PARALLEL IMPLEMENTATION OF MULTIBODY DYNAMICS FOR REAL-TIME SIMULATION

By

Darrell Wong

B.E. (Electrical) University of Auckland, 1981.

M.A.Sc. (Electrical) University of British Columbia, 1985.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

May 1991

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date 29 July 1991

DE-6 (2/88)

Abstract

A multibody dynamics formulation has been developed for the purposes of real-time simulation of large scale robotic mechanisms such as excavators. The formulation models the rigid body dynamics of any arbitrary tree structured mechanism, although at present the formulation is restricted to single degree of freedom rotational joints. This formulation is an example of the orthogonal complement approach, which describes the dynamics by projecting an initial description of the primitive equations of motion (the derivatives of translational and angular momentum plus the kinematic equations) from angular and translational Cartesian coordinates to relative angles. In this thesis the approach was developed from Newtonian and Eulerian principles. Novel single cpu algorithms for inertia matrix and force vector formation have been implemented. Novel multiprocessor algorithms were implemented for the inertia matrix and the force vector on a 2d $\frac{n(n+1)}{2} + n$ triangular mesh architecture. A feedforward systolic matrix solution technique was also implemented. These algorithms are of O(n) complexity, and together they form a parallel formulation which is more efficient than other parallel formulations in the literature for mechanisms with fewer than 15 degrees of freedom. A Caterpillar 215B excavator was simulated in real-time using an array of transputers, and teleoperation experiments were conducted to verify the formulation. Single cpu simulations of the PUMA 600 and a human torso were also conducted.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Peter Lawrence for his encouragement and financial and technical support. My committee members, Dr. M.R. Ito, Dr. F. Sassani and Dr. C.W. da Silva have provided encouragement and guidance. To my coworkers, Nariman Sepehri, Dan Chan, Frank Wan, and Real Frenette I owe a great debt of gratitude for their technical advice and diligent proofreading. A special acknowledgement is owed to Ahmed Ibrahim for pointing me in the right direction. Lastly I would like to thank Serena Mar for her constant encouragement and faith, and my parents for their patience and support.

Chapter 1

Introduction

1.1 Modeling Multibody Dynamics

The dynamics of rigid body motion have been derived and analysed since the 18th century. Indeed, the field has been so well investigated it is now called classical dynamics. Until the early part of this century, dynamicists concentrated on the analytical mechanics of single bodies such as tops, in which the motion could be mathematically described in closed form [Whittaker 27]. Systems and motions of greater complexity were considered too difficult. The development of the computer in the 1950s and 60s, with its ability to numerically integrate, renewed interest in the dynamics of more complex multibody systems for which elegant closed form solutions for the equations of motion could not be obtained.

Multibody computer simulations require the equations of motion in the form of an algorithm which generates the derivative of the system state from the known previous state. The state consists of joint positions and their velocities. The joint variables may be rotational (e.g. Euler angles), or translational (e.g. prismatic). The objective of the computer simulation is to continuously solve for the derivative variables (generally joint accelerations and joint velocities) and numerically integrate over time to determine the new state variables (joint displacements and velocities).

In the fields of space dynamics, mechanism analysis, and vehicle dynamics, the equations of motion have been studied extensively. The formulations have steadily increased in capability, permitting the analysis and simulation of closed chain (kinematic loop) structures, flexible bodies and advanced control techniques. This greater capability has unfortunately also resulted in significantly increased computational complexity (in its crudest sense, the number of multiplies and adds as a function of the size of the problem (in this case the number of degrees of freedom

n)). In addition, the automatic generation of the equations of motion has become more difficult. As a consequence, non-real-time simulation has been the norm, although supercomputers have trimmed computational delays. Online man-in-the-loop interaction, however, has not been possible except for hybrid electromechanical simulators e.g. aircraft training simulators.

1.1.1 Issues

There are four major issues in multibody dynamics [Schwertassek 89] which continue to attract attention.

1. The basic issue is the choice of an approach for deriving the equations. This involves the principle of mechanics, the choice of state variables, and the description of system topology. Conceptual simplicity, efficiency and computational complexity are the main criteria for evaluation.

2. A second issue is the incorporation of other types of dynamic behaviour, including such effects as flexibility and friction. These effects are secondary and dominated by the large scale rigid body rotations, but they can greatly increase the complexity of the equations and may often require much smaller step sizes for the integrator, resulting in greatly increased execution time. These characteristics become more important as advances are made in control theory, structural mechanics, and materials engineering, all of which will require models of light and flexible linkages.

3. The third issue is the choice of appropriate methods for dealing with closed chains. Constraint equations representing closed chain forces can lead to the addition of Lagrange multipliers, or to transformations to minimal sets of equations. The presence of Lagrange multipliers can lead to numerical instability, and thus coordinate partitioning and other reduction techniques have been proposed to stabilise the system and minimise the number of equations [Schwertassek 84]. A further problem arises when the closed chains open and close over time, leading to system representations which vary in form over time. These problems are numerical and theoretical in nature, and can greatly increase the complexity of the formulation.

4. The fourth issue is real-time simulation. The need for shorter design cycles and man-in-the-loop simulation capability requires formulations that execute in real-time. Item one above approaches this through theoretical considerations. Another approach is to use multiple processors to generate and solve the equations. At present this area of research is in its infancy, and the issues of algorithm expandability, architectural complexity, and the integration of various phases in the computation are still research problems. Questions arise as to the most appropriate formulation for multiple processor implementation, and the best computer architecture to use. This thesis addresses these last two questions.

1.1.2 Motivation

The robotics research group at the UBC Electrical Engineering Department is interested in applying telerobotic control techniques to robot linkages such as excavators. A simulator is required when experiments in control, sensing, kinematics, and hand controller strategies need to be conducted in a controlled setting, before major field tests with real systems begin. This thesis is concerned with the design and implementation of a real-time manipulator simulator of the Caterpillar 215B, an excavator-type machine.

1.1.3 Scope

In addition to excavator-type machines, which are basically four link structures, mechanisms with complicated wrists (e.g. the FMS Timberjack feller buncher) and multilegged bodies (e.g. the Kaiser Spyder) have also been designed. Robots such as the proposed Special Purpose Dextrous Manipulator, designed in Canada for the Mobile Service platform on the space station, have multiple chains of links. Modelling these machines efficiently requires formulations of more complex multibody systems than single-branch chains.

One aim of this thesis is to take a step towards the development of an algorithm that accommodates branched (tree-like) systems. Initially only rotational degrees of freedom will be modelled. Prismatic (translational) joints are not included here, but are easily modeled ([Kim 86a] and Chapter 6). The thesis is limited to the dynamics of rigid body systems, since these large links do not flex appreciably. The formulation does not simulate closed chains, but this area of

research will be discussed in the chapter on future work. Control issues will not be considered, but the dynamics of hydraulic actuation will be briefly considered in chapter 5, when the issue of interfacing to actuators and controllers is examined.

The second aim of this thesis is to implement a real-time simulator for the dynamic equations of the Caterpillar 215B. Researchers in robotics have attempted to obtain real-time simulations of common robots by concentrating on computational efficiency and limiting the problem to simple serial systems (a single open kinematic chain), each joint having only one degree of freedom. Often only one phase of the computation is addressed (e.g. inertia matrix, force vector, or matrix solver), without consideration of the integration of these different phases of the formulation. Even these simplified algorithms have required multiprocessors, since they have are too computationally complex for real-time simulation with existing single cpu computers. Our approach is also to use a multiprocessor. The computer architecture, however, will implement a parallel formulation which is concerned with the integration and efficiency of all phases of the simulation. The proposed formulation is evaluated using the computational complexity of the dynamics algorithm and the simplicity and efficiency of the multiprocessor architecture as the criteria.

Initially a single cpu formulation will be developed for branched chains. The parallel architecture, however, is specific for a single chain mechanism. In the discussion on future work, modifications for the inclusion of branched systems in the parallel architecture are examined.

1.2 The Principles of Dynamics

The dynamics of a multibody system are determined by the kinematics, which describe the topology, and the kinetics, which describe the forces. Kinematics is focused on the geometric aspects of motion, excluding the forces involved. Link length and rotational or translational displacement and velocities are calculated to obtain the end-effector (or some link in between) position or velocity in Cartesian coordinates (forward kinematics), or conversely, to determine the joint displacements that produce the endpoint position (inverse kinematics). Kinetics is concerned with describing the balance of forces that influence body motion, such as gravity, actuator moments, and Coriolis forces due to the effects of angular velocity.

Together, the kinematics and kinetics completely describe dynamic behaviour through the differential equations of motion, in terms of joint variables, structural parameters, and the internal and external forces applied to the joints. Examples of the external forces which affect the motion are the joint torques exerted by the actuators, and forces from the environment such as contact forces.

The dynamics can be obtained from two main approaches: Lagrangian, and Newton-Euler. The next two sections outline the fundamentals of these two approaches. Chapter 2 will discuss specific examples in greater detail.

1.2.1 Lagrangian

The Lagrangian approach requires the designer to select a set of independent state variables q which can completely characterize the position and orientation of the system. This set is known as the set of generalized coordinates. For a simple manipulator with rotary or prismatic joints, relative joint angles and displacements are easy to measure, and thus make a good set of coordinates. Another choice is inertially referenced angles and displacements.

Consider the single chain of n bodies in Fig. 1.1, with one degree of freedom per joint. Assume that the generalised coordinates are $q_1, ...q_n$ (measured relative to the previous link, or with respect to an inertial frame). The Lagrangian is defined as the difference between the kinetic (E_k) and potential (E_p) energy of the system.

$$\mathcal{L}(q, \dot{q}) = E_k(q, \dot{q}) - E_p(q). \tag{1.1}$$

The equations of motion are defined as:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = F_i \qquad\qquad i = 1, ...n \tag{1.2}$$

where F_i is the force acting in the direction of the q_i coordinate. The E_k of a link i is given by

$$E_k(i) = \frac{1}{2} m_i \dot{x}_i^T \dot{x}_i + \frac{1}{2} w_i^T J_i w_i. \tag{1.3}$$

J_i is the link inertia tensor, and $\dot{x}_i$ and w_i are the 3×1 translational and angular velocity vectors of the centre of gravity of the ith body, measured relative to an inertial frame of reference. The velocities $\dot{x}_i$ and w_i are the Cartesian velocity coordinates of the ith body. The system vector

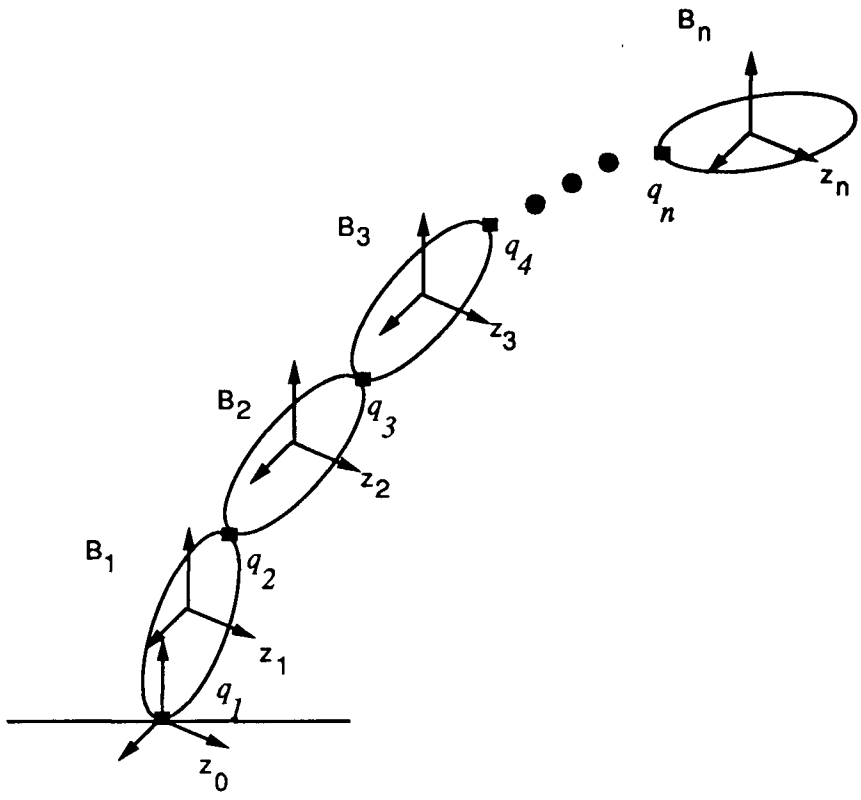

Figure 1.1: Open loop, fixed base, serial chain manipulator

$[\dot{x} \;\; w]$ forms a nonminimum set of coordinates which may be reduced to a minimum set when the hinge constraints are added. Transformations from Cartesian coordinates to q or $\dot{q}$ are necessary because w (a 3×1 vector) is not a minimum representation and is generally not a measurable variable (unless it consists of a rotation about a single axis) since it is a function of up to three Euler-type angular rotation rates.

One generalised set of q is θ_i, rotational displacements (for a system with all rotational joints and no translational joints) which may be measured with respect to a global (inertial) frame or with respect to an adjacent link (relative). The transformation between $[\dot{x}_i \;\; w_i]$ and any other $\dot{q}$ is

$$\begin{aligned} \dot{x}_i &= B_T^{(i)} \dot{q} \\ w_i &= B_R^{(i)} \dot{q} \end{aligned} \tag{1.4}$$

where $B_T^{(i)}$ and $B_R^{(i)}$ are translational and rotational 3×n Jacobian velocity transformations to the system vector $\dot{q}$. The Jacobians for $\dot{q} = \dot{\theta}$ can be derived from the following kinematic velocity and acceleration relations:

$$\begin{aligned} w_i &= w_{i-1} + z_i\dot{\theta}_i \\ \dot{w}_i &= \dot{w}_{i-1} + w_i \times z_i\dot{\theta}_i + z_i\ddot{\theta}_i \qquad (1.5) \\ x_i &= x_{i-1} + l_{i-1} - k_i \\ \dot{x}_i &= \dot{x}_{i-1} + (\dot{w}_{i-1} \times l_{i-1}) - (w_i \times k_i) \\ \ddot{x}_i &= \ddot{x}_{i-1} + \dot{w}_{i-1} \times l_{i-1} + w_{i-1} \times w_{i-1} \times l_{i-1} - \dot{w}_i \times k_i - w_i \times w_i \times k_i \qquad (1.6) \end{aligned}$$

where k_i is the vector from the centre of gravity of a body to the lower hinge, and l_i is a similar vector to the upper hinge on a body (Fig. 1.2). z_i is the unit vector describing the axis of rotation of the ith hinge. k_i, l_i, and z_i are assumed to be constants when referred to their own coordinate frame, but in equations 1.5 and 1.6, they are time varying, as the equations have been implicitly referred to a common coordinate frame.

The system kinetic energy

$$E_k = \sum_{i=1}^{n} E_k(i)$$

can be written as

$$\begin{aligned} E_k &= \frac{1}{2}\dot{q}^T \mathcal{M} \dot{q} \\ \text{where } \mathcal{M} &= \sum_{i=1}^{n} (m_i B_T^{(i)T} B_T^{(i)} + B_R^{(i)T} J_i B_R^{(i)}) \qquad (1.7) \end{aligned}$$

where $\mathcal{M}$ is the $n \times n$ inertia tensor for the complete manipulator system.

The potential energy for the system is due to gravity g, and is

$$E_p = \sum_{i=1}^{n} m_i g^T x_i. \qquad (1.8)$$

When the Lagrangian is differentiated as per equation 1.2, the equation of motion for the ith link is

$$\sum_{j=1}^{n} \mathcal{M}_{ij}\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}\dot{q}_j\dot{q}_k + \sum_{j=1}^{n} m_j g^T B_{Ti}^{(j)} = F_i \qquad i = 1, ...n \qquad (1.9)$$

where F_i is the actuator torque, and $B_{Ti}^{(j)}$ is the ith column of the $B_T^{(j)}$ matrix appearing as a result of the differentiation of E_p with respect to ∂q_i. h_{ijk} represents the terms that occur

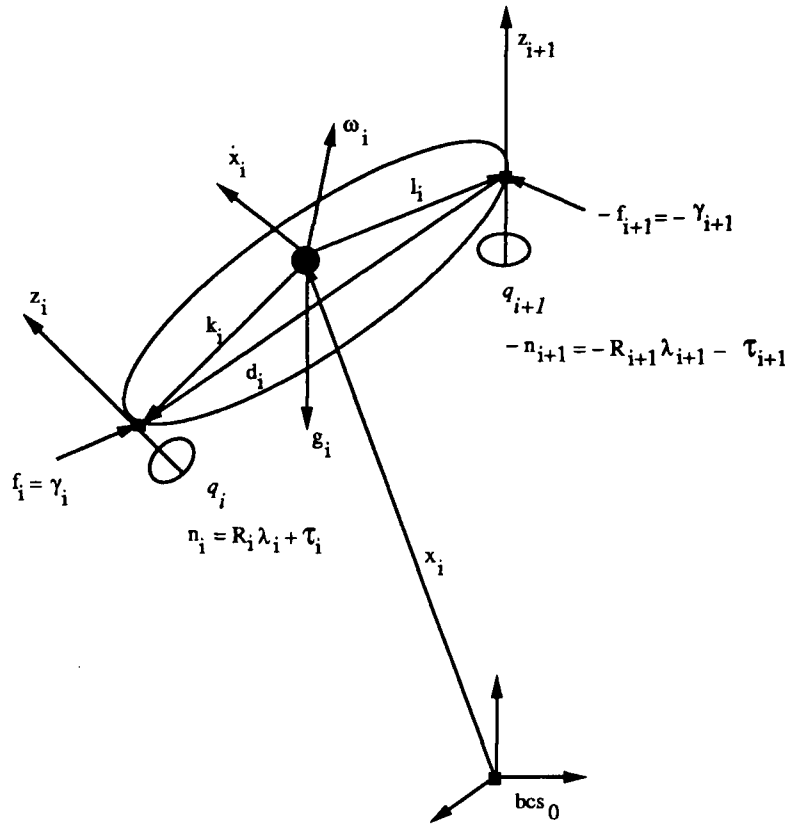

Figure 1.2: Forces and torques on the ith body assuming all rotational joints

as a result of differentiating $\mathcal{M}$ with respect to ∂q. The first term of equation (1.9) represents the moment due to inertia, the second term represents the Coriolis and centrifugal moments, and the third term represents the moment due to gravity vector g. The equation describing the system is then

$$\mathcal{M}(q)\ddot{q} = F - h(q, \dot{q}) - G(B_T). \tag{1.10}$$

The equations must then be solved for $\ddot{q}$ given that $\mathcal{M}$ and B_T are functions of q, and h is a function of q and $\dot{q}$, which are known from the previous time step. F is provided from an actuator algorithm.

The benefit of working with the Lagrangian approach is that the forces of constraint between hinges do not have to be considered. The internal forces never explicitly appear in the equations because a kinetic energy argument is used to describe the formulation, and the kinematics are

used to describe the constraints and transform to generalised coordinates. Although the derivation of the formulation is elegant, developers of computer codes have not used the complete Lagrangian equation for systems of significant complexity due to the computational inefficiency of the technique. Ibrahim [Ibrahim 88] and Silver [Silver 82], however, have shown that this is not necessarily true, by demonstrating that the principle of mechanics does not greatly influence the efficiency of the equations. It is the cost of performing the transformation to different q that determines the efficiency of the algorithms.

1.2.2 Newton-Euler

The Newton-Euler approach uses Newton's Second Law of Motion and Euler's equation for rotational momentum to describe the kinetics of each body. Newton's Second Law,

$$F = \dot{P} = \frac{d}{dt}(m\dot{x}) = m\ddot{x} \tag{1.11}$$

describes the sum of the translational forces F on a body as equal to the rate of change of linear momentum, $\dot{P}$. m and $\dot{x}$ are described as in the previous section. Rotational motion is described by Euler's equation for the inertial torque of a body, which describes the time rate of change of angular momentum:

$$N = \dot{L} = \frac{d}{dt}(Jw) = J\dot{w} + w \times Jw \tag{1.12}$$

where L is the angular momentum, N is its derivative or moment, J is the link inertia (relative to its own coordinate frame), and w is the body's angular velocity (measured with respect to an inertial frame) defined in the previous section. $J\dot{w}$ is known as the angular acceleration torque and $w \times Jw$ is known as the gyroscopic torque. The gyroscopic torque appears because the orientation of J varies with time, producing a non-zero time derivative.

