newton-Euler like recursive formulation .

As an answer to the need for more efficient dynamics code, symbolic generation was introduced as an alternative to numerical programming. The Newton-Euler method for rigid link manipulators was programmed in the symbolic generators *NEDYN*¹, *HMAT*², *BVECT*³ and *HBMAT*⁴. An alternative method to generate the manipulator Jacobian matrix was implemented in the symbolic generator *JACOB*. King's method for flexible link manipulators was programmed in the symbolic generators *FLEX*⁵, *FLEXH*⁶, *FLEXB*⁷ and *FLEXHB*⁸. A description of the different techniques employed in the symbolic generation was given in chapter three. Following the generation process, a post-processor called *CLEAR* is utilized to further simplify and optimize the generated dynamics. This thesis involved development and implementation of many important improvements in the operation of *CLEAR*. New added procedures such as renaming of duplicate multiplications terms and pre-computations of terms involving constants greatly improved the simplifying power of *CLEAR*. A typical run of the post-processor *CLEAR* will eliminate 50 % of the arithmetic calculations contained in the code produced by the symbolic generators. These improvements in *CLEAR* enabled the production of very efficient dynamics code for both rigid and flexible link manipulators.

Several test cases were studied to verify both the dynamic models and the correct implementation of the different algorithms in the symbolic generators. Although these

¹Inverse dynamics

²Inertia matrix

³Bias vector

⁴Direct dynamics

⁵Inverse dynamics

⁶Inertia matrix

⁷Bias vector

⁸Direct dynamics

test cases do not constitute rigorous proofs, they nevertheless indicate the soundness of the dynamics algorithms and their implementation in the symbolic generators. The verification procedure included some of the following tests:

- Agreements between the algorithm and analytical solutions for simple cases.
- Cross-checking between the dynamics codes for the same manipulators which are produced using totally different algorithms. For example the cross-checking between dynamics code generated by *DYNAM* which is based on the recursive Lagrangian algorithm and *NEDYN* which is based on the Newton-Euler algorithm.
- For the flexible manipulators, the dynamic model for flexible links should converge to the dynamic model for rigid links if stiffness is increased.
- Dynamic models should demonstrate closure.
- Reasonable qualitative behaviour of the coupling between the different generalized variables in the model.

The implementation of the symbolic generation techniques resulted in great reductions in the computational requirements for both rigid link and flexible link manipulators. As illustrated in chapter four, the computational requirements of the symbolically generated direct dynamics for flexible link manipulators by *FLEXHB* is only about 2.8 to 5.7% of the computational requirement of the generic numerically programmed dynamics. The same note applies to the dynamics of rigid manipulators, where the computational efficiency of the symbolically generated dynamics greatly surpasses the efficiency of the numerically programmed version.

For the case of rigid link manipulators, the higher computational efficiency of the numerical Newton Euler method carried on to the symbolically generated

dynamics. The symbolically generated dynamics codes using *NEDYN* are generally more efficient than generators implementing other methods especially for higher degrees of freedom and recursive dynamics. For these cases, the dynamic codes generated by *NEDYN* required only about half of the computations needed by the corresponding dynamic code generated by *DYNAM* [42, 43] which is based on Hollerbach's recursive Lagrangian method. The computational requirements of the dynamic code generated by the symbolic generator *NEDYN* matched (and in many cases surpassed) the most efficient dynamics reported in literature to date including Neuman and Murray's *ARM* generator [27, 28, 29, 30, 31].

For the case of flexible link manipulators the implementation of King's algorithm [22] also resulted in great savings in the number of computations. In this algorithm, a modified Lagrangian method is used where the equations are written in a Newton-Euler like structure to capitalize on the inherent efficiency of the Newton-Euler method. Therefore the numerically programmed version of King's algorithm is more efficient than other Lagrangian based algorithms including Book's [4] algorithm which is one of the successful algorithms reported in the literature. Similar to the rigid link dynamics, the higher efficiency of King's algorithm (compared to Book's algorithm) is carried on to the symbolically generated dynamics code. As reported in chapter four, the dynamic code generated by the symbolic generator *FLEXHB* requires only 50 to 80 % of the number of computations needed in code generated by Mackay's *FLXDYN* [26]. The computational cost of adding more degrees of freedom to the manipulator model varies depending on the nature of the new degrees of freedom. For example, it is more computationally expensive to add a new rigid link than to increase the number of assumed modes that approximate a certain deflection.

The important contributions of this thesis include the presentation of very efficient methods to generate the dynamics of rigid and flexible link manipulators using

symbolic generation techniques. Furthermore, to the author's knowledge, this is the first known systematic documentation of the computational requirements for the flexible link manipulators, with the exception of Mackay's work [26].