In the Newtonian approach, each link is isolated as a free body, and the forces and torques due to gravity, inertia, velocity, and actuation appear in translational and rotational equations as a balance of forces or moments (Fig. 1.2).

The equation describing the translational forces on a link i is (Fig. 1.2):

$$m_i\ddot{x}_i = \gamma_i - \gamma_{i+1} + m_i g_i \tag{1.13}$$

where the RHS is the sum of the translational forces about link i. γ_i and γ_{i+1} are the forces applied to link i at the hinge points i and $i+1$ by the attached links, which seek to constrain the body's motion. g_i is the acceleration vector due to gravity.

The rotational torques are:

$$\dot{L}_i = J_i\dot{w}_i + w_i \times J_i w_i = R_i\lambda_i - R_{i+1}\lambda_{i+1} + k_i \times \gamma_i \; - l_i \times \gamma_{i+1} + \tau_i - \tau_{i+1} \tag{1.14}$$

where τ_i is the actuator torque occurring about the axis of motion i. $R_i\lambda_i$ (or $R_{i+1}\lambda_{i+1}$) are moments that act to oppose rotational motion in axes orthogonal to the axis of motion of hinge i (or hinge $i+1$). $k_i \times \gamma_i$ and $l_i \times \gamma_{i+1}$ are moments due to the translational hinge forces γ_i and γ_{i+1}. All terms in (1.13) and (1.14) have been referred to a common coordinate frame.

The above equations can be written for all the bodies in a system with the differentiated state variables $\ddot{x}$ and $\dot{w}$. As in the Lagrangian approach, $\ddot{x}$ and $\dot{w}$ can be projected to a different set of states, such as Euler angle accelerations $\ddot{\theta}$ or Euler parameter accelerations $\ddot{q}_p$, and may be relatively or inertially measured. For one degree of freedom relative joint angles, $\dot{w}_i = f(z_1\ddot{\theta}_1, z_2\ddot{\theta}_2, ...z_i\ddot{\theta}_i)$, where z_i is the ith axis of rotation. The equations of motion, 1.13 and 1.14, are then a function of $\ddot{\theta}$, and, due to w, functions of θ and $\dot{\theta}$.

The projection from $[\ddot{x} \quad w]$ to an alternative set of $\ddot{q}$ such as $\ddot{\theta}$ leads to the elimination of the internal constraint forces and moments γ and λ. Once the equations for all bodies are assembled and the constraint forces and moments are eliminated, the equations must then be rearranged to solve for $\ddot{\theta}$. The formulations may be $O(n)$ computational complexity if the projection/elimination of the constraint forces is done recursively from link to link (e.g. the algorithms of Featherstone [Featherstone 83], Rodriguez [Rodriguez 88], Bae and Haug [Bae 87b], Roberson and Schwertassek [Roberson 88], and Brandl [Brandl 86]). If the projection is done as a global transformation, $O(n^2)$ computational complexity (e.g. Hwang and Haug [Hwang 88]), or $O(n^3)$ computational complexity is obtained (e.g. Angeles and Ma [Angeles 88], Kim and Vanderploeg [Kim 86a], and Hemami [Hemami 82]).

1.3 Overview of the Thesis

In the next chapter, Chapter 2, some of the formulations proposed by the research community are reviewed. The approaches taken once the Lagrangian or Newtonian primitive equations have

been established are examined, and examples of parallel implementations are described. In the latter half of Chapter 2, the Newton Euler State Space method [Hemami 82], a formulation based on orthogonal complements, is presented. This method is the starting point for the development of the parallel formulation presented in this thesis in later chapters.

In Chapter 3, the equations of motion are reorganised by referring them to body coordinates. This change illuminates the recursive nature of the equations, determining the structure of the algorithm, which then influences the multiprocessor architecture. The formulation is then generalised to branched systems and the equations are shown to be computable in an automated manner. Simulations of a PUMA 600 manipulator and a branched six degree of freedom human torso are presented to demonstrate the formulation.

In Chapter 4, a multiprocessor algorithm for calculating the inertia matrix for simple serial chains (an open kinematic series of bodies) is obtained. A multiprocessor algorithm for the calculation of the driving force/torque vector is also developed which integrates well with the inertia matrix algorithm. Chapter 4 also describes the implementation of a parallel feedforward systolic matrix equation solver derived by Jainandunsing [Jainandunsing 89]. A parallel systolic architecture decomposes a problem into simple parallel tasks which are identical. These tasks may be run simultaneously on identical cpus, with intermediate results being passed between neighbours in a pipelined manner, analogous to the way blood is pulsed through the body. This solver completes the set of parallel algorithms used for the simulator.

Chapter 5 describes the excavator simulator, including the computer architecture and the interfaces to a graphics computer and joystick control. Measurements of relative efficiency between single and multiple cpu simulations are given, and a teleoperation experiment is presented to demonstrate the usefulness of the simulator.

Chapter 6 concludes the thesis with a discussion on results and future work. Ideas for extending the formulation and the architecture to deal with branched sysytems, multiple degree of freedom joints, and closed chains are presented.

Chapter 2

Existing Formulations of the Equations of Motion

In this chapter some of the formulations proposed by the research community are reviewed. A brief review of the development of multibody dynamics is presented by pointing out the most important characteristics required of a formulation. Following this, some of the most relevant approaches will be discussed, and examples of parallel implementations described. Finally the Newton Euler State Space method [Hemami 82], a formulation based on orthogonal complements, is presented. This method is the starting point for the parallel formulation developed in this thesis.

2.1 A Brief Literature Review

In this section, a very brief review of the literature is presented. Following this, those formulations most relevant to the thesis objectives are examined in more detail.

2.1.1 Coordinate selection

Selecting the principle of mechanics, as discussed in the previous chapter, is just one step in the formulation process. The selection of the final generalised coordinates (e.g. relative or inertial, minimum or nonminimum set) also greatly influences the final form of the equations of motion. Transformations of the initial equations of motion from Cartesian coordinates $[\dot{w} \quad \ddot{x}]$ to other coordinate spaces can reduce the number of equations in the final system to varying degrees, and give the user more insight into the mechanics of the formulation. The final equations may be easier to derive (but require a larger set of equations and coordinates) or harder to derive (but result in a smaller set of equations using a minimum set of coordinates).

In the 1960s, Roberson and Wittenberg [Roberson 66] and Hooker and Margulies [Hooker 66]

developed algorithms for problems in spacecraft dynamics that were implementable on a computer. Newton-Euler principles were used to form the primitive equations for individual bodies, and these primitive Cartesian equations were augmented by constraint equations and Lagrange multipliers, rather than eliminating the constraint forces by projective transformation. Haug [Haug 89] used inertially measured Cartesian generalised coordinates in the DADS software package, in which constraint forces were not eliminated, and the coordinates were not reduced to other state spaces such as relative angles. This choice generated large numbers of equations, but resulted in simpler definitions for constraints and forcing functions.

Hooker [Hooker 70] and Jerkovsky [Jerkovsky 78] used transformations to relative joint variables when it was realised that they were easier to measure, more intuitive, and less prone to numerical error. This choice of coordinates resulted in a smaller but more complex set of equations. Examples of similar transforms are found in [Hemami 82], [Bae 88a], [Kim 86a], [Angeles 88], [Bae 87a], and in the work performed in this thesis.

2.1.2 Reference point

When defining the equations of motion for a large system of bodies, both Roberson and Hooker used the augmented body approach, which defined a body barycenter as the point of application of body forces. The barycenter is the center of gravity of a body which has been augmented by point masses at the hinge positions around the body. These point masses are the cumulative masses of all the links in branches outboard from each hinge. The augmented body approach uses the system center of gravity as the system reference point. He and Goldenberg [He 89] have used this technique to computationally simplify the inverse dynamics (calculating the desired torques) for single chains. The barycentric approach has generally been supplanted in the literature by the direct path method introduced by Ho [Ho 74] and subsequently used by Jerkovsky and others. The direct path approach assumes a designated home body as a reference point, and then describes a body by relating it to home via the chain of bodies between them. This results in simple recursive equations for the kinematics and kinetics in which the point of application is at the hinge or the centre of gravity of a body. In this thesis the direct path method is used, with the home body being the first body in the chain.

2.1.3 Topology

Another requirement of a computer formulation is a systematic method for specifying the system topology. Graph theory, which mathematically expresses the topology as a matrix (or linked list) of connectivity, is a popular way of describing the connections and constraints among links. Roberson and Wittenberg used graph theory to define the topology of a system using an incidence matrix. The incidence matrix can be quite complex if the numbering of the bodies and hinges is arbitrary. The simplest incidence matrix uses a strict numbering system related to the direct path described by Ho [Ho 74]. In this method each body in a branch is labelled from a home body with an ascending integer, and each branch is completely labelled before another branch is begun. As a result, the ith row of the incidence matrix has a 1 in the jth column if the jth body lies on the path between body i and home. Most formulations, including the one developed in this thesis, now use some form of matrix or list to describe the topology (although roboticists have generally limited their simulations to robots without branches and consequently seldom use an incidence matrix).

2.1.4 Closed chains

Closed chains present significant problems for researchers due to the difficulty in selecting appropriate coordinates among a large initial set (the coordinates produced when closed loops are cut). This is because of degrees of freedom which are lost or gained when the system forms geometries which cause the hinges to lock during motion .

Numerous researchers have modeled closed chains, but no method has demonstrated significant advantages over others. The augmentation method, using Lagrange multipliers to represent the unknown closed chain constraint forces, plus constraint equations to represent the kinematic constraints, has been discussed by Nikravesh [Nikravesh 84], Orlandea [Orlandea 77], and Wittenberg [Wittenberg 77]. One method is through direct integration and solution of the augmented equations (a differential-algebraic set). The problem with this technique is that numerical errors appear due to numerical integration inaccuracies, resulting in constraint equation violations. Numerical constraint stabilisation has been used by Baumgarte [Baumgarte 72] and Bayo [Bayo 88] to monitor the amount of violation and then numerically dampen the system to

minimise the errors. State space reduction techniques have also been developed which minimise the number of independent coordinates by partitioning the initial set of coordinates into dependent and independent sets u and v respectively. The constraint equations are then redefined so that functions of u coordinates become functions of v coordinates. Coordinates and velocities u, v, $\dot{u}$ and $\dot{v}$ can be calculated, followed by the solution to the differential-algebraic equations. Separating $\ddot{q}$ into $\ddot{u}$ and $\ddot{v}$ enables the algorithm to control the amount of numerical error. This technique was initially proposed by Wehage, Mani and Haug [Wehage 82] [Mani 84]. Kim and Vanderploeg [Kim 86b] refined it using QR decomposition of the constraint Jacobian to identify the independent coordinates. Another refinement of Mani's technique was made by Park and Haug [Park 86], who combined Baumgarte's numerical stabilisation and Mani's partitioning technique. This thesis does not consider the problem of closed chains, although in Chapter 6 an approach developed by Murray and Lovell [Murray 89] is suggested as a logical extension to the current thesis work.

2.1.5 Other characteristics of multibody systems

Researchers such as Ho, Hughes [Hughes 79], Bodley [Bodley 78], Keat [Keat 83], Kim [Kim 88] and Ibrahim [Ibrahim 88] have developed formulations which include such characteristics as flexible links and more sophisticated joint constraints in the context of space dynamics. Vehicle dynamics researchers such as Fuhrer [Fuhrer 89], and Rulka [Rulka 90] have developed algorithms for simulating railguided vehicles and automobiles, which need to deal with friction and nonlinear wheel-rail forces. Characteristics such as these have not been considered in this thesis work as the emphasis has been on the development of the basic rigid body equations as a first effort.

2.1.6 Parallel formulations

All the formulations described above were developed for single processors. Bae, Kuhl and Haug [Bae 88b] and Hwang, Bae and Haug [Hwang 88] developed and implemented parallel algorithms on an Alliant FX/8 shared memory multiprocessor. Bae's algorithm is an $O(n)$ computational complexity algorithm which uses one cpu for each topological branch (chain) in the system. Hwang's algorithm is an n cpu, $O(n + \log_2 n)$ computational complexity, parallel

version of an $O(n^2)$ single cpu algorithm [Bae 88a]. Robotics researchers such as Lee and Chang [Lee 88], Fijany and Bejczy [Fijany 89] and Amin-Javaheri and Orin [Amin-Javaheri 88] have parallelised single chain linkages using the $O(n^3)$ single cpu algorithm proposed by Orin and Walker [Walker 82] as the basis formulation. They derive $O(n + \lceil \log_2 n \rceil)$ (Lee, Amin-Javaheri) or $O(\log_2 n)$ (Fijany) complexity parallel algorithms that use a generalised cube or a hypercube computer architecture utilising n cpus.

This brief survey, which does not reflect the contributions of many other researchers, shows that a variety of formulations have been derived. The most popular ones use relative joint coordinates as generalised coordinates, and connectivity matrices using the direct path method to describe the topology (and as a consequence, to describe the constraint and kinematic relationships). Modeling techniques for closed chain constraints and secondary characteristics such as flexibility and friction continue to be developed. Research on parallel solutions has only just begun, and will continue to evolve as computer hardware and algorithms change. In this thesis relative joint angles are used as generalised coordinates and a connectivity matrix is used to describe the topology and kinematics. Closed chain constraints are not dealt with, as the emphasis is on real-time performance achieved through parallel processing.

2.2 Forward Dynamics Algorithms in Detail

This section discusses several of the most recent formulations, which are divided into three types, each a different approach to the transformation of the primitive equations of Chapter 1. The first approach is known as the composite rigid body method, which uses the Newton Euler inverse dynamics method derived by Luh, Walker and Paul [Luh 80], plus a technique for calculating the inertia matrix which uses partial sets of links at the open end of the chain as composite bodies. The second approach uses recursive kinematic and kinetic equations to form the dynamics in $O(n)$ time, by locally eliminating the hinge constraint forces. Accelerations and forces are locally projected from body to body across the joints. The third approach uses a global Jacobian velocity transformation to eliminate the hinge forces and project from $[\dot{w}\ \ddot{x}]$ to $\ddot{\theta}$. This third approach is pursued in later chapters of this thesis.

2.2.1 The composite rigid body algorithm

formulations

The most commonly used algorithm for single chained robots is the composite rigid body method of Walker and Orin [Walker 82] (method 3), derived for a single chain manipulator with one degree of freedom joints. The algorithm begins by establishing the Lagrangian form of equation 1.10 as the final form of the desired equations, in which the coordinates are relative joint angles and relative prismatic displacements. The computations required to obtain the elements of the matrices in equation 1.10 are then derived from the Newton-Euler inverse dynamics algorithm derived by Luh. The RHS of equation 1.10 (excluding F) are forces due to Coriolis, centrifugal and gravity effects, and are calculated by artificially setting $\ddot{q}$ to zero in Luh's algorithm.

Initially, the kinematic equations (1.5) and (1.6) are calculated, except that $\ddot{q}_i$ (equivalent to $\ddot{\theta}_i$) is set to zero and x_i is described through a vector p_i, measured from the inertial coordinate frame to the hinge point of body i, rather than the centre of gravity of body i. The equations are written in a forward spatial recursion:

$$\begin{aligned} p_i &= x_{i-1} + l_{i-1} \\ x_i &= p_i - k_i \\ \ddot{x}_i &= \ddot{p}_i - \dot{w}_i \times k_i - w_i \times w_i \times k_i \\ \text{where } \ddot{p}_i &= \ddot{p}_{i-1} - \dot{w}_{i-1} \times d_{i-1} - w_{i-1} \times w_{i-1} \times d_{i-1} \end{aligned} \tag{2.1}$$

and $d_i = k_i - l_i$ is the vector from hinge $i+1$ to hinge i across body i. In this thesis, equation (2.1) has been changed with respect to Luh's original equations to reflect the fact that in the notation used in this thesis, the ith coordinate frame is placed at the bottom hinge of link i, rather than at the top (in Luh's description). The acceleration due to gravity is the first link's acceleration $\ddot{p}_1$. The total force F_i and total moment N_i about a body are then written as in equations (1.11) and (1.12). The hinge force f_i and hinge moment n_i are the force and moment exerted on link i by link $i-1$, and are given in a backward recursion as

$$\begin{aligned} f_i &= F_i + f_{i+1}; \qquad f_n = f_e \\ n_i &= n_{i+1} + N_i - k_i \times F_i - d_i \times f_{i+1}; \qquad n_n = n_e \\ \text{or } N_i &= n_i - n_{i+1} + k_i \times f_i - l_i \times f_{i+1} \end{aligned} \tag{2.2}$$

where the equation for N_i in (2.2) is rearranged to consist of hinge quantites only. f_n and n_n are equal to the external force or moment at the manipulator tip. Equation (2.2) describes the moments and forces about the isolated body. f_i is equivalent to γ_i in equations (1.13) and (1.14), and n_i is the sum of the moments due to $R_i\lambda_i + \tau_i$ (with a single z axis rotation, $R_i\lambda_i$ is about the x and y axes, while τ_i is about the z axis).

The elements of the force vector f_v (Coriolis, centrifugal and gravity forces) are projections of the moments n_i about the hinge axes z_i:

$$f_{v(i)} = n_i^T z_i \qquad\qquad i = 1, ...n \tag{2.3}$$

The inertia matrix element $\mathcal{M}_{ij}$ from equation (1.9) is calculated using the composite mass $\hat{M}_j$, composite centre of mass $\hat{c}_j$, and composite inertia $\hat{J}_j$ of a series of rigid bodies (a $\hat{}$ implies a composite variable). A composite rigid body is comprised of the subset of links j to n at the free end of the chain, where in this case n (no subscript) represents the number of bodies in the chain. The $\hat{M}_j$, $\hat{c}_j$, and $\hat{J}_j$ can be computed recursively in a backward recursion according to the following equations:

$$\hat{M}_j = \hat{M}_{j+1} + m_j, \tag{2.4}$$

$$\hat{c}_j = \frac{1}{\hat{M}_j}(\hat{M}_{j+1}(\hat{c}_{j+1} - d_j) - m_j k_j), \tag{2.5}$$

$$\text{and} \quad \hat{J}_j = \hat{J}_{j+1} + b_j, \qquad i = n, ...1 \tag{2.6}$$

$$\text{with} \quad \hat{M}_n = m_n$$

$$\hat{c}_n = -k_n$$

$$\hat{J}_n = J_n,$$

$$\text{and} \quad b_j = \hat{M}_{j+1}\Big((r_j^T r_j)I - (r_j r_j^T)\Big) + J_j + m_j\Big((r_j^{*T} r_j^*)I - (r_j^* r_j^{*T})\Big) \tag{2.7}$$

$$\text{where} \quad r_j = \hat{c}_{j+1} + d_j - \hat{c}_j \tag{2.8}$$

$$r_j^* = -\hat{c}_j - k_j. \tag{2.9}$$

$\hat{c}_j$ is measured relative to the jth coordinate frame (at hinge j).

The calculation of a column j of $\mathcal{M}$ requires $\ddot{q}$ to be artificially set to zero in the kinematic equations for all links other than j ($\ddot{q}_j = 1$, a unit acceleration). In addition, the joint velocities $\dot{q}$ and gravitational forces are zero. The output of the inverse dynamics algorithm with $\ddot{q}_j = 1$

and these other quantities set to zero results in the jth column of the inertia matrix $\mathcal{M}$. Since joint j is the only joint in motion (links j to n are static), the total forces F_j and N_j on the composite body are

$$\begin{aligned} F_j &= \hat{M}_j(z_j \times \hat{c}_j) \\ N_j &= \hat{J}_j z_j \end{aligned} \tag{2.10}$$

for rotational axes (prismatic joints are considered in Walker's paper but not included here). For $i < j$, F_i and N_i are zero, and f_i and n_i are computed in a backward recursion as

$$\begin{aligned} f_i &= f_{i+1} \\ n_i &= n_{i+1} + d_i \times f_{i+1} \qquad\qquad i = 1, ...j-1; \\ \text{and} \quad f_j &= F_j, \\ n_j &= N_j + \hat{c}_j \times F_j \qquad\qquad \text{when } i = j. \end{aligned} \tag{2.11}$$

For $i > j$, f_i and n_i are zero. An element $\mathcal{M}_{ij}$ in column j is then calculated by projecting the hinge moment n_i onto the ith joint axis. Thus for the jth column, when $\ddot{q}_j = 1$,

$$\mathcal{M}_{ij} = n_i^T z_i \qquad\qquad i \leq j \tag{2.12}$$

where this value of n_i must be recalculated for each value of j (each new column of $\mathcal{M}$). $\mathcal{M}_{ij}$ for a translational joint i can be calculated by projecting the hinge force f_i onto z_i. For Walker's algorithm, the computational complexity of forming the inertia matrix is $O(n^2)$, of forming the RHS is $O(n)$, and of solving for $\mathcal{M}\ddot{q} = RHS$ using Cholesky factorisation and backsubstitution is $O(n^3)$ (see Table 2.1).

Algorithm	×	+
Walker [1982]	$\frac{n^3}{6} + \frac{27n^2}{2} + \frac{577n}{3} - 49$	$\frac{n^3}{6} + 8n^2 + \frac{1001n}{6} - 64$
Angeles [1988],[1989]	$\frac{7n^3}{6} + 21n^2 + \frac{539n}{6} - 91$	$n^3 + 16n^2 + 75n - 90$
Brandl [1986]	$250n - 222$	$220n - 198$
Featherstone [1983] (no force)	$199n - 198$	$174n - 173$
Bae [1988a] (no inversion)	$3n^2 + 186n - 36$	$\frac{5n^2}{2} + \frac{411n}{2} - 78$

Table 2.1: Computational Complexity for single chain systems on one cpu

parallel implementations

There have been three parallel implementations of Walker and Orin's forward dynamics algorithm. Lee and Chang [Lee 88] reorganised Walker and Orin's algorithm for an n cpu, single-instruction multiple-data (SIMD) multiprocessor by dividing the computation into three parallelisable tasks - formation of the inertia matrix, formation of the force vector, and solution of the matrix equation. The first two tasks used n cpus, while the matrix solver may use an $\frac{n(n+1)}{2}$ mesh systolic algorithm or an n cpu set-ordering technique.

The computation of the force vector is carried out using n cpus connected in a generalised cube network [Lee 88]. The 3×3 rotation matrices A_i^0 which refer a body coordinate system (*bcs*) variable to the inertial frame are calculated by multiplying a cascaded sequence $[A_1^0 ... A_{i-1}^{i-2} A_i^{i-1}]$. Using the generalised cube network, the A_i^{i-1} $i = 1, ...n$ can be calculated all at once on all n cpus, followed by $O(\lceil \log_2 n \rceil)$ time computation of A_i^0, through the use of the recursive doubling algorithm (RDA). The RDA algorithm pairs the computations of the rotation matrices in each of $\log_2 n$ levels e.g. $A_1^0 A_2^1$, $A_3^2 A_4^3$, etc in level 1; followed by $A_2^0 A_4^2$ and $A_2^1 A_4^2$ in level 2 etc . The inertia J_i, axis z_i, and vectors d_i and k_i, all in body coordinates, are then referred to the inertial frame using the A_i^0. This is done in constant time using all n cpus. Recursive kinematic equations 1.5, 1.6 and 2.1, and hinge force and moment equations 2.2 can be computed in $O(\lceil \log_2 n \rceil)$ time using the RDA algorithm once they have been referred to the inertial frame. The force vector elements, projections of the hinge moments n_i onto the rotation axes z_i, are calculated in constant O(1) time using all n cpus. The overall complexity for the force vector is thus $O(\lceil \log_2 n \rceil)$. The inertia matrix is obtained by first calculating $\hat{M}_j$, $\hat{c}_j$, and $\hat{J}_j$ in $O(\lceil \log_2 n \rceil)$ computations using the same computer architecture and RDA algorithm as for the force vector calculation. The f_i and n_i $(i < j)$ (equation 2.11), for all j, are obtained in $O(n)$ time, and $\mathcal{M}_{ij}$ $(i < j)$, the axis projections of n_i for a specific j, are obtained in O(1) constant time. The matrix is solved in $O(n^2)$ time with n cpus using a set-ordering technique, and in $O(n)$ time when an $\frac{n(n+1)}{2}$ 2d grid and a parallel systolic Cholesky factorisation matrix solving algorithm is used. The total complexity for all three dynamics tasks is thus $O(n^2 + n + \lceil \log_2 n \rceil)$ for n cpus and $O(n + \lceil \log_2 n \rceil)$ for $n + \frac{n(n+1)}{2}$ cpus (exact complexity polynomials for the parallel algorithms given in this chapter are presented in Chapter 4).

Amin-Javaheri and Orin [Amin-Javaheri 88] developed a similar algorithm to Lee's, but for the

inertia matrix alone. As in Lee's work, the algorithm was developed for both n and $\frac{n(n+1)}{2}$ cpus, and could be computed in $O(n + \lceil \log_2 n \rceil)$ time for an n cpu hypercube architecture, $O(n)$ time for a pipelined n cpu architecture, and $O(n)$ time for an $\frac{n(n+1)}{2}$ cpu mesh architecture. Different parallel interpretations of Walker's algorithm were developed in which the calculation of $\mathcal{M}_{ij}$ was calculated by row or diagonal rather than by column. The most efficient parallel version was the column algorithm, implemented on both the pipeline ($O(n)$ time) and the hypercube ($O(n + \lceil \log_2 n \rceil)$ time). The hypercube column version was equivalent to Lee and Chang's. They also developed a VLSI processor on which the algorithms could be implemented.

Fijany and Bejczy [Fijany 89] developed an algorithm based on the Composite Rigid Body Spatial Inertia algorithm. This algorithm is very similar to Walker's, except that the $\hat{c}_j$, r_j^* and r_j equations in Walker's algorithm are replaced by equations for computing the first and second moments of the mass, from which n_i and f_i are easily derived. All equations are referred to the inertial frame. Computation of $\hat{M}_j$, A_j^0, and first and second moment of mass is accomplished in $O(\lceil \log_2 n \rceil)$ time with n cpus. The f_i, the n_i, the diagonal inertia matrix elements $\mathcal{M}_{ii}$, and the off-diagonal elements $\mathcal{M}_{ij}$ are calculated in constant time using $\frac{n(n+1)}{2}$ cpus. Several architectures and algorithms were presented by Fijany, but the most efficient used $\frac{n(n+1)}{2}$ cpus in a 2 dimensional mesh to calculate the inertia matrix in $O(\lceil \log_2 n \rceil)$ time.

2.2.2 Recursive O(n) techniques

formulations

The second approach calculates the acceleration vector in $O(n)$ time by generating the explicit form of the dynamic equations ie. $\ddot{\theta}_i$ is obtained without matrix inversion. Vereschagin [Vereschagin 74] devised the first $O(n)$ algorithm by observing that hinge constraint equations could be generated recursively and combined with other equations to generate the accelerations. The formulation applied to single chain open systems with prismatic and revolute joints. Armstrong [Armstrong 79] independently developed a similar but less efficient formulation for systems with spherical joints, which was additionally capable of modeling flexible links.

Featherstone [Featherstone 83] independently redeveloped Vereschagin's method using different initial principles, and introduced the concepts of articulated-body inertias $\hat{J}_i^A$ and bias forces

$\hat{P}_i$. $\hat{J}_i^A$ and $\hat{P}_i$ are the inertia and force which relate the applied spatial force $\hat{f}_i = [f_i\ n_i]$ to the inertially measured absolute Cartesian acceleration $\hat{a}_i = [\ddot{x}_i\ \dot{w}_i]$

$$\hat{f}_i = \hat{J}_i^A \hat{a}_i + \hat{P}_i \tag{2.13}$$

Initially, kinematic equations similar to 1.5 and 1.6 are calculated in a forward recursion to obtain values for x_i and $\hat{s}_i$ ($\hat{s}_i$ is a 6×1 vector describing the ith hinge axis vector z_i and the vector defined by $(x_i + k_i) \times z_i$, referred to the inertial frame). Then, articulated-body inertias were defined in a backward recursion by Featherstone as

$$\begin{aligned} \hat{J}_i^A &= \hat{J}_i + \hat{J}_{i+1}^A - \frac{\hat{J}_{i+1}^A \hat{s}_{i+1} \hat{s}_{i+1}^T \hat{J}_{i+1}^A}{\hat{s}_{i+1}^T \hat{J}_{i+1}^A \hat{s}_{i+1}} \qquad i = n, \ldots 1 \\ \text{where } \hat{J}_n^A &= \hat{J}_n \end{aligned} \tag{2.14}$$

and

$$\hat{J}_i = \begin{bmatrix} m_i \tilde{x}_i & m_i I \\ J_i - m_i (\tilde{x}_i)^2 & -m_i \tilde{x}_i \end{bmatrix}. \tag{2.15}$$

Here $\hat{J}_i$ is defined as the spatial inertia matrix relative to the inertial frame (and is not related to the composite body inertia), and $\tilde{x}_i$ is the skew symmetric matrix of vector x_i measured from the inertial frame to the centre of gravity of body i. All quantities are referred to the inertial frame. The third term in 2.14 locally projects the articulated-body inertia $\hat{J}_{i+1}^A$ about the $(i+1)$th axis $\hat{s}_{i+1}$. The bias force $\hat{P}_i$ is defined as

$$\begin{aligned} \hat{P}_i &= \hat{P}_{i+1} + \frac{\hat{J}_{i+1}^A \hat{s}_{i+1} (Q_{i+1} - \hat{s}_{i+1}^T \hat{P}_{i+1})}{\hat{s}_{i+1}^T \hat{J}_{i+1}^A \hat{s}_{i+1}} \qquad i = n, \ldots 1 \\ \text{where } \hat{P}_n &= 0 \end{aligned} \tag{2.16}$$

Q_{i+1} is the $(i+1)$th scalar force vector element, consisting of the actuator torque or force, plus the Coriolis, centrifugal, and gravity effects. $\hat{a}_i$ and relative joint accelerations $\ddot{\theta}_i$ can be found in a forward recursion

$$\hat{a}_i = \hat{a}_{i-1} + \hat{s}_i \ddot{\theta}_i \tag{2.17}$$

$$\begin{aligned} \hat{a}_0 &= 0 \\ \ddot{\theta}_i &= \frac{Q_i - \hat{s}_i^T \hat{J}_i^A \hat{a}_{i-1} - \hat{s}_i^T \hat{P}_i}{\hat{s}_{i+1}^T \hat{J}_{i+1}^A \hat{s}_{i+1}} \qquad i = 1, \ldots n \end{aligned} \tag{2.18}$$

The complexity is shown in Table 2.1. Since Featherstone's algorithm appeared in the robotics literature, multibody dynamics researchers have applied these concepts to develop their own formulations from other basic principles. Bae and Haug [Bae 87a] used variational-vector calculus as the basis for transforming Cartesian equations to relative coordinates and for defining the equations of motion. Brandl, Johanni and Otter [Brandl 86], using Newton-Euler principles, introduced modes of motion and modes of constraint to describe complex hinges. They also integrated the calculation of the force vector into their algorithm, whereas Featherstone assumed that the force vector Q would be calculated by an inverse dynamics algorithm such as Luh's. Rodriguez [Rodriguez 88] derived a linear operator technique, based on recursive estimation and Kalman filtering theory, to develop an $O(n)$ algorithm similar to Featherstone's. Rodriguez, like Featherstone, assumed the calculation of the force vector would be done by an inverse dynamics algorithm. Brandl's algorithm is the most efficient of all formulations for chains with greater than 6 degrees of freedom, regardless of approach ($O(n)$ or otherwise) when simulated on a single cpu [Brandl 86] (Table 2.1), and is thus the most efficient algorithm for simulating redundant ($n > 6$) robots with one cpu.

parallel implementations

Bae and Haug [Bae 87a] and Rodriguez [Rodriguez 88] have shown that their $O(n)$ algorithms can be parallelised, but only one cpu can be used for each branch in the topology of the system. Fijany [Fijany 89] and Lee [Lee 88] have stated that within a single chain, parallelism cannot be achieved at the equation level using $O(n)$ algorithms. Since most robots are single chains, parallel versions of $O(n)$ algorithms are not useful unless more complicated branched robots need to be modelled (there is of course an abundance of branched systems outside the robotics field).

2.2.3 Orthogonal complement algorithms

formulations

The third approach, which is the most relevant to this thesis, is the orthogonal complement method. Orthogonal complement algorithms use a global Jacobian transformation matrix which

projects the $[\dot{w}\ \ddot{x}]$ derivative state space to a reduced (smaller) independent set of differentiated state variables e.g. relative acceleration angles $\ddot{\theta}$. The transformation matrix contains vectors which are tangential to the constraint manifold of the system i.e. the vectors are orthogonal to the coefficients orienting the hinge constraint forces γ and λ. As a result, the transformation matrix eliminates the hinge forces from the primitive equations.

One example of this transformation matrix approach is the Newton Euler State Space formulation developed by Hemami [Hemami 82]. This $O(n^3)$ algorithm uses a two step transformation to eliminate the constraint forces (the translational hinge forces, and the rotational hinge constraint torques which are orthogonal to the axes of rotation) from the Newtonian equations of motion. The first step removes the translational acceleration coordinate $\ddot{x}$ from the principal equations (1.13) and (1.14) by using equation (1.6) (repeated here in (2.19)),

$$\ddot{x}_i = \ddot{x}_{i-1} + \dot{w}_{i-1} \times l_{i-1} + w_{i-1} \times w_{i-1} \times l_{i-1} - \dot{w}_i \times k_i - w_i \times w_i \times k_i \qquad (2.19)$$

to derive a matrix which projects the system from $\ddot{x}$ to the inertially measured angular acceleration state $\dot{w}$ and simultaneously eliminates the translational hinge forces γ. Following this, a second step projects the system from $\dot{w}$ to either inertially measured [Hemami 82] or relatively measured [Buchner 86] Euler angle accelerations $\ddot{\theta}$. This step utilises the angular velocity Jacobian from equation (1.4), which is a symbolic version of equation (1.5) (repeated here in (2.20)):

$$\dot{w}_i = \dot{w}_{i-1} + w_i \times z_i\dot{\theta}_i + z_i\ddot{\theta}_i \qquad (2.20)$$

This reduced derivative state space $\ddot{\theta}$ contains only the rotational degrees of freedom (assuming only rotational joints in the system), which are orthogonal to the rotational hinge constraint forces λ of equation (1.14). As a result of this orthogonality, the λ are eliminated by the projection and the equations appear in a final form similar to that derived by the Lagrangian approach. Note that at this point, translational quantities in (1.13) are referred to and measured in inertial (base) coordinates, while rotational quantities in (1.14) are measured and referred to relative coordinates.

Another example of an orthogonal transformation was formulated by Bae, Hwang and Haug [Bae 88a] [Hwang 88], who developed a recursive orthogonal complement formulation for closed chain systems based on variational vector calculus. Instead of defining a matrix which globally

(i.e. system-wide) projects the inertially measured and referred Cartesian derivative state $[\ddot{x}\ \dot{w}]$ to relative coordinates, they defined a local transformation which acts upon every element in a system 'accumulated inertia' matrix. A 6×6 link 'inertia' K_i was defined as an accumulation (i to n) of link masses and rotational link inertias measured with respect to the inertial frame.

$$\begin{aligned} K_i &= \hat{M}_i + K_{i+1} \qquad i = n, \ldots 1 \\ K_n &= \hat{M}_n \end{aligned} \tag{2.21}$$

$$\text{where } \hat{M}_i = \begin{bmatrix} m_i I & -m_i \tilde{x}_i \\ m_i \tilde{x}_i & J_i - m_i (\tilde{x}_i)^2 \end{bmatrix}. \tag{2.22}$$

I is a 3×3 unit matrix, and $\hat{M}_i$ is a rearranged version of Featherstone's spatial inertia matrix $\hat{J}_i$. K_i can only be accumulated if x_i and J_i are referred to the inertial frame. For a simple open chain in which all joints are rotational and each has a single degree of freedom, each element of the system inertia matrix $\mathcal{M}_{ij}$ is $B_i^T K_i B_j$, where

$$B_i = \begin{bmatrix} (x_i + k_i) \times z_i \\ z_i \end{bmatrix} \tag{2.23}$$

The vector $(x_i + k_i)$ describes the position of the ith hinge measured relative to and referred to the inertial frame. B_i locally projects the accumulated 'inertia' about the rotational axes of body i. The computational complexity of forming the inertia matrix plus the force vector is given in Table 2.1. The complexity of the matrix solver is excluded as it was not given. Note that all quantities must be referred to the inertial frame.

Kim and Vanderploeg [Kim 86a] developed a formulation for closed chains using the Jacobian velocity transformation to relative joint angles proposed by Jerkovsky [Jerkovsky 78]. Although the Lagrangian kinetic energy expression is used as the principle for deriving the inertia matrix, rather than the Newtonian and Eulerian principles, the final transformation is the same as Hemami's. Kim and Vanderploeg also modeled closed chains using Lagrange multipliers and constraint Jacobians. Ibrahim [Ibrahim 88] developed a similar formulation to Kim's using the kinetic energy expression and a projection matrix. This technique was developed within a formulation which included flexible linkages and variable mass deployment.

Angeles and Ma [Angeles 88] presented a formulation for a serial chain manipulator using Kane's approach to produce a natural orthogonal complement to the kinematic constraint equations.

This natural orthogonal complement is combined with the Lagrangian kinetic energy expression to derive the formulation. The natural orthogonal complement matrix is the same as both Hemami's and Kim's final transformations, although an interesting feature of the algorithm is the referral of each body's variables to its own coordinates. This feature is exploited in the algorithm developed here in Chapter 3 to derive a similar algorithm to Angeles', but based on Hemami's method.

Lilly and Orin have recently developed two algorithms which are related to the orthogonal complement concept [Lilly 91]. Lilly/Orin I is similar to those algorithms described above, using a recursive definition of the velocity Jacobian for chains of successively longer length to identify the system inertia matrix. The algorithm is $O(n^3)$ complexity. Lilly/Orin III is similar to Bae, Hwang and Haug's algorithm [Bae 88a] [Hwang 88]. The variables, however, are referred to body coordinates. This algorithm is $O(n^2)$ complexity.

parallel implementations

Hwang, Bae and Haug [Hwang 88] parallelised their algorithm for an n processor shared bus computer architecture. Generally speaking, Hwang's paper does not lend itself to a computational complexity analysis, as closed chain constraint equations are included (the complexity of single chain branches becomes only a small part of the problem). Our estimates of Hwang's algorithm for a single chain (see Chapter 4) indicate that for the inertia matrix, $O(\lceil \log_2 n \rceil)$ time is taken for the computation of the rotational matrices A_i^0 (using the RDA algorithm proposed by Lee), constant time is taken to refer the link inertias J_i to the inertial frame, $O(\lceil \log_2 n \rceil)$ for calculating x_i (referred to the inertial frame) and accumulating K_i, and $O(n)$ time is taken to locally project each accumulated 'inertia' K_i to relative coordinates (i.e. $B_i^T K_i B_j$). The overall complexity for forming the inertia matrix is thus $O(n + \lceil \log_2 n \rceil)$ using n cpus arranged in a hypercube. The architecture for the matrix solver, which is used in the following phase, is best suited to a nearest-neighbour 2d systolic mesh. As a result, both the hypercube and mesh architectures are theoretically required for implementing the simulator (it must be noted that Hwang chose to implement the algorithm on a shared bus, and consequently our estimate is for an ideal architecture). Lilly/Orin III has not been parallelised at this time.

The forward dynamics algorithms proposed by Hemami, Kim and Vanderploeg, Ibrahim, and

Angeles are all relatively similar. They have not yet been implemented on a parallel architecture, although Hemami and Zheng [Zheng 86] have developed an architecture for inverse dynamics using inertially measured coordinates. One of these algorithms would be a logical reference point for the development of an alternative parallel formulation.