As a final note, the following problems could be recommended for further investigations:

- The simplification procedures in the post-processor *CLEAR* could be reexamined to determine if more efficient simplifications could be achieved.
- The symbolic generation technique showed great utility in the area of robot dynamics. Therefore it should be investigated if the benefits of this technique could be realized in other problems in the dynamics of mechanisms such as helicopter rotors or multi-legged robots.
- Experimental verification especially for the flexible link models.
- With the interest in parallel computation, the problem of automatic partitioning of the dynamics code for execution on a number of independent processors could be explored.

References

- [1] R.E. Bishop and D.C. Johnson, *The Mechanics of Vibration*, Cambridge University Press, Cambridge, 1979.
- [2] S.K. Biswas and R.D. Klafter, "Dynamic Modeling and Optimum Control of Flexible Robotic Manipulator", *IEEE 1988 International Conference on Robotics and Automation*, Vol 1, pp 15-20, 1988.
- [3] W.J. Book, O. Maizza-Neto and D.E. Whitney, "Feedback Control of Two Beam, Two Joint Systems With Distributed Flexibility", *ASME Journal of Dynamic Systems, Measurement and Control*, Vol 97, No. 4, Dec., pp 424-430, 1975.
- [4] W.J. Book, "Recursive Lagrangian Dynamics of Flexible Manipulators", *The International Journal of Robotics Research*, Vol. 3, No. 3, pp 87-101, 1984.
- [5] M. Brady , J. Hollerbach , T. Johnson , T. Lozano-Perez, and M. Mason, *Robot Motion: Planning and Control*, The MIT Press, Cambridge, Mass, 1982.
- [6] S. Cetinkunt, B. Siciliano, W.J. Book, " Symbolic Modelling and Dynamic Analysis of Flexible Manipulators", *1986 IEEE International Conf. Systems, Man, and Cybernetics*, Atlanta, pp 798-803, 1986.
- [7] S. Cetinkunt and W.J. Book, "Symbolic modeling of Flexible Manipulators", *1987 IEEE Conference on Robotics and Automation*, pp 2074-2080, 1987.

- [8] S. Cetinkunt and W.J. Book, "Symbolic Modeling and Dynamic Simulation of Robotic Manipulators with Compliant Links and Joints", *Robotics & Computer-Integrated Manufacturing*, Vol.5, No.4, pp 301-310, 1989.
- [9] S.C. Chapra and R.P. Canale, *Numerical Methods For Engineers*, 2nd Ed., McGraw Hill Book company, pp 288-291, 1988.
- [10] R.W. Clough, *Dynamics of Structures*, McGraw-Hill, pp 328-335, 1975.
- [11] J. Denavit and R.S. Hartenberg, "A Kinematic Notation for lower-Pair Mechanisms", *ASME J. of Applied Mechanics*, June, pp 215-221, 1955.
- [12] S.R. Dillon, *Computer Assisted Equation Generation in Linkage Dynamics*, PhD dissertation, Dept. of Electrical Engineering, Ohio State University, Columbus, OH, August, 1973.
cf. Neuman and Murray [28]
- [13] H. Goldstein, *Classical Mechanics*, 2nd ed., Addison-Wesley Publishing, 1980.
- [14] G.G. Hasting and W.J. Book, "A linear Dynamic Model for Flexible Robotic Manipulators", *IEEE Control System Magazine*, 7(1), Feb., pp 61-64, 1987
- [15] J.M. Hollerbach, "A Recursive Formulation of Lagrangian Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity". *IEEE Trans. Systems, Man, and Cybernetics* SMC-10, pp 730-736, 1980.
- [16] P.C. Hughes, "Dynamics of a Flexible Manipulator Arm for Space Shuttle", Paper 28, *Proceeding of the American Astronautical Society and American Institute of Aeronautics and Astrodynamicics Specialist Conference*, Jackson Hole, Wyoming, Sept. 7-9, 1977.