2.2.4 The Newton Euler State Space orthogonal complement algorithm

In this thesis the Newton Euler State Space formulation has been chosen as a possible algorithm for parallelisation. In the remainder of this chapter the equations of motion described by Hemami are developed using relative rather than inertially measured coordinates. In Chapter 3 these equations will be made more efficient by referring each body's equations to its own coordinate system.

Consider the n link open chain linkage depicted in Figure 1.1. Each body is numbered B_1 through B_n, with a home body labelled 1, inertial frame labelled 0 (coincident with joint 1), and all other bodies labelled with increasing integer value as the distance along the direct path between an arbitrary body and home increases. The hinge on body i nearest to the home body and along the direct path is labelled joint i, about which the coordinate q_i rotates. Each of the n joints has one degree of rotational freedom and is driven by an ideal rotational torque actuator.

A body coordinate system (*bcs*) is attached to the centre of gravity of each body. In Hemami's formulation, the *bcs* was aligned with the principal axes of the body. As a result, the body's inertia matrix J, referred to it's own coordinates, is diag$[J_{xx}, J_{yy}, J_{zz}\ ]$. This alignment is restrictive since the representation of axes of rotation not aligned with the principal axes is difficult. In this thesis, it has been found that a Denavit-Hartenberg-type axis orientation is more suitable, especially for single degree of freedom (dof) joints. In such a coordinate description (Fig. 2.1) the axis of rotation at joint i is assigned to be the z_i axis of bcs_i and the x_i axis is the common perpendicular between the z_{i-1} and z_i axes, pointing in direction from z_{i-1} to z_i. Although the coordinate origin is drawn at the hinge point in Fig. 2.1, it is actually positioned at the centre of gravity of each body in this algorithm since most of the body quantities are defined relative to the centre of gravity of each body. It is drawn at the hinge since it is easy to graphically identify the direction of the axis of motion z_i and the x_i

axis when the coordinate frame is shown at the hinge.

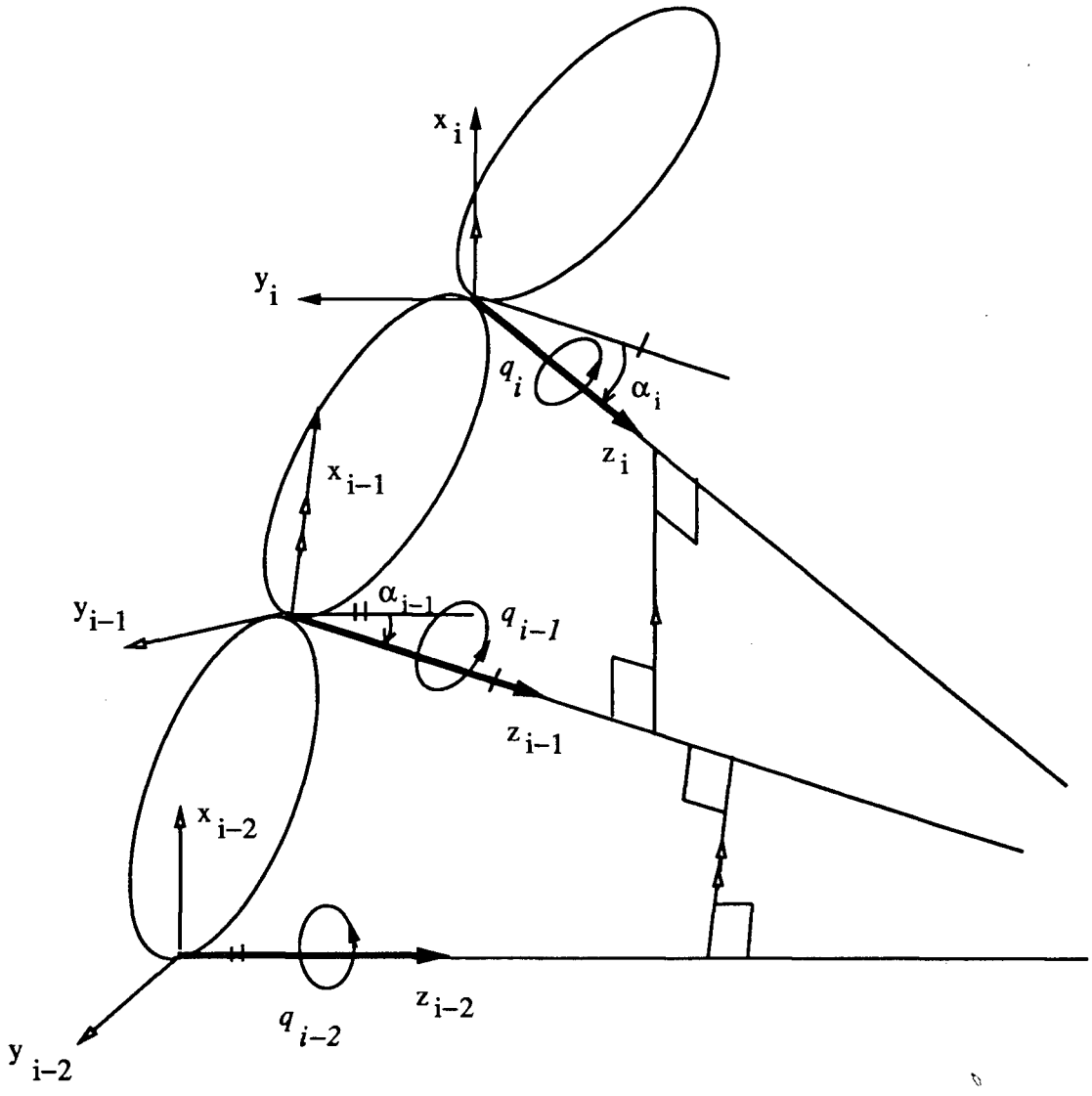

Figure 2.1: Body coordinate frames

For each body, the Newtonian forces and torques acting on the free body are shown as in Fig. 1.2. $\gamma^0_{i,0}$ represents a vector of translational hinge constraint forces measured in and referred to inertial coordinates; $\lambda^i_{i,i}$ represents rotational constraint torques at joint i, measured in and referred to bcs_i coordinates, which act in the axes of motion (x_i and y_i) other than the actual

rotational degree of freedom (z_i); $\tau^i_{i,i}$ represents ideal actuator torques at joint i defined in and referred to bcs_i, rotating about axis z_i. Newtonian equations of motion for body i are:

$$\begin{aligned} J^i_i \dot{w}^i_{i,0} &= f^i_{i,0} + \tilde{k}^i_{i,i} A^i_0 \gamma^0_{i,0} - \tilde{l}^i_{i,i+1} A^i_0 \gamma^0_{i+1,0} + R^i_{i,i} \lambda^i_{i,i} - A^i_{i+1} R^{i+1}_{i+1,i+1} \lambda^{i+1}_{i+1,i+1} \\ &\quad + \tau^i_{i,i} - A^i_{i+1} \tau^{i+1}_{i+1,i+1} \qquad (2.24) \\ m_i \ddot{x}^0_{i,0} &= m_i g + \gamma^0_{i,0} - \gamma^0_{i+1,0} \qquad (2.25) \end{aligned}$$

Equation 2.24 describes the torques applied to the ith body in bcs_i coordinates, including rotational constraint torques, actuator torques, and moments due to the translational hinge constraint forces. (It should be noted that vectors appear in lower case, and matrices appear in upper case, or have a ˜ over the variable to indicate that a skew-symmetric matrix has been created from a vector. The superscript on each variable indicates the *bcs* to which the variable has been referred. The subscripts may have a number of meanings depending on the type of variable).

Parameters not already defined are:

1. $\dot{w}^i_{i,0}$ is the angular acceleration of body i (first subscript) measured with respect to the inertial frame 0 (second subscript) and referred to frame i (superscript).
2. $f^i_{i,0}$ is the gyroscopic torque $-(w^i_{i,0} \times J^i_i w^i_{i,0})$ [Hemami 82].
3. $R^i_{i,i}$ describes those axes (measured with respect to, and referred to, ith coordinates) in which the i^{th} body cannot rotate (with one dof (z axis) per joint, $R^i_{i,i}$ is a 3×2 matrix of two unit vectors in the x and y axes).
4. A^i_{i-1} is the rotation matrix referring a vector in $i-1$ to ith coordinates.
5. $\tilde{k}^i_{i,i}$ is the skew symmetric matrix of vector $k^i_{i,i}$, measured from the centre of gravity of body i to the ith hinge (inboard (see Fig. 1.2)) and referred to the ith coordinate frame.
6. $\tilde{l}^i_{i,i+1}$ is the skew symmetric matrix of vector $l^i_{i,i+1}$, measured from the centre of gravity of body i (first subscript) to the outboard hinge $i+1$ on body i (second subscript) and referred to the ith coordinate frame.
7. J^i_i is the inertia matrix of a body measured from its centre of gravity (first subscript) and referred to its own coordinates (superscript). Because it is not aligned with the principal axes, J^i_i is not necessarily diagonal.

Equation (2.25) describes the translational forces affecting the centre of gravity (cog) of body i in inertially measured and referred coordinates. $m_i g$ is the force due to gravity, where g is $[0\ 0\ -9.81]^T$. x_i is the length vector from the inertial coordinate origin to the cog of body i.

Given the above equations for n bodies, a matrix equation can be developed to represent the entire system:

$$\begin{aligned} J\dot{w} &= f + H_1\gamma + N_1\lambda + E\tau \\ M\ddot{x} &= g + H_2\gamma \end{aligned}$$

$$\begin{bmatrix} J & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} \dot{w} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} \gamma + \begin{bmatrix} N_1 \\ 0 \end{bmatrix} \lambda + \begin{bmatrix} E \\ 0 \end{bmatrix} \tau \tag{2.26}$$

where H_1 and H_2 are coefficient matrices for the linear constraint forces; N_1 is the coefficient for the rotational constraint torques; and E relates the actuator torques to the appropriate coordinate frames. f and g are vectors describing the system forces generated by the gyroscopic torques and gravity respectively, and J and M are block diagonal matrices $J = [J_1^1, J_2^2, ...J_n^n]$ and $M = [m_1 I, m_2 I, ...m_N I]$. An example of these coefficient matrices for a five body chain is presented in Appendix A.

The internal constraint forces γ and λ may be eliminated by multiplying (2.26) by matrices which are orthogonal complements of $[H_1^T\ H_2^T]^T$ and N_1, respectively. First, $\ddot{x}$ is redefined as a function of $\dot{w}$ using translational kinematic hinge constraint equations. An equation for the ith hinge constraint is illustrated in Fig. 2.2. The constraint and its second derivative, referred to inertial coordinates, are:

$$x_{i,0}^0 + A_i^0 k_{i,i}^i - x_{i-1,0}^0 - A_{i-1}^0 l_{i-1,i}^{i-1} = 0 \tag{2.27}$$

$$\ddot{x}_{i,0}^0 - \ddot{x}_{i-1,0}^0 + A_i^0 (\tilde{w}_{i,0}^i)^2 k_{i,i}^i - A_{i-1}^0 (\tilde{w}_{i-1,0}^{i-1})^2 l_{i-1,i}^{i-1} + A_i^0 \tilde{k}_{i,i}^i \dot{w}_{i,0}^i - A_{i-1}^0 \tilde{l}_{i-1,i}^{i-1} \dot{w}_{i-1,0}^{i-1} = 0 \tag{2.28}$$

Equation (2.27) describes the paths through bodies i and $i-1$ for getting to the ith joint and (2.28) is (2.27) twice differentiated ($\tilde{w}$ is the skew symmetric matrix derived from w). The differentiated hinge constraint equation for the whole system is:

$$U\ddot{x} = Q\dot{w} + \dot{Q}w \tag{2.29}$$

$$\ddot{x} = U^{-1}Q\dot{w} + U^{-1}\dot{Q}w \tag{2.30}$$

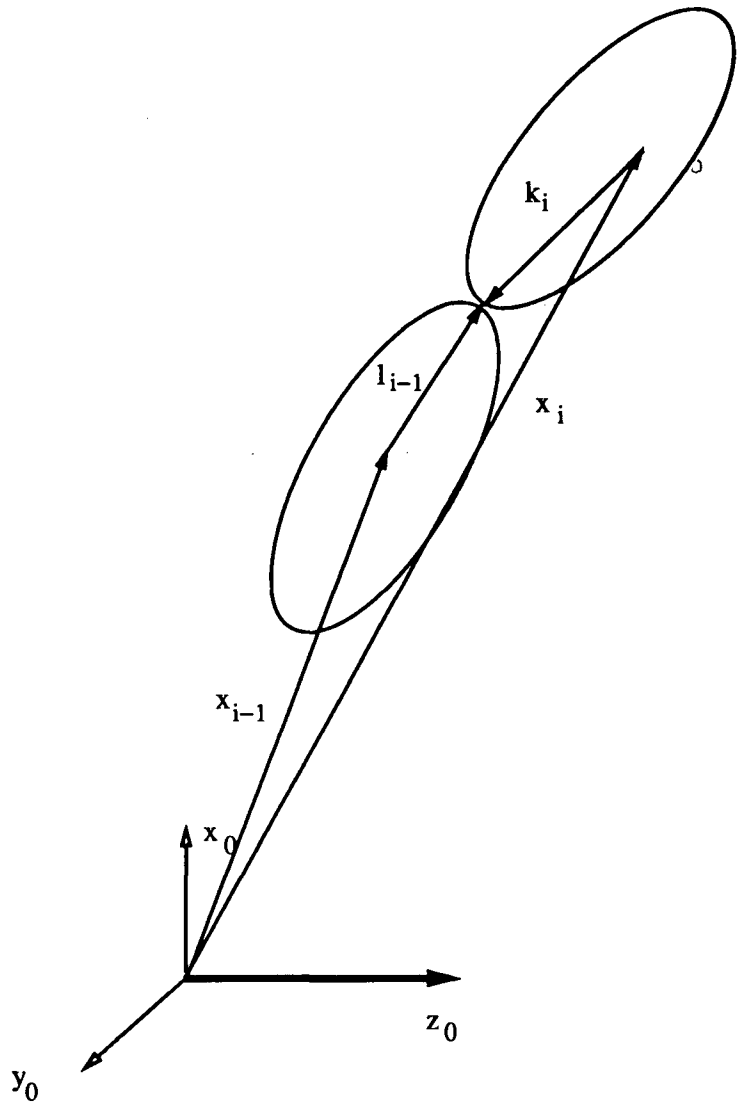

Figure 2.2: Vectors describing the ith hinge constraint

Appendix A describes the internal structure of the matrices $U, Q, \dot{Q}w$, $U^{-1}Q$ and U^{-1}.

$\ddot{x}$ can be eliminated from (2.26) by substituting (2.30), resulting in:

$$\begin{bmatrix} J & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} I \\ U^{-1}Q \end{bmatrix} \dot{w} = \begin{bmatrix} f \\ g \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} \gamma + \begin{bmatrix} N_1 \\ 0 \end{bmatrix} \lambda + \begin{bmatrix} E \\ 0 \end{bmatrix} \tau - \begin{bmatrix} 0 \\ MU^{-1}\dot{Q}w \end{bmatrix} \quad (2.31)$$

Hemami describes $[I^T \ (U^{-1}Q)^T]^T$ as the orthogonal complement to $[H_1^T \ H_2^T]^T$, with the rows of $[I^T \ (U^{-1}Q)^T]^T$ spanning the tangent space of the kinematic constraint surface. The columns of $[H_1^T \ H_2^T]^T$, which are normal to this constraint surface, are orthogonal to the tangent space. Premultiplying (2.31) by $[I^T \ (U^{-1}Q)^T]$ thus eliminates γ, since it can be shown that $H_2 = U^T$ and $H_1 = -Q^T$, and so the result is zero.

A second transformation can be used to eliminate λ and project the system to relative joint angle acceleration space $\ddot{\theta}$. Kinematic equation 1.5 is used to project $w_{i,0}^i$ to relative joint angle velocities. Equation 1.5 can be written as

$$w_{i,0}^i = A_{i-1}^i w_{i-1,0}^{i-1} + w_{i,i-1}^i \tag{2.32}$$

where $w_{i,i-1}^i$ is the angular velocity relative to the previous body, but referred to its own coordinates. $w_{i,i-1}^i$ is related through kinematic differential equations to the joint angle rates $\dot{\theta}_i$ using the rotational sequence Body-3 xyz [Hughes 86]:

$$\begin{bmatrix} w_{ix} \\ w_{iy} \\ w_{iz} \end{bmatrix} = \begin{bmatrix} \cos\theta_{iy}\cos\theta_{iz} & \sin\theta_{iz} & 0 \\ -\cos\theta_{iy}\sin\theta_{iz} & \cos\theta_{iz} & 0 \\ \sin\theta_{iy} & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\theta}_{ix} \\ \dot{\theta}_{iy} \\ \dot{\theta}_{iz} \end{bmatrix} \tag{2.33}$$

When limited to a single dof in the z axis, $\dot{\theta}_{ix}$ and $\dot{\theta}_{iy}$ are zero, as $\theta_{ix} = \alpha_i$ (the twist angle constant) and $\theta_{iy} = 0$. Thus if $\theta_i = \theta_{iz}$ the above equation simplifies to

$$w_{i,i-1}^i = [0 \ 0 \ 1]^T \dot{\theta}_i = z_{i,i}^i \dot{\theta}_i = z\dot{\theta}_i \tag{2.34}$$

as in equation 1.5. Note that $z_{i,i}^i$ is the z axis of the i^{th} body referred to itself. Thus for the whole system

$$\begin{aligned} w &= C\dot{\theta} \\ \text{and } \dot{w} &= C\ddot{\theta} + \dot{C}\dot{\theta} \end{aligned} \tag{2.35}$$

where C is described in Appendix A. $\dot{\theta}$ is the system vector of $\dot{\theta}_{iz}$ joint rates. Hemami [Hemami 82] and Zheng [Zheng 84] projected from w to the state space of inertially measured joint angle rates, while Buchner projected to the relatively measured joint rates $\dot{\theta}$ used here. Hemami [Hemami 82] has shown that $C^T N_1 = 0$, the rows of of C^T being orthogonal to the

columns of N_1. Substituting (2.35) into (2.31) to get a new transform, we get:

$$\begin{bmatrix} I \\ U^{-1}Q \end{bmatrix} C\ddot{\theta} = \begin{bmatrix} C \\ U^{-1}QC \end{bmatrix} \ddot{\theta} \tag{2.36}$$

with the final equations of motion:

$$\begin{bmatrix} C^T & (U^{-1}QC)^T \end{bmatrix} \begin{bmatrix} J & 0 \\ 0 & M \end{bmatrix} \begin{bmatrix} C \\ U^{-1}QC \end{bmatrix} \ddot{\theta} = \begin{bmatrix} C^T & (U^{-1}QC)^T \end{bmatrix} \left(\begin{bmatrix} f \\ g \end{bmatrix} - \begin{bmatrix} 0 \\ MU^{-1}\dot{Q}w \end{bmatrix} - \begin{bmatrix} J \\ MU^{-1}Q \end{bmatrix} \dot{C}\dot{\theta} + \begin{bmatrix} E \\ 0 \end{bmatrix} \tau \right) \tag{2.37}$$

The final form, (2.37), is the same as that developed using the Lagrangian approach. On the RHS of (2.37), $(U^{-1}QC)^T g$ are the forces of gravity, $C^T E\tau$ are the actuator torques, and the remaining terms constitute the centrifugal and Coriolis forces. The LHS coefficient of $\ddot{\theta}$ is the inertia matrix.

The constraint elimination transformation of (2.36) is also Kim's velocity transformation matrix [Kim 86a] and Angeles' matrix of twists. In Hemami's method, the transformation is used principally to eliminate constraint forces, and secondly to project to relative coordinates, while for Kim and Angeles the matrix is used with the kinetic energy expression to project to relative coordinates, since the hinge constraint forces do not naturally appear.

Although equation (2.37) has the same final form as the equations developed using Lagrangian techniques, it is important to realise that it is not the principles of physics that determine the efficiency of the computation. The procedure and organisation for the formation of the transformation matrices play a more significant role.

2.2.5 Discussion

In this chapter a number of formulations have been discussed. In general, they form three major schools of thought: the $O(n^2)$ composite rigid body method, which is popular in robotics; $O(n)$ local axis projection algorithms; and $O(n^2)$ or $O(n^3)$ orthogonal complement global projection algorithms.

Composite rigid body algorithms have been parallelised by several researchers. Parallel $O(n + \lceil \log_2 n \rceil)$ and $O(\lceil \log_2 n \rceil)$ algorithms have been developed for n and $\frac{n(n+1)}{2}$ cpu architectures, respectively. One drawback of these parallel algorithms is that the architecture developed for calculating the matrices is not ideal for solving the matrix equation that follows.

Recursive $O(n)$ techniques were originally developed by roboticists, but have since been adopted by others. Forces and moments are locally projected onto hinge axes in a recursive manner. $O(n)$ algorithms are faster on a single cpu for systems with more than six degrees of freedom, but as noted by Fijany, a single chain cannot be parallelised at the equation level using this approach (parallel matrix multipliers can be applied, however).

Orthogonal complement algorithms use a global Jacobian transformation to project from Cartesian to other coordinates such as relative joint angles, and eliminate internal constraint forces. Bae, Hwang and Haug's algorithm [Bae 88a] is a variation in which the terms are grouped in a different manner so that the projection is locally performed on an 'accumulated inertia' matrix K_i. With the exception of Hwang, Bae and Haug's work [Hwang 88] which used a shared bus architecture, the orthogonal complement algorithms developed by Hemami, Kim, Ibrahim, Angeles and Lilly have not been parallelised.

In the next chapter, an efficient version of the Newton Euler State Space algorithm is developed, and in Chapter 4, a parallel version of this algorithm for a 2 dimensional mesh architecture is presented.

Chapter 3

Efficient Computation using Body Coordinates

The thesis has thus far provided an overview of the fundamentals and some current research in the area of multibody dynamics. This chapter focuses on the efficiency and complexity of the Newton Euler State Space formulation in the context of a single cpu simulation.

In this chapter each individual equation in (2.37) is referred to its body coordinates, rather than the inertial frame used by Kim and Vanderploeg [Kim 86a], or the mixed frames proposed by Hemami (in Hemami's approach $\ddot{x}_i$ equations were referred to the inertial frame, and $\dot{w}_i$ equations were referred to body coordinates). It is shown that this leads to a computationally efficient recursive definition of the matrix elements in which the final transformation matrix is the same as the natural orthogonal complement matrix presented by Angeles and Ma [Angeles 88].

In addition, a connectivity matrix is used to extend the formulation to include branched systems, and a novel algorithm to calculate the force vector is presented. The computational complexity of the overall formulation, which includes the cost of the inertia matrix, the force vector, and the matrix solver, is used as a metric to compare other formulations.

Finally, two linkages are simulated which demonstrate the correctness of the formulation. A PUMA 600 is modeled using joint angle, velocity, and acceleration profiles suggested by Angeles and Ma [Angeles 88]. A six degree of freedom model of the human torso is simulated using the same profiles to demonstrate the branched algorithm.

3.1 Referral of the Equations to Body Coordinates

3.1.1 Single Chain Algorithm

In equation (2.30), $\ddot{x}$, measured with respect to the inertial frame, is also referred to the inertial coordinate system. If each element in the $\ddot{x}$ system vector is instead referred to its own *bcs*, (2.30) can be modified to

$$\ddot{x}_{bcs} = (U^{-1}Q)_{bcs}\dot{w} + (U^{-1}\dot{Q}w)_{bcs} \tag{3.1}$$

An example of $(U^{-1}Q)_{bcs}$ for a five link single chain is contained in Appendix A. By referring (2.30) to body coordinates, the computation of the transformation matrix in equation (2.36), now modified to $[C^T \;\; ((U^{-1}Q)_{bcs}C)^T]^T$, becomes more efficient due to its recursive nature. As an example, consider the computation of matrix elements (2,1) and (3,1) in $(U^{-1}Q)_{bcs}C$:

$$\begin{aligned}
[(U^{-1}Q)_{bcs}C]_{2,1} &= \begin{bmatrix} A_1^2\tilde{d}_{2,1}^1 & \tilde{k}_{2,2}^2 & 0\cdots \end{bmatrix} \begin{bmatrix} z \\ A_1^2 z \\ A_2^3 A_1^2 z \\ \vdots \\ A_{N-1}^N \cdots A_1^2 z \end{bmatrix} \\
&= A_1^2\tilde{d}_{2,1}^1 z + \tilde{k}_{2,2}^2 A_1^2 z \\
&= (A_1^2\tilde{d}_{2,1}^1 A_2^1 + \tilde{k}_{2,2}^2)A_1^2 z \\
&= (A_1^2 d_{2,1}^1 + k_{2,2}^2) \times A_1^2 z \qquad (3.2) \\
&= \rho_{2,1}^2 \times A_1^2 z \\
&= (A_1^2(\rho_{1,1}^1 - l_{1,2}^1) + k_{2,2}^2) \times A_1^2 z \qquad (3.3) \\
\text{where } \rho_{1,1}^1 &= k_{1,1}^1 \\
\text{and } d_{i+1,i}^i &= k_{i,i}^i - l_{i,i+1}^i \qquad (3.4)
\end{aligned}$$

where $d_{i+1,i}^i$ is the vector measured across body i from hinge $i+1$ to hinge i. $\rho_{2,1}^2$ is the vector from the centre of gravity of body 2 to the lower hinge of body 1, in body 2 coordinates.

Similarly, element (3,1) is computed thus:

$$
\begin{aligned}
[(U^{-1}Q)_{bcs}C]_{3,1} &= \begin{bmatrix} A_2^3A_1^2\tilde{d}_{2,1}^1 & A_2^3\tilde{d}_{3,2}^2 & \tilde{k}_{3,3}^3 & 0\cdots \end{bmatrix} \begin{bmatrix} z \\ A_1^2z \\ A_2^3A_1^2z \\ \vdots \\ A_{N-1}^N\cdots A_1^2z \end{bmatrix} \\
&= A_2^3A_1^2\tilde{d}_{2,1}^1z + A_2^3\tilde{d}_{3,2}^2A_1^2z + \tilde{k}_{3,3}^3A_2^3A_1^2z \\
&= (A_2^3A_1^2\tilde{d}_{2,1}^1A_2^1A_3^2 + A_2^3\tilde{d}_{3,2}^2A_3^2 + \tilde{k}_{3,3}^3)A_2^3A_1^2z \\
&= (A_2^3A_1^2d_{2,1}^1 + A_2^3d_{3,2}^2 + k_{3,3}^3) \times A_2^3A_1^2z \qquad (3.5) \\
&= \rho_{3,1}^3 \times A_2^3A_1^2z \\
&= (A_2^3(\rho_{2,1}^2 - l_{2,3}^2) + k_{3,3}^3) \times A_2^3A_1^2z \qquad (3.6)
\end{aligned}
$$

In this case $\rho_{3,1}^3$ is the vector from the centre of gravity of body 3 to the lower hinge of body 1 (joint 1), in bcs_3, while $A_2^3A_1^2z$ can be defined as $c_{3,1}^3$.

The (i,j)th term of $[U^{-1}Q_{bcs}C]$ is therefore

$$
\begin{aligned}
[(U^{-1}Q)_{bcs}C]_{i,j} &= \rho_{i,j}^i \times c_{i,j}^i \qquad (3.7) \\
\text{or as a system, } [(U^{-1}Q)_{bcs}C] &= [\rho \times C] \qquad (3.8) \\
\text{where } \rho_{j,j}^j &= k_{j,j}^j; \quad c_{j,j}^j = z; \\
\rho_{i,j}^i &= A_{i-1}^i(\rho_{i-1,j}^{i-1} - l_{i-1,i}^{i-1}) + k_{i,i}^i \qquad (3.9) \\
\text{and } c_{i,j}^i &= A_j^iz = A_{i-1}^ic_{i-1,j}^{i-1} \qquad (3.10)
\end{aligned}
$$

where $[\rho \times C]$ is a symbolic representation. $\rho_{i,j}^i$ is equivalent in magnitude to the $s_{i,j}$ vector used by Angeles [Angeles 88] but with a reversed direction, and $c_{i,j}^i$ is equivalent to Angeles' $e_{i,j}$. In this thesis, the order of the cross product is reversed to compensate for the change of direction in ρ_{ij}^i. The $3\times N$ matrices for $c_{i,j}^i$ and $\rho_{i,j}^i \times c_{i,j}^i$, where $i = n$; $j = 1,...n$; are the components of the standard end effector Jacobian.

In Angeles' method, J and M are factorised to $\bar{J}^T\bar{J}$ and $\bar{M}^T\bar{M}$ where $\bar{M}_i = \sqrt{m_i}I_{3\times3}$. In this thesis, J and M remain unfactorised.

Now that we have efficiently defined $(U^{-1}Q_{bcs}C)$ to be $(\rho \times C)$, equation 2.37 is modified to:

$$[C^T \ (\rho \times C)^T] \begin{bmatrix} JC \\ M(\rho \times C) \end{bmatrix} \ddot{\theta} = \begin{bmatrix} C^T & (\rho \times C)^T \end{bmatrix} \left(\begin{bmatrix} f \\ g_{bcs} \end{bmatrix} + \begin{bmatrix} 0 \\ M(U^{-1}\dot{Q}w)_{bcs} \end{bmatrix} - \begin{bmatrix} J \\ M(\rho \times C) \end{bmatrix} \dot{C}\dot{\theta} + \begin{bmatrix} E \\ 0 \end{bmatrix} \tau \right)$$

$$\text{or} \quad \mathcal{M}\ddot{\theta} = [\bar{H}^T H]\ddot{\theta} = b \tag{3.11}$$

In equation (3.11), the ith component of g_{bcs} is $g^i_{i,0} = m_i A^i_0 g$, the force due to gravity of each body when it is referred to body coordinates. The transformation matrix $\bar{H} = [C^T \ (\rho \times C)^T]$ on the LHS of (3.11) was developed independently of Angeles' work. Another interesting feature of the $\bar{H}$ matrix is its recursive nature. The matrix can be considered as a series of N $3 \times N$ Jacobian matrices, the ith one representing a manipulator with i links. This characteristic was also noted by [Lilly 91].

3.1.2 Branched systems algorithm

Consider the branched system shown in Fig. 3.1. The two branches consist of branch1 = {1,2,3} and branch2 ={1,4,5}. Each new branch starts from the home body (1) and is completely labelled out to the tip before a new branch is started. If a branch already has some bodies previously labelled, the next unlabelled body is tagged with the lowest unused integer.

Consider body 1. Joint j1 connects body 1 to ground, joint j2 connects body 2 to body 1, and joint j4 connects body 4 to body 1. Define $l^1_{1,2}$ and $l^1_{1,4}$ as the vectors from the cog of body 1 to the hinge connecting bodies 2 and 4 to 1, referred to bcs_1. The l vectors point to hinges that are further from the home body. $k^1_{1,1}$ is the vector from the cog of body 1 to the joint connecting body 1 to ground (joint 1), or in the general case the body which is one closer to home. Then, define $d^1_{2,1} = k^1_{1,1} - l^1_{1,2}$ as the vector from joint 2 to joint 1, across body 1, in bcs_1. In general the vector across body i from joint j to joint i can be defined as $d^i_{j,i} = k^i_{i,i} - l^i_{i,j}$ where $i = prev(j)$, the body that is one closer than j on the direct path to the home body.

The orientation of each body's coordinate system (not shown in Fig. 3.1) remains the same as for the simple chain (e.g. the x_4 axis is parallel to the common perpendicular between the z_1

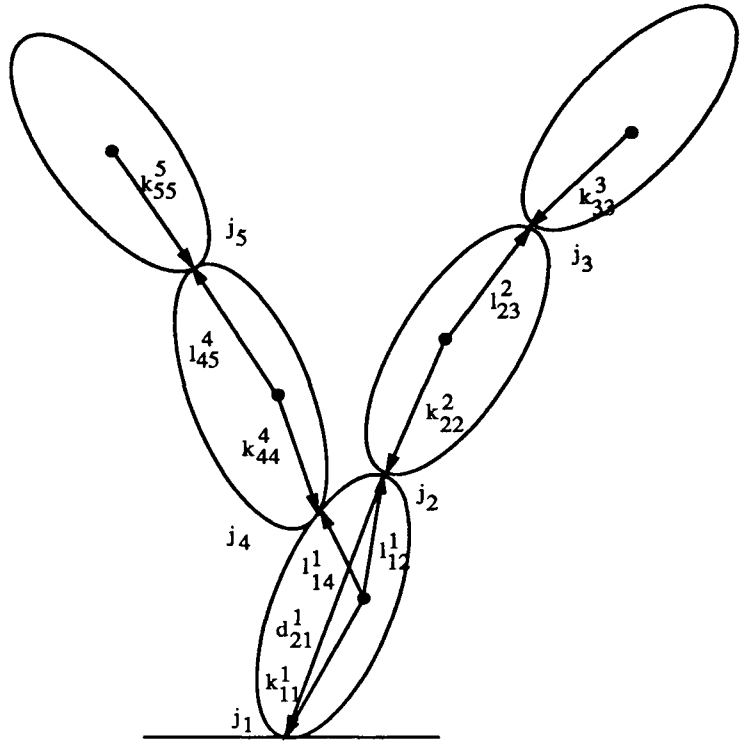

Figure 3.1: A branched tree of five bodies

and z_4 axes).

From Fig. 3.1, equations of motion similar to (2.26) may be defined (see Appendix B). The structure of the matrices differ from the single chain configuration as extra elements arise for bodies which have more than one outward body attached (e.g. body 1).

The elimination of constraint forces proceeds as before, with the statement of kinematic translational hinge constraint equations, referred to the inertial frame:

$$
\begin{aligned}
x^0_{1,0} \;+\; & A^0_1 k^1_{1,1} = 0 \\
x^0_{2,0} \;+\; & A^0_2 k^2_{2,2} - x^0_{1,0} - A^0_1 l^1_{1,2} = 0 \\
x^0_{3,0} \;+\; & A^0_3 k^3_{3,3} - x^0_{2,0} - A^0_2 l^2_{2,3} = 0 \\
x^0_{4,0} \;+\; & A^0_4 k^4_{4,4} - x^0_{1,0} - A^0_1 l^1_{1,4} = 0
\end{aligned}
$$

$$x^0_{5,0} \;+\; A^0_5 k^5_{5,5} - x^0_{4,0} - A^0_4 l^4_{4,5} = 0 \tag{3.12}$$

Referring each equation to body coordinates and then differentiating twice yields the branched versions of matrices U, U^{-1}, Q and $\dot{Q}w$ (defined in Appendix B). As in the single chain case, the translational constraint forces are eliminated due to the orthogonality between $[H_1^T \; H_{2bcs}^T]^T$ and $[I^T \;\; ((U^{-1}Q)_{bcs}C)^T]^T$, and the rotational constraints are eliminated by the orthogonality between C and N_1 [Hemami 82].

To automate the derivation of the kinematic constraints and the equations of motion, it is necessary to obtain U and U^{-1}. This is because the C, $(U^{-1}Q)_{bcs}$ and $(\rho \times C)$ matrices (shown below) have structures identical to U^{-1}. U^{-1} is a trivially generated connectivity matrix describing the bodies that appear along the path between a given body and the home body. For example, row 5 in U^{-1} (in Appendix B) has an I in positions 1, 4, and 5, indicating that the path from home to body 5 includes bodies 1, 4 and 5. It is also useful to identify adjacent bodies. Considering body 4, the body previous to it in the path is found by searching for a $-I$ in the fourth row of U. This occurs in column 1. Once U^{-1} is generated, H_2 and U are easily constructed. U and U^{-1} can be incorporated into simple algorithms for determining other matrices e.g. the computation of $\dot{Q}w$ and $(\rho \times C)$.

As an example of the underlying structure of the matrices, the $(U^{-1}Q)_{bcs}$ (3N $\times$ 3N) and C (3N $\times$ N), when computed from U^{-1}, Q, and the kinematics equations (1.5) and (1.6) (Appendix B), are:

$$(U^{-1}Q)_{bcs} = \begin{bmatrix} \tilde{k}^1_{1,1} & 0 & 0 & 0 & 0 \\ A^2_1\tilde{d}^1_{2,1} & \tilde{k}^2_{2,2} & 0 & 0 & 0 \\ A^3_2A^2_1\tilde{d}^1_{2,1} & A^3_2\tilde{d}^2_{3,2} & \tilde{k}^3_{3,3} & 0 & 0 \\ A^4_1\tilde{d}^1_{4,1} & 0 & 0 & \tilde{k}^4_{4,4} & 0 \\ A^5_4A^4_1\tilde{d}^1_{4,1} & 0 & 0 & A^5_4\tilde{d}^4_{5,4} & \tilde{k}^5_{5,5} \end{bmatrix}, \text{and } C = \begin{bmatrix} z & 0 & 0 & 0 & 0 \\ A^2_1z & z & 0 & 0 & 0 \\ A^3_2A^2_1z & A^3_2z & z & 0 & 0 \\ A^4_1z & 0 & 0 & z & 0 \\ A^5_4A^4_1z & 0 & 0 & A^5_4z & z \end{bmatrix} \tag{3.13}$$

Post multiplying $(U^{-1}Q)_{bcs}$ by C results in a 3N $\times$ N matrix similar to the $(\rho \times C)$ matrix

generated in equation 3.8 but with non-zero elements in the same positions as those in U^{-1}:

$$[U^{-1}Q_{bcs}C] = [\rho \times C] = \begin{bmatrix} \rho^1_{11} \times c^1_{11} & 0 & 0 & 0 & 0 \\ \rho^2_{21} \times c^2_{21} & \rho^2_{22} \times c^2_{22} & 0 & 0 & 0 \\ \rho^3_{31} \times c^3_{31} & \rho^3_{32} \times c^3_{32} & \rho^3_{33} \times c^3_{33} & 0 & 0 \\ \rho^4_{41} \times c^4_{41} & 0 & 0 & \rho^4_{44} \times c^4_{44} & 0 \\ \rho^5_{51} \times c^5_{51} & 0 & 0 & \rho^5_{54} \times c^5_{54} & \rho^5_{55} \times c^5_{55} \end{bmatrix} \tag{3.14}$$

Calculating $\rho^i_{i,j}$ requires knowledge of the path between bodies i and j, which is found in U and U^{-1}.

A recursive algorithm to generate $c^i_{i,j}$, $\rho^i_{i,j}$ and $[(U^{-1})Q_{bcs}C]^i_{i,j}$ that uses U and U^{-1} to take into account the system topology is:

for $j = 1$ to N {
  for $i = 1$ to N {
    for $p = 1$ to i
      if $(U_{i,p} = -I)$ $p(i) = p$;
    $c^i_{i,i} = z$;
    $\rho^i_{i,i} = k^i_{i,i}$;
    $c^i_{i,j} = U^{-1}_{i,j} A^i_{p(i)} c^{p(i)}_{p(i),j}$; $i < j$
    $\rho^i_{i,j} = U^{-1}_{i,j}(A^i_{p(i)}(\rho^{p(i)}_{p(i),j} - l^{p(i)}_{p(i),i}) + k^i_{i,i})$; $i < j$
    $[U^{-1}Q_{bcs}C]^i_{i,j} = \rho^i_{i,j} \times c^i_{i,j}$;
  }
}

(3.15)

Since U^{-1} and U are trivially generated, arbitrarily branched systems can be modeled in an automated manner by utilising this algorithm.

3.2 Inertia Matrix Calculation and Complexity

Simulation of the dynamics requires the formation of the terms in (3.11), followed by the linear equation solution of $\mathcal{M}\ddot{\theta} = b$. In this section the computational complexity of assembling

the inertia matrix $\mathcal{M}$ for the serial chain algorithm (on one cpu) is estimated. Because the complexity of the branched algorithm depends greatly on the system topology, it is not useful to estimate the cost of an arbitrarily branched system.