- [17] M.A. Hussain and B. Noble, "Application of Symbolic Computation to the Analysis of Mechanical Systems, Including Robot Arms", in *Computer Aided Analysis and Optimization of Mechanical System Dynamics*, NATO ASI Series, Vol. F9 (E.J. Haug, ed.), Springer-Verlag, Berlin, pp 283-306, 1984.
- [18] T.R. Kane and D.A. Levinson, "The Use of Kane's Dynamical Equations in Robotics", *The Int'l. J. of Robotics Research*, Vol. 2, No. 3, pp 3-21, 1983.
- [19] I. Kermack, *The Effect of Misalignments on Manipulator Performance*, M.Sc. Thesis, Dept. of Mech. Eng., University of Alberta, 1986.
- [20] W. Khalil and J. Kleinfinger, "Minimum Operations and Minimum Parameters of the Dynamic Models of Tree Structure Robots", *IEEE Journal of Robotics and Automation*, Vol. RA-3, No.6, pp 517-526, 1987.
- [21] J.O. King, V.G. Gourishankar and R.E. Rink, "Lagrangian Dynamics of Flexible Manipulators Using Angular Velocities Instead of Transformation Matrices", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-17, No. 11, pp 1059-1068, 1987.
- [22] J.O. King, *Recursive Models and Controllers of Flexible Manipulators*, Ph.D. Thesis, Dept. of Elect. Eng., University of Alberta, 1988.
- [23] M.B. Leahy, Jr., L.M. Nugent, G.N. Saridis, and K.P. Valavanis, "Efficient PUMA Manipulator Jacobian Calculation and Inversion", *Journal of Robotic Systems*, 4(2), pp 185-197, 1987.
- [24] J.Y.S. Luh and C.S. Lin, "Automatic Generation of Dynamic Equations for Mechanical Manipulators", *Proc. Joint Automatic Control Conf.*, Charlottesville, Va., 1981.

- [25] J. Luh, M. Walker and R.P. Paul, "On-line Computational Scheme for Mechanical Manipulators", *ASME J. of Dynamic Systems, Measurement, and Control*, Vol. 102, 1980.
- [26] D.J. Mackay, *A Simulator for Robots with Flexible Links*, M.Sc. Thesis, Dept. of Mech. Eng., University of Alberta, 1988.
- [27] J.J. Murray and C.P. Neuman, "ARM: An Algebraic Robot Dynamic Modeling Program", *Proc. Int'l. Conf. Robotics*, Atlanta, Ga., pp 103-113, 1984.
- [28] C.P. Neuman and J.J. Murray, "Computational Robot Dynamics: Foundations and Applications", *Journal of Robotic Systems*, Vol.2, No.4, pp 425-452, 1985.
- [29] C.P. Neuman and J.J. Murray, "Customized Computational Robot Dynamics", *Journal of Robotic Systems*, Vol.4, No.4, pp 503-526, 1987.
- [30] C.P. Neuman and J.J. Murray, "Symbolically Efficient Formulations for Computational Robot Dynamics", *Journal of Robotic Systems*, Vol.4, No.6, pp 743-769, 1987.
- [31] J.J. Murray and C.P. Neuman, "Organizing Customized Robot Dynamics Algorithms for Efficient Numerical Evaluation", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol.18, No.1, pp 115-125, 1988.
- [32] P.E. Nielan and T.R. Kane, "Symbolic Generation of Efficient Simulation/Control Routines for Multibody Systems in Dynamics of Multibody Systems", *IUTAM/IFTOMM Symposium Udine, Italy* (G. Bianchi and W. Schiehlen, eds.), Springer-Verlag, Berlin, pp 153-164, 1986.
- [33] S. Nicosia, P. Tomei and A. Tornambe, "Dynamic Modelling of Flexible Robot Manipulators", *1986 IEEE*, pp 365-472, 1986.

- [34] R.P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, The MIT Press, Cambridge, Mass., 1981.
- [35] R. Paul, M. Renaud, and C.N. Stevenson, "A Systematic Approach for obtaining the Kinematics of Recursive Manipulators Based on Homogeneous Transformations", *Robotics Research*, M. Brady and R. Paul (ed.), The MIT Press, Cambridge, Mass., 1984.
- [36] D.E. Orin and W.W. Schrader, "Efficient Jacobian Determination for Robot Manipulators", *Robotics Research*, M. Brady and R. Paul (ed.), The MIT Press, Cambridge, Mass., 1984.
- [37] Robotic System International Ltd, *Excalibur User's Manual*, Sidney, B.C., Canada, 1986.
- [38] D.E. Rosenthal and M.A. Sherman, "Symbolic Multibody Equations via Kane's Method", *AAS/AIAA Specialist Conference*, paper 83-803, Lake Placid, New York.
- [39] R.P. Singh and P.W. Likins, "Manipulator Interactive Design with Interconnected Flexible Elements", *Proceedings of the 1983 Automatic Control conference*, The American Control Council, San Francisco, Ca., June, 1983.
- [40] W. Sunada and S. Dubowsky, "On the Dynamic Analysis and Behavior of Industrial Robotic Manipulators with Elastic Members", *Transactions of the ASME J. Mech., Trans., and Automation in Design*, 105, pp 42-51, 1983.
- [41] M.W. Silver, "On the Equivalence of Lagrangian and Newton-Euler Dynamics for Manipulators", *Proc. of the 1981 Joint Automatic Control Conference*, The American Automatic Control Council, June, San Francisco, Ca., 1983.