The inertia matrix in (3.11) is factored into two rectangular matrices, $\bar{H}^T = [C^T \ (\rho \times C)^T]$ (N×6N) and

$$H = \begin{bmatrix} JC \\ M(\rho \times C) \end{bmatrix}, \tag{3.16}$$

which is (6N ×N), where $(\rho \times C) = U^{-1}Q_{bcs}C$. The algorithm for calculating (3.16) is:

$$\begin{array}{l}
\text{for } j = 1 \text{ to N } \{ \\
\quad \text{for } i = j \text{ to N } \{ \\
\qquad c^i_{i,i} = z^i_{i,i} \quad ; \\
\qquad \rho^i_{i,i} = k^i_{i,i} \quad ; \\
\qquad c^i_{i,j} = A^i_{i-1} c^{i-1}_{i-1,j} \quad ; \ i > j \\
\qquad \rho^i_{i,j} = A^i_{i-1}(\rho^{i-1}_{i-1,j} - l^{i-1}_{i-1,i}) + k^i_{i,i}) \quad ; \ i > j \\
[U^{-1}Q_{bcs}C]^i_{i,j} = \rho^i_{i,j} \times c^i_{i,j} \quad ; \\
\qquad H^i_{i,j} = \begin{bmatrix} J^i_i c^i_{i,j} \\ m_i(\rho^i_{i,j} \times c^i_{i,j}) \end{bmatrix} \\
\quad \} \\
\}
\end{array} \tag{3.17}$$

The computations in (3.17) can be decomposed into the following phases:

1. Forming the rotation matrices A^i_{i-1} which refer a vector in bcs_{i-1} to bcs_i:

$$A^i_{i-1} = \begin{bmatrix} \cos\theta_i & \cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i \\ -\sin\theta_i & \cos\alpha_i \cos\theta_i & \sin\alpha_i \cos\theta_i \\ 0 & -\sin\alpha_i & \cos\alpha_i \end{bmatrix} \tag{3.18}$$

   Assuming the n ($\sin\theta_i$, $\cos\theta_i$) values have been calculated, and $\sin\alpha_i$ and $\cos\alpha_i$ are constants, this phase requires $4n$ multiplies and 0 adds for an n-body chain.

2. Calculating $c^i_{i,j}$, which is the $z^j_{j,j}$ axis referred to bcs_i coordinates. Consider the system C matrix for a five body chain:

$$C = \begin{bmatrix} z^1_{1,1} & 0 & 0 & 0 & 0 \\ A^2_1 z^1_{1,1} & z^2_{2,2} & 0 & 0 & 0 \\ A^3_1 z^1_{1,1} & A^3_2 z^2_{2,2} & z^3_{3,3} & 0 & 0 \\ A^4_1 z^1_{1,1} & A^4_2 z^2_{2,2} & A^4_3 z^3_{3,3} & z^4_{4,4} & 0 \\ A^5_1 z^1_{1,1} & A^5_2 z^2_{2,2} & A^5_3 z^3_{3,3} & A^5_4 z^4_{4,4} & z^5_{5,5} \end{bmatrix} ; \text{ where } c^j_{j,j} = z^j_{j,j} = z \text{ and } c^i_{i,j} = A^i_{i-1} c^{i-1}_{i-1,j} \tag{3.19}$$

Each $c^i_{i,j}$ in equation (3.19) is a matrix-vector product requiring 8× and 5+. The C matrix thus requires $\frac{(n-2)(n-1)}{2}(8\times, 5+)$. This is because the first diagonal consists only of constant $z^j_{j,j}$ vectors, while the second diagonal refers constant z^i_i vectors to bcs_{i+1}, which is equivalent to just selecting the last column of the A^{i+1}_i matrix. The other diagonals require the computation of (3.19).

3. Calculating $\rho^i_{i,j}$, the displacement vector from the centre of gravity of body i to the j^{th} hinge:

$$[U^{-1}Q]_{bcs} = \rho = \begin{bmatrix} \rho^1_{1,1} & 0 & 0 & 0 & 0 \\ \rho^2_{2,1} & \rho^2_{2,2} & 0 & 0 & 0 \\ \rho^3_{3,1} & \rho^3_{3,2} & \rho^3_{3,3} & 0 & 0 \\ \rho^4_{4,1} & \rho^4_{4,2} & \rho^4_{4,3} & \rho^4_{4,4} & 0 \\ \rho^5_{5,1} & \rho^5_{5,2} & \rho^5_{5,3} & \rho^5_{5,4} & \rho^5_{5,5} \end{bmatrix} ;$$

$$\rho^i_{i,j} = A^i_{i-1}(\rho^{i-1}_{i-1,j} - l^{i-1}_{i-1,i}) + k^i_{i,i} \tag{3.20}$$

The first diagonal values of the ρ matrix ($i = j$) are $\rho^i_{i,i} = k^i_{i,i}$, so no computations are necessary. The rest of the elements require $\frac{(n)(n-1)}{2}(8\times, 11+)$.

4. Multiplying every element in the i^{th} row of the C matrix by J^i_i.

$$J^i_i c^i_{i,j} \qquad\qquad j = 1, \ldots i \tag{3.21}$$

The first diagonal, $i = j$, can be calculated off line as the multiplicands and the results are constant, and hence the cost will not be counted here. The rest of the matrix requires $\frac{(n)(n-1)}{2}(9\times, 6+)$.

5. Calculating the cross product of $c^i_{i,j}$ and $\rho^i_{i,j}$, and then scaling by the mass m_i.

$$m_i(\rho_{i,j} \times c_{i,j}) \tag{3.22}$$

The first diagonal can be calculated off line. The cost of the rest of this phase is $\frac{(n)(n-1)}{2}(9\times,3+)$. Note that we have multiplied by J^i_i and m_i, instead of $\bar{J}^i_i$ and $\sqrt{m_i}$. This is because the force vector algorithm (to be described later) use C and $(\rho \times C)$ rather than $\bar{J}C$ or $\bar{M}(\rho \times C)$.

6. The final phase in the formation of the inertia matrix is to multiply C^T by JC and $(\rho \times C)^T$ by $M(\rho \times C)$ according to:

$$\mathcal{M}_{ik} = \sum_{j=i}^{n}\left[(c^j_{j,i})^T \cdot (J^j_j c^j_{j,k}) + (\rho^j_{j,i} \times c^j_{j,i})^T \cdot (m_j(\rho^j_{j,k} \times c^j_{j,k}))\right] \tag{3.23}$$

Since the inertia matrix $\mathcal{M}$ is symmetric, only half of the matrix needs to be computed. The complexity for this phase is then $(\frac{n^3}{6} + \frac{n^2}{2} + \frac{n}{3} - n)(6\times,5+) - \frac{n(n-1)}{2}(1\times,1+)$. The $-n(6\times,5+)$ appears because terms multiplying two first diagonal elements, being constants, can be calculated off line. The $-\frac{n(n-1)}{2}(1\times,1+)$ term is present because $m_j(\rho^j_{j,j} \times c^j_{j,j})$ vectors have zeros as their last elements.

The total computational complexity for forming the inertia matrix is the sum of the previous phases' complexities, and is presented in Table 3.1. The algorithm is slightly more efficient than Angeles' because some of the terms are calculated off line. The complexity polynomial given by Angeles also has minor errors, and these have been corrected in Table 3.1, which presents the complexity for a number of single cpu algorithms. Table 3.2 gives data on the costs for calculating the inertia matrix for linkages of various size for various inertia matrix algorithms. The algorithm Lilly III, developed by Lilly and Orin [Lilly 91] and similar to [Bae 88a], is the most efficient in Table 3.2. In this study however, the proposed algorithm will be parallelised because it posesses an architecturally attractive computational structure, compared to Hwang's parallel formulation discussed earlier.

Algorithm		force vector	inertia matrix	solver	$\sum$
Angeles	×	$105n - 99$	$n^3 + \frac{39n^2}{2} - \frac{33n}{2} + 8$	$\frac{n^3}{6} + \frac{3n^2}{2} + \frac{4n}{3}$	$\frac{7n^3}{6} + 21n^2 + \frac{539n}{6} - 91$
	+	$90n - 95$	$\frac{5n^3}{6} + \frac{29n^2}{2} - \frac{46n}{3} + 5$	$\frac{n^3}{6} + \frac{3n^2}{2} + \frac{n}{3}$	$n^3 + 16n^2 + 75n - 90$
Proposed	×	$3n^2 + 77n - 44$	$n^3 + \frac{39n^2}{2} - \frac{49n}{2} + 8$	$\frac{n^3}{6} + \frac{3n^2}{2} - \frac{2n}{3}$	$\frac{7n^3}{6} + 24n^2 + \frac{311n}{6} - 36$
	+	$3n^2 + 62n - 43$	$\frac{5n^3}{6} + \frac{29n^2}{2} - \frac{61n}{3} + 5$	$\frac{n^3}{6} + n^2 - \frac{7n}{6}$	$n^3 + \frac{37n^2}{2} + \frac{81n}{2} - 38$
Walker	×	$137n - 22$	$12n^2 + 56n - 27$	$\frac{n^3}{6} + \frac{3n^2}{2} - \frac{2n}{3}$	$\frac{n^3}{6} + \frac{27n^2}{2} + \frac{577n}{3} - 49$
	+	$101n - 11$	$7n^2 + 67n - 53$	$\frac{n^3}{6} + n^2 - \frac{7n}{6}$	$\frac{n^3}{6} + 8n^2 + \frac{1001n}{6} - 64$
Brandl	×				$250n - 222$
	+				$220n - 198$
Fijany	×		$\frac{9n^2}{2} + \frac{231n}{2} - 181$		
	+		$4n^2 + 88n - 137$		
Lilly I	×		$n^3 + 22n^2 - 35n + 12$		
	+		$n^3 + 15n^2 - 26n + 10$		
Lilly III	×		$\frac{25n^2}{2} + \frac{49n}{2} - 37$		
	+		$8n^2 + 40n - 48$		
He	×	$91n - 97$			
	+	$86n - 96$			

Table 3.1: Complexity of Forward Dynamics for One Cpu

Algorithm	4	5	6	7
Proposed	286×, 209+	498×, 370+	779×, 585+	1135×, 851+
Walker	389×, 327+	555×, 457+	741×, 601+	953×, 759+
Angeles	318×, 229+	538×, 395+	827×, 615+	1191×, 894+
Fijany	353×, 279+	509×, 403+	664×, 535+	848×, 675+
Lilly I	288×, 210+	512×, 380+	810×, 610+	1188×, 906+
Lilly III	261×, 240+	398×, 352+	560×, 480+	747×, 624+

Table 3.2: Computational Complexity for Inertia Matrix Assembly for One Cpu

3.3 Force Vector Calculation and Complexity

The RHS of equation (3.11), which includes the actuator forces or torques plus the effects of Coriolis, centrifugal, and gravitational forces (f_v), appears to be computationally expensive when calculated according to equation (3.11). It is commonly calculated more efficiently using the O(n) inverse dynamics algorithm proposed by Luh, Walker and Paul [Luh 80] (Chapter 2). An O(n) algorithm which is more efficient than Luh's, proposed by Angeles and Ma [Angeles 89] (Table 3.1), uses Kane's generalised force approach.

In this section a new algorithm based on Angeles' algorithm is proposed. Although the proposed

algorithm is of $O(n^2)$ complexity rather than $O(n)$, it is slightly more efficient than Angeles' for systems with six or less degrees of freedom, as it makes use of the $\bar{H}^T = [C\ (\rho \times C)]^T$ matrix, which has previously been calculated during the computation of the inertia matrix. In contrast, Angeles' algorithm does not make use of the previous inertia matrix calculations.

Consider a chain of n bodies. Assuming there are no dissipative forces such as friction, the equation of torque is [Angeles 89]:

$$f_v = \sum_{j=1}^{n}\left[\left(\frac{\partial w_j}{\partial \dot{q}}\right)^T \dot{L}^j_{j,0} + \left(\frac{\partial \dot{x}_j}{\partial \dot{q}}\right)^T m_j(\ddot{x}^j_{j,0} - g)\right] \tag{3.24}$$

$\frac{\partial w_j}{\partial \dot{q}}$ and $\frac{\partial \dot{x}_j}{\partial \dot{q}}$ are known as the partial velocities, and are the jth $3\times n$ Jacobian row matrices found within the C and $(\rho \times C)$ matrices, respectively. The angular momentum derivative $\dot{L}^j_{j,0}$ from (1.12), measured with respect to the inertial frame and referred to the jth frame, is:

$$\dot{L}^j_{j,0} = J^j_j \dot{w}^j_{j,0} + w^j_{j,0} \times J^j_j w^j_{j,0} \tag{3.25}$$

The translational acceleration $\ddot{x}_j$ is the acceleration of the x_j vector measured from the inertial frame to the centre of gravity of body j. It may be replaced by $\ddot{p}_j$, which as defined here includes the effect of the acceleration due to gravity (i.e. $\ddot{p}_j = \ddot{x}_j - g$). Angeles' recursive equation for $\ddot{p}_j$ is:

$$\ddot{p}^j_{j,0} = A^j_{j-1}\left(\ddot{p}^{j-1}_{j-1,0} + \dot{w}^{j-1}_{j-1,0} \times l^{j-1}_{j-1,j} + w^{j-1}_{j-1,0} \times w^{j-1}_{j-1,0} \times l^{j-1}_{j-1,j}\right) - \dot{w}^j_{j,0} \times k^j_{j,j} - w^j_{j,0} \times w^j_{j,0} \times k^j_{j,j} \tag{3.26}$$

An efficient algorithm for calculating $\ddot{p}^i_{i,0}$ and $\dot{L}^i_{i,0}$ is [Angeles 89]:

$$w^i_{i,0} = A^i_{i-1} w^{i-1}_{i-1,0} + z^i_{i,i}\dot{\theta}_i \tag{3.27}$$

$$\dot{w}^i_{i,0} = A^i_{i-1} \dot{w}^{i-1}_{i-1,0} + w^i_{i,0} \times z^i_{i,i}\dot{\theta}_i + z^i_{i,i}\ddot{\theta}_i \tag{3.28}$$

$$\text{Let}\quad \tilde{W}_i = \tilde{\dot{w}}^i_{i,0} + (\tilde{w}^i_{i,0})^2. \tag{3.29}$$

$$\text{Then}\quad \ddot{p}^i_{i,0} = A^i_{i-1}\left(\ddot{p}^{i-1}_{i-1,0} + \tilde{W}_{i-1} l^{i-1}_{i-1,i}\right) - \tilde{W}_i k^i_{i,i} \tag{3.30}$$

$$\dot{L}^i_{i,0} = J^i_i \dot{w}^i_{i,0} + w^i_{i,0} \times J^i_i w^i_{i,0}. \tag{3.31}$$

Equations (3.27) and (3.28) are simply kinematic equation 1.5 referred to body coordinates. Equation (3.29) combined with (3.30) is an efficient method of representing (3.26), while (3.31) is (1.12) referred to body coordinates.

Angeles substituted the 3×n row matrices in C and $(\rho \times C)$ into (3.24) to get:

$$f_{v(i)} = \sum_{j=i}^{n} \left[(c_{j,i}^{j})^T \dot{L}_j + (\rho_{j,i}^{j} \times c_{j,i}^{j})^T m_j \ddot{p}_{j,0}^{j} \right] \qquad i = 1, ...n \tag{3.32}$$

and

$$f_{v(i)} = (c_{i,i}^{i})^T \sum_{j=i}^{n} A_j^i \left[\dot{L}_j + \rho_{j,i}^{j} \times m_j \ddot{p}_{j,0}^{j} \right] \qquad i = 1, ...n \tag{3.33}$$

where $(c_{j,i}^{j})^T = (c_{i,i}^{i})^T A_j^i$ and $(\rho_{j,i}^{j} \times c_{j,i}^{j})^T m_j \ddot{p}_{j,0}^{j} = (c_{j,i}^{j})^T (\rho_{j,i}^{j} \times m_j \ddot{p}_{j,0}^{j})$. Angeles showed that equation (3.33) is computable in a backward recursion in O(n) time (Table 3.1).

3.3.1 O(n^2) force vector algorithm and complexity

In this section we define a new algorithm for the force vector which makes use of the $[C^T \ (\rho \times C)^T]$ matrix. Equation (3.24) may also be written as

$$f_v = \begin{bmatrix} C^T & (\rho \times C)^T \end{bmatrix} \begin{bmatrix} \dot{L} \\ m\ddot{p} \end{bmatrix} \tag{3.34}$$

where $[\dot{L} \ m\ddot{p}]$ is the vector of $\dot{L}_i$ and $m_i \ddot{p}_i$, $i = 1, ...n$. Thus it can be seen that the momentum derivatives are related to (3.11) by

$$\bar{H}^T \begin{bmatrix} \dot{L} \\ m\ddot{p} \end{bmatrix} = \bar{H}^T \left(\begin{bmatrix} f \\ g_{bcs} \end{bmatrix} + \begin{bmatrix} 0 \\ M(U^{-1}\dot{Q}w)_{bcs} \end{bmatrix} - \begin{bmatrix} J \\ M(\rho \times C) \end{bmatrix} \dot{C}\dot{\theta} \right) \tag{3.35}$$

The complexity of computing equation (3.34) is O(n^2) due to the matrix-vector multiplication, and requires the initial computation of $\dot{L}$ and $m\ddot{p}$ from equations (3.27) to (3.31), followed by the matrix computation of (3.34). The cost may be estimated as follows:

1. Calculating $w_{i,0}^{i}$ requires $(n-1)(8\times, 6+)$ using (3.27). $w_{1,0}^{1} = z_{1,1}^{1}\dot{\theta}_1$ is a scalar assignment operation.

2. Calculating $\dot{w}_{i,0}^{i}$ requires the last term in equation (3.28) to be ignored (the $\ddot{\theta}_i$ are set to zero to exclude the inertial terms). The second term is a cross product which only requires $2\times$ since $z_{i,i}^{i}\dot{\theta}_i$ has only one nonzero element. Thus the complexity is $(n-1)(10\times, 7+)$, with $\dot{w}_{1,0}^{1} = 0$.

3. Calculating $\tilde{W}_i$ requires $(n-1)(6\times,9+)$, plus $1\times$ for $\tilde{W}_1$.

4. Calculating $\ddot{p}_i$ requires $(n-1)(26\times,23+)$, plus $12\times,9+$ to calculate

$$\ddot{p}^1_{1,0} = -A^1_0 g - \tilde{W}_1 k^1_{1,1} \tag{3.36}$$

where $g = [0\ 0\ -9.81]^T$. The first term in (3.36) compensates for gravity by adding an additional acceleration to that induced by the angular velocity $w^0_{i,0}$.

5. Calculating $m_i\ddot{p}_i$ requires $n(3\times)$.

6. Calculating $\dot{L}_i$ in (3.31) requires $(n-1)(24\times,18+)$ plus $15\times,10+$ for $\dot{L}^1_{1,0}$.

7. Calculating the matrix-vector product in equation (3.34). The first diagonal elements in C are $z = [0\ 0\ 1]^T$, and so no cost is associated with dot products involving these terms. The complexity of (3.34) is

$$\frac{n(n-1)}{2}(6\times,5+) + n(3\times,3+) + \frac{n(n-1)}{2}(1+) = 3n^2(\times,+). \tag{3.37}$$

The first term calculates the dot products

$$[c^i_{i,j}\ (\rho^i_{i,j} \times c^i_{i,j})] \begin{bmatrix} \dot{L}_i \\ m_i\ddot{p}_i \end{bmatrix} \tag{3.38}$$

for those computations not involving the first diagonal. The second term adds the computations for $[(\rho^i_{i,i} \times c^i_{i,i})][m_i\ddot{p}_i]$, and the last term describes the operation in which the rotational and translational dot products are added together to form the force vector.

The complexity for the proposed force vector algorithm is given in Table 3.1. Complexity values for multibody systems of various size are given in Table 3.3, which shows that the algorithm presented here is more efficient than Angeles and Walker's algorithms for systems of six dof or less, but less efficient than He's. For parallel formulations, however, the thesis will demonstrate in the next chapter that the algorithm developed here integrates very well with the parallel inertia matrix algorithm, making it a good choice for parallel implementation.

The force vector for a branched system can be calculated if the $[C^T\ (\rho \times C)^T]$ matrix is generated using equations (3.13) and (3.15), and if the kinematic equations (3.27) and (3.28) take into account the branching.

dof	Proposed	Angeles	Walker	He
4	312×, 253+	321×, 265+	526×, 393+	267×, 248+
5	416×, 342+	426×, 355+	663×, 494+	358×, 334+
6	526×, 437+	536×, 445+	800×, 595+	449×, 420+
7	642×, 538+	636×, 535+	937×, 696+	540×, 506+
8	764×, 645+	741×, 625+	1074×, 797+	631×, 592+
9	892×, 758+	846×, 715+	1211×, 898+	722×, 678+
10	1026×, 877+	951×, 805+	1348×, 999+	813×, 764+

Table 3.3: Complexity of single cpu force vector algorithm

3.4 Equation Solving

The Cholesky decomposition linear equation solver [Walker 82] was used as the matrix solver for the single cpu case, as the inertia matrix is positive definite and symmetric. The complexity of the Cholesky algorithm, which consists of a decomposition and subsequent backsubstitution, is given by Walker as:

$$\begin{aligned} \sum \times &= n^3/6 + 3n^2/2 - 2n/3 \\ \sum + &= n^3/6 + n^2 - 7n/6 \end{aligned} \tag{3.39}$$

3.5 Simulations

In this section the simulations of two linkages are discussed. Models of a PUMA 600 six degree of freedom rotary jointed manipulator, and a six degree of freedom model of the human torso (trunk, head, and two upper and lower arms) have been implemented on single cpus. A single cpu simulation of the excavator will be presented in Chapter 5.

The PUMA 600 was chosen as an example of a rotary manipulator which is commonly discussed in the literature. In this thesis, profiles of θ, $\dot{\theta}$, and $\ddot{\theta}$ given by Angeles and Ma [Angeles 88] have been used to drive the inverse dynamics algorithm previously developed in this chapter in sec. 3.4. Although the desired profiles are relatively slow, and hence may not excite all the dynamics of the system, it was necessary to produce results which could be directly compared to those existing in the literature. The inverse dynamics then controls the manipulator model

Joint	α_i(deg)	a_i(m)	b_i(m)	Initial θ_i
1	0	0	0	0
2	-90	0	-0.149	0
3	0	0.432	0	0
4	90	0.02	-0.432	0
5	-90	0	0	0
6	90	0	-0.056	0

Table 3.4: Denavit Hartenberg parameters for the PUMA 600

in an open loop control mode. All parameters are in metric units, with mass in kg, inertia in $kg \cdot m^2$, length in m, and torque in Nm.

The human torso was chosen as an example of a branched system. Open loop inverse dynamics control was used to simulate joint profiles similar to those used for the PUMA simulation.

3.5.1 Puma 600 manipulator

A block diagram of the system appears in Fig. 3.2. The length parameters for the PUMA 600 are given in Table 3.4 (Denavit Hartenberg parameters [Angeles 1988]) and the mass and inertia parameters are given in equations (3.40) to (3.42). Fig. 3.3 is a diagram showing

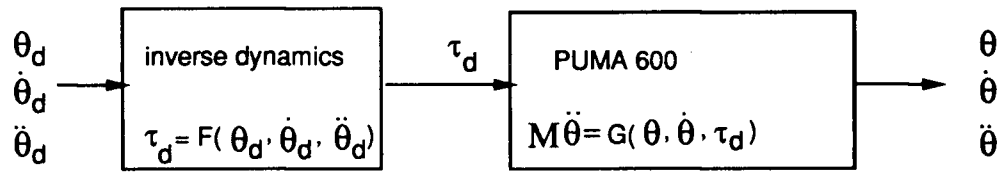

Figure 3.2: Block diagram of the open loop inverse dynamics controlled PUMA 600 dynamics simulation

the Denavit Hartenberg coordinate frames attached to the PUMA. In the coordinate system adopted in this thesis (and used in the following equations), joint i is labelled as the lower joint on link i. Because the coordinate frame is attached to the centre of gravity, the following conversions are required to generate k and l from the Denavit Hartenberg parameters and from

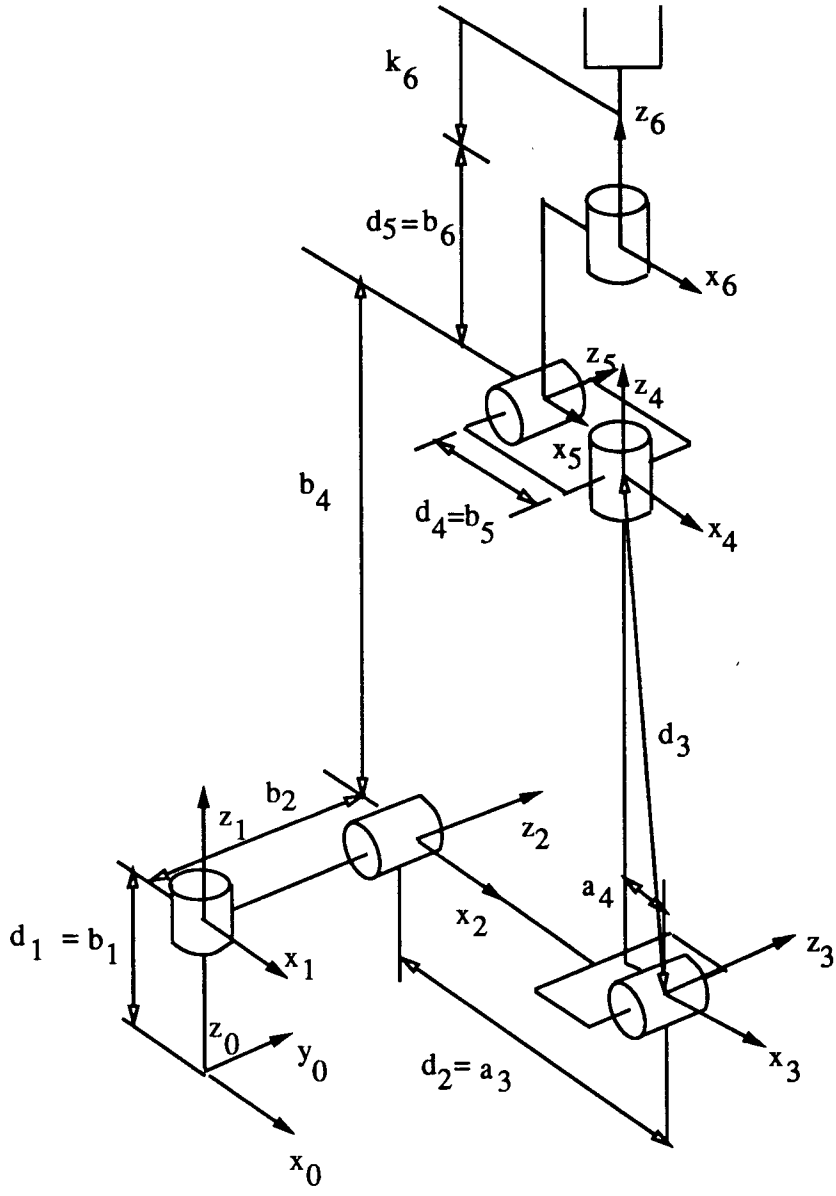

Figure 3.3: Illustration of the Denavit Hartenberg coordinate frames for the PUMA 600. Adapted from (Angeles 88)

Angeles' paper:

$$k^i_{i,i} = -\rho^i_{i,i}$$

$$
\begin{aligned}
d^i_{i+1,i} &= -\begin{bmatrix} a_{i+1} \\ b_{i+1}\sin\alpha_{i+1} \\ -b_{i+1}\cos\alpha_{i+1} \end{bmatrix} \\
l^i_{i,i+1} &= k^i_{i,i} - d^i_{i+1,i}
\end{aligned} \tag{3.40}
$$

resulting in

$$
\begin{aligned}
m_1 &= 10.521,\ k^{1T}_{1,1} = [0\ \ 0.054\ \ 0],\quad l^{1T}_{1,2} = [0\ \ 0.203\ \ 0] \\
m_2 &= 15.781,\ k^{2T}_{2,2} = [-0.14\ \ 0\ \ 0],\quad l^{2T}_{2,3} = [0.292\ \ 0\ \ 0] \\
m_3 &= 8.767,\ k^{3T}_{3,3} = [0\ \ 0.197\ \ 0],\quad l^{3T}_{3,4} = [0.02\ \ -0.235\ \ 0] \\
m_4 &= 1.052,\ k^{4T}_{4,4} = [0\ \ 0\ \ 0.057],\quad l^{4T}_{4,5} = [0\ \ 0\ \ 0.057] \\
m_5 &= 1.052,\ k^{5T}_{5,5} = [0\ \ 0.007\ \ 0],\quad l^{5T}_{5,6} = [0\ \ -0.049\ \ 0] \\
m_6 &= 0.351,\ k^{6T}_{6,6} = [0\ \ 0\ \ 0.019].
\end{aligned} \tag{3.41}
$$

and

$$
\begin{aligned}
J^1_1 &= \begin{bmatrix} 1.6120 & 0 & 0 \\ 0 & 0.5091 & 0 \\ 0 & 0 & 1.6120 \end{bmatrix}, J^2_2 = \begin{bmatrix} 0.4898 & 0 & 0 \\ 0 & 8.0783 & 0 \\ 0 & 0 & 8.2672 \end{bmatrix}, \\
J^3_3 &= \begin{bmatrix} 3.3768 & 0 & 0 \\ 0 & 0.3009 & 0 \\ 0 & 0 & 3.3768 \end{bmatrix}, J^4_4 = \begin{bmatrix} 0.1810 & 0 & 0 \\ 0 & 0.1810 & 0 \\ 0 & 0 & 0.1273 \end{bmatrix}, \\
J^5_5 &= \begin{bmatrix} 0.0735 & 0 & 0 \\ 0 & 0.0735 & 0 \\ 0 & 0 & 0.1273 \end{bmatrix}, J^6_6 = \begin{bmatrix} 0.0071 & 0 & 0 \\ 0 & 0.0071 & 0 \\ 0 & 0 & 0.0141 \end{bmatrix}.
\end{aligned} \tag{3.42}
$$

integration algorithm

The simulation used a fourth-order fixed step Runge-Kutta method described by [Press 88]. The step size was set at $h = 0.01$s, and the model was run for 10 seconds of simulation time. Desired values of θ_d, $\dot{\theta}_d$, and $\ddot{\theta}_d$ were fed into the inverse dynamics algorithm so that the desired

actuation torques τ_d were available to the PUMA model at every function evaluation (four evaluations per step h).

inverse dynamics

The $O(n^2)$ inverse dynamics algorithm developed earlier was implemented on one cpu. The desired profiles were generated according to the following equations:

$$\begin{aligned} \theta_{id}(t) &= (w_n t - \sin(w_n t))/2 \\ \dot{\theta}_{id}(t) &= (w_n - w_n \cos(w_n t))/2 \\ \ddot{\theta}_{id}(t) &= w_n^2 \sin(w_n t)/2 \qquad\qquad i = 1, ...6 \end{aligned} \tag{3.43}$$

where $w_n = 2\pi/T$, (T=10s) and t is the time variable. The torques $\tau_{id}(t)$ are calculated in correspondence with the four time points required by the integration algorithm– once at t (operation [1]), twice at $t + h/2$ ([2]and [3]), and once at $t + h$ ([4]):

$$\begin{aligned} \tau_{id}(t) &= f(\theta_d(t),\ \dot{\theta}_d(t),\ \ddot{\theta}_d(t)) && [1] \\ \tau_{id}(t + h/2) &= f(\theta_d(t + h/2),\ \dot{\theta}_d(t + h/2),\ \ddot{\theta}_d(t + h/2)) && [2,3] \\ \tau_{id}(t + h) &= f(\theta_d(t + h),\ \dot{\theta}_d(t + h),\ \ddot{\theta}_d(t + h)) && [4] \end{aligned} \tag{3.44}$$

where the τ_d are calculated using the algorithm described in sec. 3.3.1, with $\ddot{\theta}_i = \ddot{\theta}_{id}$ included in equation (3.28).

forward dynamics

The equations for the PUMA model are calculated on a separate cpu. The τ_{id} are read in at every function evaluation, and are used as the actuator torques τ_a and added to the force vector f_v to get b (f_v is calculated using sec. 3.3.1, but $\ddot{\theta}_i$ is set to zero). The simulation algorithm calculates f_v, b and $\mathcal{M}$ and then solves for $\ddot{\theta}$. The Runge-Kutta algorithm then integrates $\ddot{\theta}$ to get $\dot{\theta}$ and again to get θ. After four evaluations, the algorithm calculates a weighted response for states $\ddot{\theta}$, $\dot{\theta}$ and θ at time point $t + h$ (see Chapter 5, section 5.4.1 for an exact description).

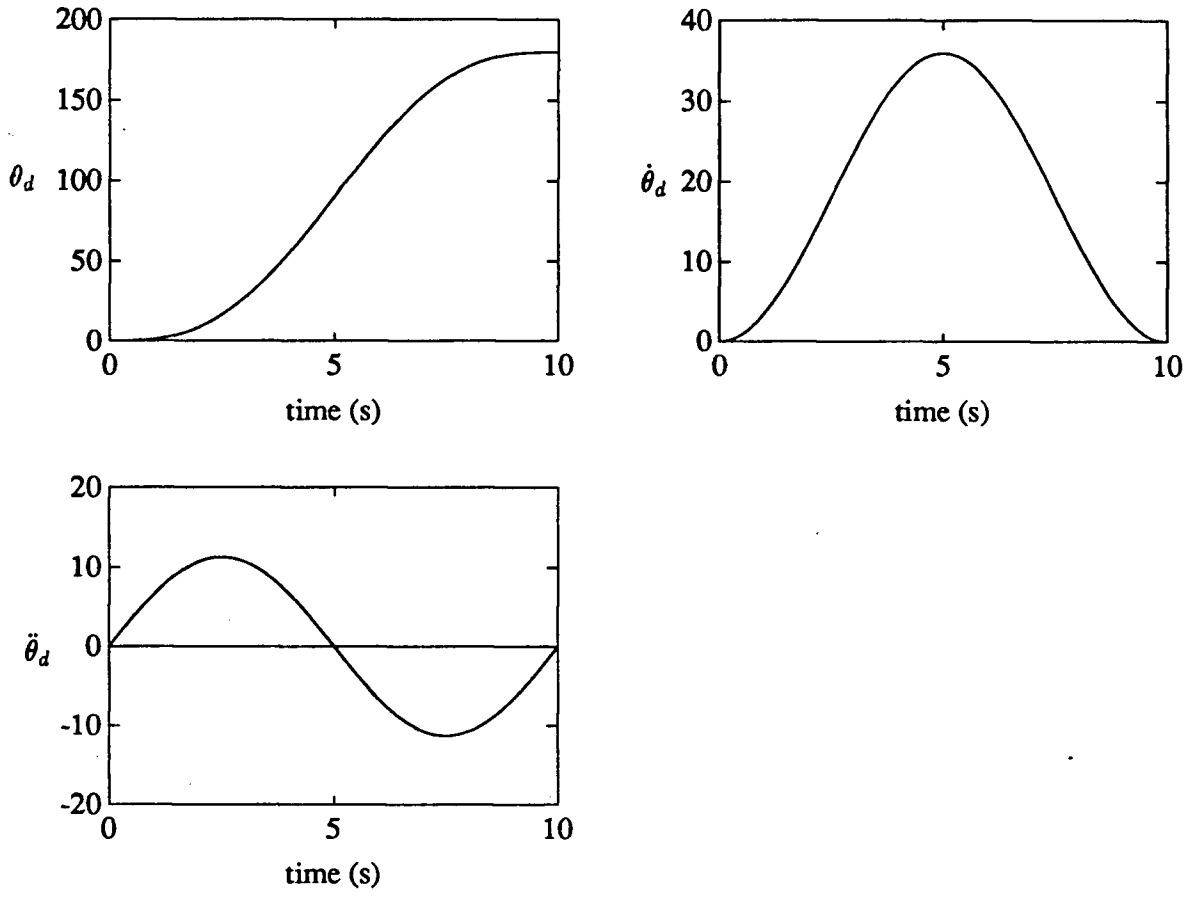

Figure 3.4: θ_d, $\dot{\theta}_d$, $\ddot{\theta}_d$ profiles for all joints of PUMA 600 model. Overlaid are the θ, $\dot{\theta}$ and $\ddot{\theta}$ responses for joint 6.

simulation results

Fig. 3.4 shows the desired θ_d, $\dot{\theta}_d$, and $\ddot{\theta}_d$, and Fig. 3.5 shows the τ_d (labelled as **tq1 to tq6**) generated by the inverse dynamics algorithm in response to the desired profiles. The simulator response for joint 6 is also shown in Fig. 3.4, and is indistinguishable from the desired values. All of the links produce a similar response when overlaid. Fig. 3.6 shows the error curves for link 6, which show maximums of -1×10^{-4} deg for θ, -5×10^{-5} deg/s

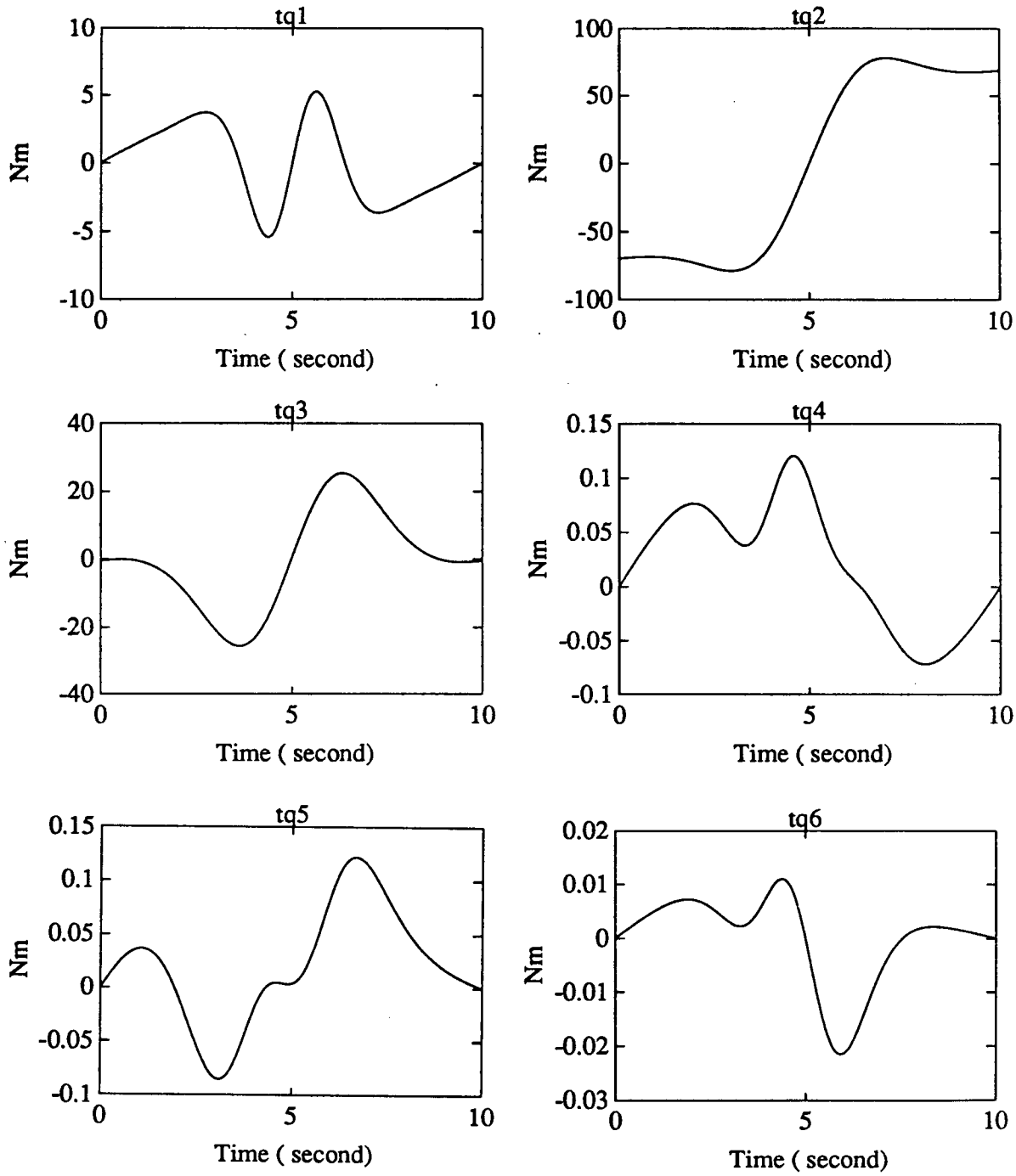

Figure 3.5: τ_d profiles for joints 1 to 6 generated by the inverse dynamics of the PUMA 600 model

for $\dot{\theta}$, and $6 \times 10^{-5}\,\mathrm{deg}/s^2$ for $\ddot{\theta}$. These results are similar to those obtained by Angeles and Ma [Angeles 88] (while their errors were similar in magnitude, their step size was larger