- [42] R.W. Toogood, *Symbolic Generation of Robot Dynamics Equations Part I: The DYNAM/CLEAR System*, ACMIR Technical Report 87-04, University of Alberta.
- [43] R.W. Toogood, *Symbolic Generation of Robot Dynamics Equations Part II: Case Studies Using the DYNAM/CLEAR System*, ACMIR Technical Report 87-05, University of Alberta.
- [44] P. Tomei and A. Tornambe, "Approximate Modeling of Robots Having Elastic Links", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 18, No. 5, Sept/Oct, pp 831- 839, 1988.
- [45] P.B. Usoro, R. Nadira and S.S. Mahil, "Control of Light Weight Flexible Manipulators: A Feasibility Study. *Proceedings of the 1984 Automatic Control conference*, San Diego, Calif. 2, pp 1209-1216, 1984.
- [46] L. Vecchio, S. Nicosia, F. Nicolo and D. Lentini, "Automatic Generation of Dynamical Models of Manipulators", *Proceedings of the Tenth International Symposium on Industrial Robots*, Milan, Italy, March, pp 293-301, 1980.
cf. Neuman and Murray [28]
- [47] C.W. Wampler, *Computer Methods in Manipulator Kinematics, Dynamics, and Control: A Comparative Study*, Doctoral Thesis, Stanford University, 1984.
- [48] M.W. Walker and D.E. Orin, "Efficient Dynamic Computer Simulation of Robotic Mechanisms", *ASME J. Dynamic Systems, Measurement, and Control*, Vol 104, pp 205-211, Sept. 1982.
- [49] R.C. Waters, *Mechanical Arm Control*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, AIM 549, 1979.

Appendix A

Newton-Euler Formulation for Rigid Link Manipulators

In this appendix, the computational scheme of the recursive Newton-Euler algorithm of Luh et al. [25] for rigid link manipulators is presented. This formulation is based on D'Alembert's principle which states "the elements of the system will be in equilibrium under a force equal to the actual force plus a reversed effective force" [13]. This reversed effective force equals the time rate of change of linear momentum. This is equivalent to Newton's second law of motion. Euler's equations relates torques and angular momentums in a similar fashion. For a system that has elements of constant mass, Newton's second law of motion reduces to

$$\mathbf{F} = m\dot{\mathbf{v}} \quad (\text{A.1})$$

For manipulators, equation A.1 relates the resultant force vector \mathbf{F} acting on the link to the acceleration $\dot{\mathbf{v}}$ of the center of mass. Euler's equation

$$\mathbf{N} = \mathbf{I} \cdot \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I} \cdot \boldsymbol{\omega}) \quad (\text{A.2})$$

is used to relate the resultant torque \mathbf{N} about the center of mass to the inertia about the center of mass \mathbf{I} , the angular velocity $\boldsymbol{\omega}$ and angular acceleration $\dot{\boldsymbol{\omega}}$. The form of the equations makes this formulation naturally recursive. The kinematics of the links are calculated recursively starting with the first link and approaching the end effector. Then the dynamic computations are performed serially starting with the last link and working backward to the first link.

Section A.1 presents the forward recursion. In this run the kinematics of links are calculated starting at the first link and moving toward the end effector. Section A.2 presents the backward recursion which calculates the dynamics of the links. This run starts at the end effector and proceeds towards the base link.

A.1 Kinematics of the Links

Link $i = 0$

$$\begin{aligned} {}^0\boldsymbol{\omega}_0 &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ {}^0\dot{\boldsymbol{\omega}}_0 &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ {}^0\mathbf{v}_0 &= \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \\ {}^0\dot{\mathbf{v}}_0 &= \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \end{aligned}$$

Link $i = 1 \dots N$

$${}^i\boldsymbol{\omega}_i = \begin{cases} {}^i\mathbf{A}_{i-1} ({}^{i-1}\boldsymbol{\omega}_{i-1} + {}^{i-1}\mathbf{z}_{i-1}\dot{q}_i) & \text{rotational;} \\ {}^i\mathbf{A}_{i-1} {}^{i-1}\boldsymbol{\omega}_{i-1} & \text{translational.} \end{cases}$$

$${}^i\dot{\boldsymbol{\omega}}_i = \begin{cases} {}^i\mathbf{A}_{i-1} [{}^{i-1}\dot{\boldsymbol{\omega}}_{i-1} + {}^{i-1}\mathbf{z}_{i-1}\ddot{q}_i + {}^{i-1}\boldsymbol{\omega}_{i-1} \times ({}^{i-1}\mathbf{z}_{i-1}\dot{q}_i)] & \text{rotational;} \\ {}^i\mathbf{A}_{i-1} {}^{i-1}\dot{\boldsymbol{\omega}}_{i-1} & \text{translational.} \end{cases}$$

$${}^i\mathbf{v}_i = \begin{cases} {}^i\boldsymbol{\omega}_i \times {}^i\mathbf{p}_i^* + {}^i\mathbf{A}_{i-1} {}^{i-1}\mathbf{v}_{i-1} & \text{rotational;} \\ {}^i\mathbf{A}_{i-1} [{}^{i-1}\mathbf{z}_{i-1}\dot{q}_i + {}^{i-1}\mathbf{v}_{i-1}] + {}^i\boldsymbol{\omega}_i \times {}^i\mathbf{p}_i^* & \text{translational.} \end{cases}$$

$${}^i\dot{\mathbf{v}}_i = \begin{cases} {}^i\dot{\boldsymbol{\omega}}_i \times {}^i\mathbf{p}_i^* + {}^i\boldsymbol{\omega}_i \times ({}^i\boldsymbol{\omega}_i \times {}^i\mathbf{p}_i^*) + {}^i\mathbf{A}_{i-1} {}^{i-1}\dot{\mathbf{v}}_{i-1} & \text{rotational;} \\ {}^i\mathbf{A}_{i-1} ({}^{i-1}\mathbf{z}_{i-1}\ddot{q}_i + {}^{i-1}\dot{\mathbf{v}}_{i-1}) + {}^i\dot{\boldsymbol{\omega}}_i \times {}^i\mathbf{p}_i^* + \\ 2 {}^i\boldsymbol{\omega}_i \times ({}^i\mathbf{A}_{i-1} {}^{i-1}\mathbf{z}_{i-1}\dot{q}_i) + {}^i\boldsymbol{\omega}_i \times ({}^i\boldsymbol{\omega}_i \times {}^i\mathbf{p}_i^*) & \text{translational} \end{cases}$$

Velocity of the center of mass ${}^i\hat{\mathbf{v}}_i$

$${}^i\hat{\mathbf{v}}_i = {}^i\boldsymbol{\omega}_i \times {}^i\hat{\mathbf{s}}_i + {}^i\mathbf{v}_i$$

$${}^i\dot{\hat{\mathbf{v}}}_i = {}^i\dot{\boldsymbol{\omega}}_i \times {}^i\hat{\mathbf{s}}_i + {}^i\boldsymbol{\omega}_i \times ({}^i\boldsymbol{\omega}_i \times {}^i\hat{\mathbf{s}}_i) + {}^i\dot{\mathbf{v}}_i$$

A.2 Dynamics of the Links

Link $i = N+1$

${}^{N+1}\mathbf{f}_{N+1}$ = External forces applied to the end effector

${}^{N+1}\mathbf{n}_{N+1}$ = External moments applied to the end effector

Link $i = N \dots 1$

$${}^i\mathbf{F}_i = M_i {}^i\hat{\mathbf{v}}_i$$

$${}^i\mathbf{N}_i = {}^i\mathbf{J}_i {}^i\dot{\boldsymbol{\omega}}_i + {}^i\boldsymbol{\omega}_i \times ({}^i\mathbf{J}_i {}^i\boldsymbol{\omega}_i)$$

$${}^i\mathbf{f}_i = {}^i\mathbf{A}_{i+1}({}^{i+1}\mathbf{f}_{i+1}) + {}^i\mathbf{F}_i$$

$${}^i\mathbf{n}_i = {}^i\mathbf{A}_{i+1}[{}^{i+1}\mathbf{n}_{i+1} + ({}^{i+1}\mathbf{A}_i {}^i\mathbf{p}_i^*) \times {}^{i+1}\mathbf{f}_{i+1}] + ({}^i\mathbf{p}_i^* + {}^i\hat{\mathbf{s}}_i) \times {}^i\mathbf{F}_i + {}^i\mathbf{N}_i$$

$$\tau_i = \begin{cases} {}^i\mathbf{n}_i^T \cdot ({}^i\mathbf{A}_{i-1} {}^{i-1}\mathbf{z}_{i-1}) & \text{rotational;} \\ {}^i\mathbf{f}_i^T \cdot ({}^i\mathbf{A}_{i-1} {}^{i-1}\mathbf{z}_{i-1}) & \text{translational.} \end{cases}$$