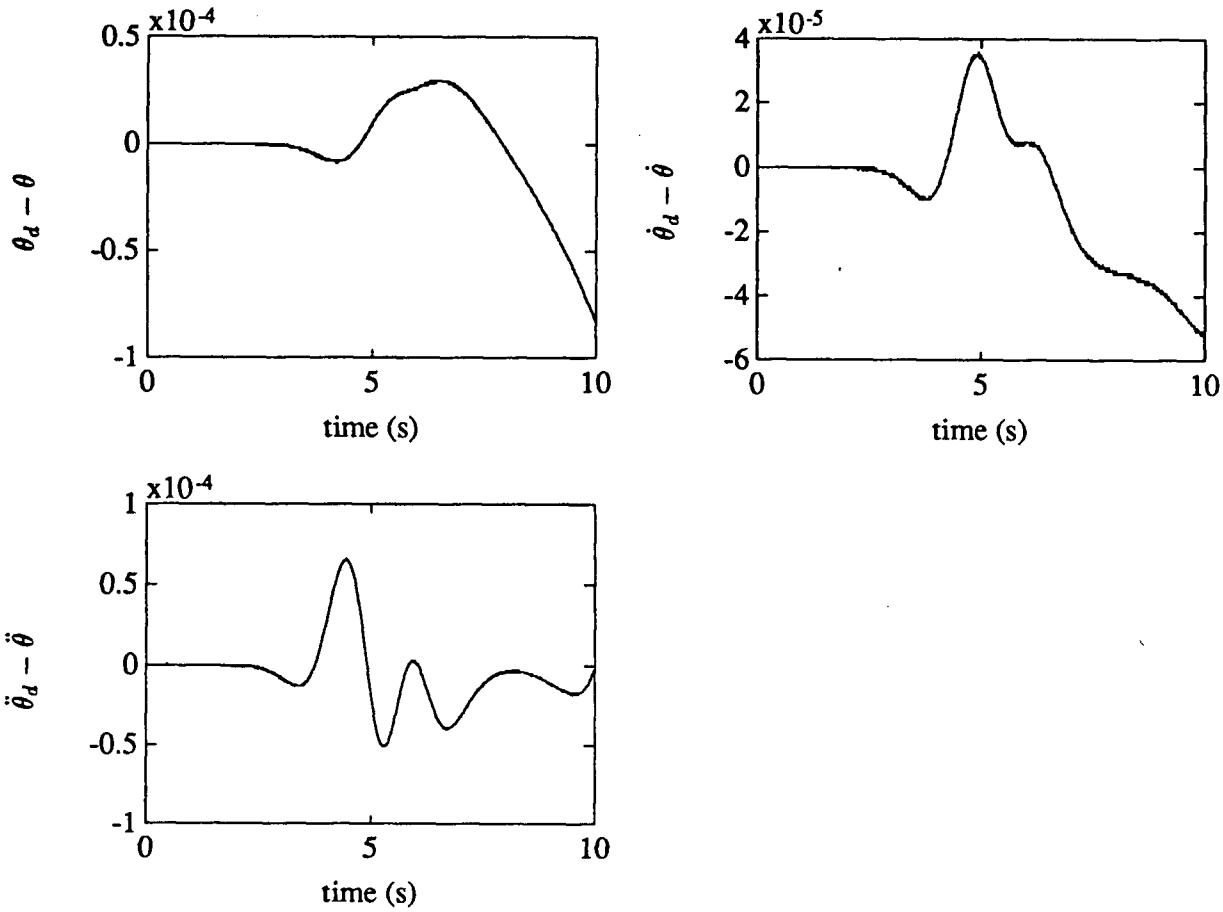

Figure 3.6: Errors $\theta_d - \theta$, $\dot{\theta}_d - \dot{\theta}$, $\ddot{\theta}_d - \ddot{\theta}$ profiles for joint 6 of PUMA 600 model

(thus requiring less computations)). Angeles' performance is slightly better (computational cost-wise) due to their using the fifth/sixth order DEVERK integration algorithm from IMSL, which calculates eight evaluations (rather than four) of the derivative before advancing one step h (this gives a better estimate).

The time taken for 10 seconds of simulation was 71.12 seconds of computer time. This time was measured assuming no results were stored and that the stack was initialised within the internal RAM of the Inmos T800 transputer (a microprocessor designed for parallel processing). The time taken for one evaluation (inertia matrix formation, force vector formation, and matrix solution) is 17.2 ms.

Theoretically, the complexity estimate from Tables 3.2, 3.3 and equation (3.39) for n=6 is

1391×, 1087+ (also shown in Table 3.5 at the end of this chapter). If a multiply and an add costs 1 μs and 0.5 μs respectively in a transputer, the theoretical computation time should have been 1.93 ms. Adding the cost of n (sin, cos) calculations at 7μs each, the total increases to 2.02 ms. This estimate would meet the maximum evaluation time of 2.5 ms when a fourth order Runge Kutta and h=0.01 are assumed.

The actual computations are thus 17.2/2.02= 8.5 times slower due to superfluous computations required for loop counters and array indexing, external memory fetches for global variables, and the cost of entering and leaving subroutines (the cost of saving data on the stack). These costs are unavoidable unless the modularity of the formulation is removed by not using indexed arrays and by replacing all subroutines with inline code.

3.5.2 Human torso

In this model of the torso, the head, trunk and left and right upper and lower arms are assumed to have one degree of freedom per joint. Fig. 3.7 describes the structure of the system, including the body coordinate systems. The mass and length parameters used by the formulation are:

$$
\begin{aligned}
m_1 &= 41.0,\ k_{1,1}^T = [0\ \ 0\ \ -0.42], & l_{1,2}^T &= [0\ \ 0\ \ 0.242] \\
& & l_{1,3}^T &= [0.15\ \ 0\ \ 0.15] \\
& & l_{1,5}^T &= [-0.15\ \ 0\ \ 0.15] \\
m_2 &= 5.0,\ k_{2,2}^T = [0\ \ 0.12\ \ 0], & & \\
m_3 &= 2.8,\ k_{3,3}^T = [0\ \ 0.2\ \ 0], & l_{3,4}^T &= [0\ \ -0.25\ \ 0] \\
m_4 &= 1.4,\ k_{4,4}^T = [0\ \ 0.2\ \ 0], & & \\
m_5 &= 2.8,\ k_{5,5}^T = [0\ \ -0.2\ \ 0], & l_{5,6}^T &= [0\ \ 0.25\ \ 0] \\
m_6 &= 1.4,\ k_{6,6}^T = [0\ \ -0.2\ \ 0], & & \qquad (3.45)
\end{aligned}
$$

J_1^1, the trunk inertia, is obtained from [Khosravi 87] while the rest of the inertia parameters are found by approximating each limb as solid cylindrical links with radius r_i and length l_i (and

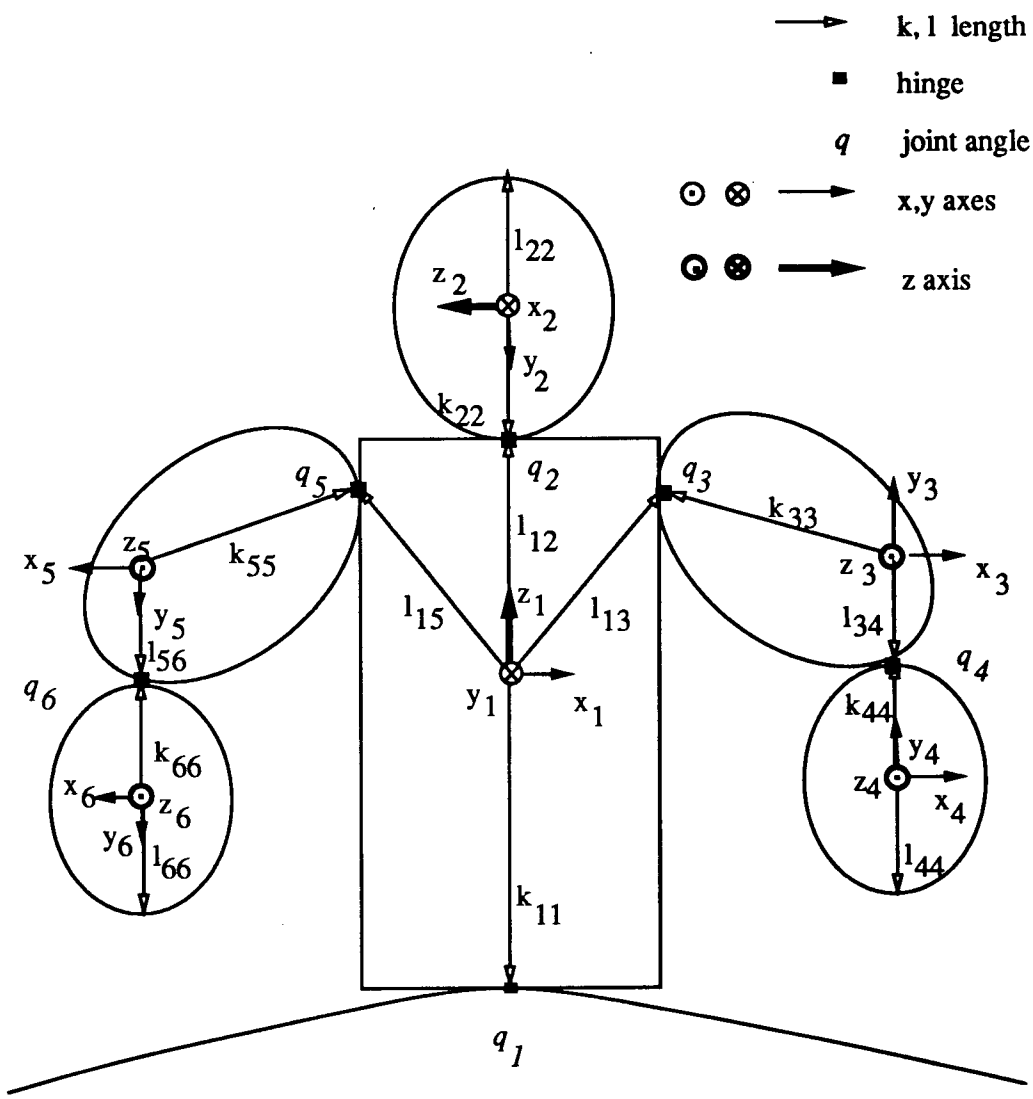

Figure 3.7: Coordinate diagram of human torso model with six single dof rotational joints

for the head, a solid sphere of radius r_2). The values are:

$$r_2 = 0.12,\ r_3 = 0.05,\ r_4 = 0.05,\ r_5 = 0.05,\ r_6 = 0.05,$$

and $$l_3 = 0.45,\ l_4 = 0.4,\ l_5 = 0.45,\ l_6 = 0.4$$

These values only approximate those given by Khosravi, since in his case the arms are modeled as one single link, as he was modeling a four link chain performing a somersault. Thus:

$$J_1^1 = \begin{bmatrix} 2.7 & 0 & 0 \\ 0 & 0.40 & 0 \\ 0 & 0 & 2.7 \end{bmatrix}, J_2^2 = \begin{bmatrix} \frac{2m_2r_2^2}{5} & 0 & 0 \\ 0 & \frac{2m_2r_2^2}{5} & 0 \\ 0 & 0 & \frac{2m_2r_2^2}{5} \end{bmatrix},$$

$$J_3^3 = \begin{bmatrix} m_3(\frac{r_3^2}{4} + \frac{l_3^2}{12}) & 0 & 0 \\ 0 & \frac{m_3r_3^2}{2} & 0 \\ 0 & 0 & m_3(\frac{r_3^2}{4} + \frac{l_3^2}{12}) \end{bmatrix}, J_4^4 = \begin{bmatrix} m_4(\frac{r_4^2}{4} + \frac{l_4^2}{12}) & 0 & 0 \\ 0 & \frac{m_4r_4^2}{2} & 0 \\ 0 & 0 & m_4(\frac{r_4^2}{4} + \frac{l_4^2}{12}) \end{bmatrix},$$

$$J_5^5 = \begin{bmatrix} \frac{m_5r_5^2}{2} & 0 & 0 \\ 0 & m_5(\frac{r_5^2}{4} + \frac{l_5^2}{12}) & 0 \\ 0 & 0 & m_5(\frac{r_5^2}{4} + \frac{l_5^2}{12}) \end{bmatrix}, J_6^6 = \begin{bmatrix} \frac{m_6r_6^2}{2} & 0 & 0 \\ 0 & m_6(\frac{r_6^2}{4} + \frac{l_6^2}{12}) & 0 \\ 0 & 0 & m_6(\frac{r_6^2}{4} + \frac{l_6^2}{12}) \end{bmatrix} \quad (3.46)$$

The twist angles (in degrees) are:

$$\alpha_1 = 0,\ \alpha_2 = -90,\ \alpha_3 = 90,\ \alpha_4 = 0,\ \alpha_5 = -90,\ \alpha_6 = 0 \quad (3.47)$$

connectivity matrix

The branched formulation makes use of the following connectivity matrix U^{-1} to describe the topology:

$$U^{-1} = \begin{bmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ 1 & 0 & 1 & & & \\ 1 & 0 & 1 & 1 & & \\ 1 & 0 & 0 & 0 & 1 & \\ 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (3.48)$$

integration, inverse dynamics and joint profiles

The integration algorithm uses the fourth order fixed step Runge-Kutta method employed for the PUMA 600 model, except that the step size was set at $h = 0.001$, as larger values

dof	Proposed	Angeles	Walker	Brandl
3	367×, 277+	399×, 306+	631×, 513+	528×, 462+
4	630×, 484+	679×, 530+	907×, 742+	778×, 682+
5	969×, 752+	1029×, 810+	1208×, 991+	1028×, 902+
6	1391×, 1087+	1456×, 1152+	1537×, 1261+	1278×, 1122+
7	1903×, 1495+	1967×, 1562+	1893×, 1553+	1528×, 1342+
8	2512×, 1982+	2569×, 2046+	2279×, 1868+	1778×, 1562+
9	3225×, 2554+	3269×, 2610+	2694×, 2207+	2028×, 1782+
10	4049×, 3217+	4074×, 3260+	3141×, 2571+	2278×, 2002+
11	4991×, 3977+	4991×, 4002+	3619×, 2961+	2528×, 2222+
12	6058×, 4840+	6027×, 4842+	4131×, 3378+	2778×, 2442+

Table 3.5: Complexity of complete forward dynamics for single chains on one cpu

caused instability due to numerical error accumulation, leading to incorrect results. The inverse dynamics algorithm is the same as that derived earlier in the chapter, but adapted to suit the branched algorithm. The joint angle, velocity and acceleration profiles are similar to those suggested by Angeles and Ma for the PUMA 600, except that the period T of the profile is 5 seconds rather than 10.

simulation results

Fig. 3.8 illustrates the θ_d, $\dot{\theta}_d$ and $\ddot{\theta}_d$ profiles applied to each of the joints. Fig. 3.9 shows the torque profiles produced by the inverse dynamics, and Fig. 3.10 shows the errors between the desired and actual angle, velocity, and acceleration profiles for joint six.

The results of the simulation of joint 6 are overlaid on top of the desired reponses in Fig. 3.8. The errors in joint 6 plotted in Fig. 3.10 indicate that the human torso model was simulated correctly although instability occurred when simulating long periods of time (approximately 5 seconds). This is due to the dynamics of the system, as the higher frequency dynamics are not adequately modeled by the integrator, due to step sizes that are too large. This problem can be overcome with smaller step sizes or more sophisticated integration algorithms.

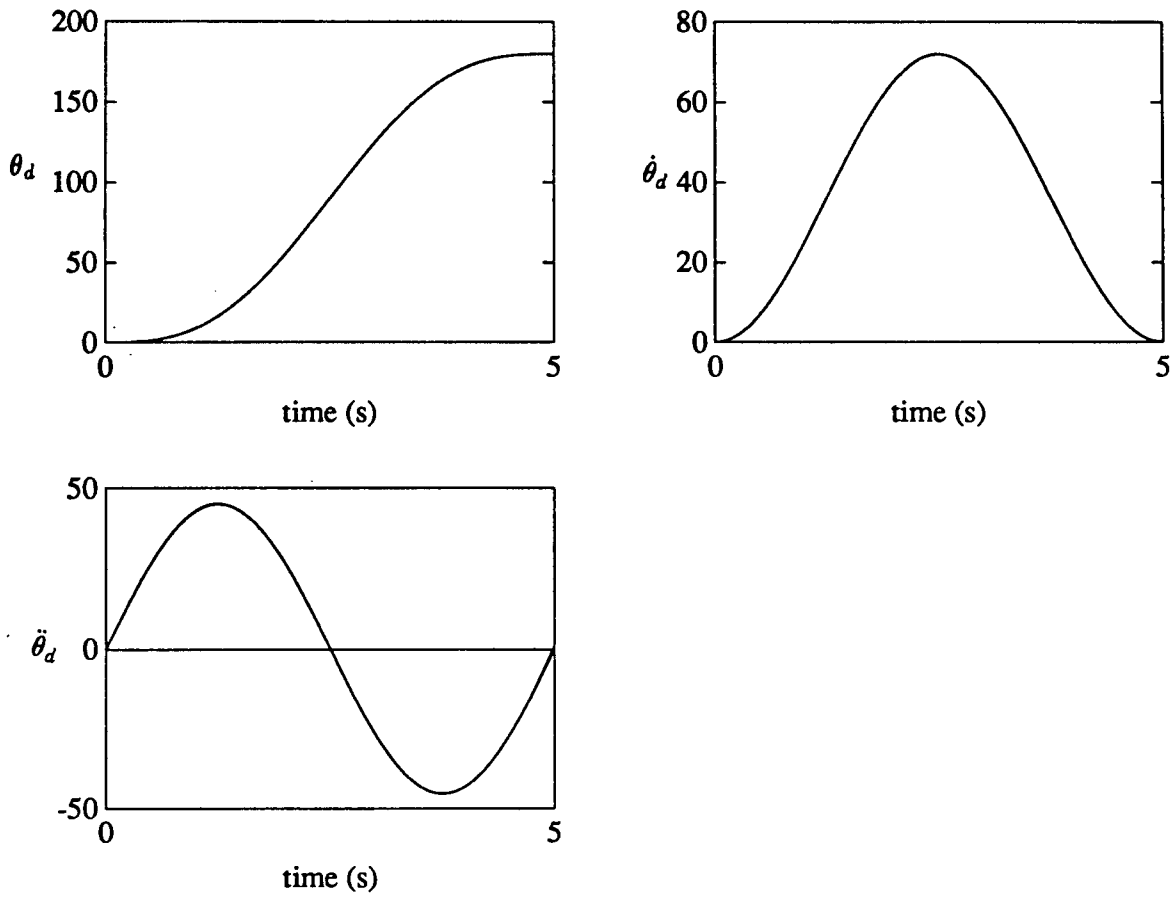

Figure 3.8: θ_d, $\dot{\theta}_d$, $\ddot{\theta}_d$ and profiles applied to all joints for human torso model. Overlaid are the reponses for joint 6.

3.6 Summary

This chapter has examined the complexity of the Newton Euler State Space algorithm for forward dynamics when calculated on one cpu. By referring all equations to body coordinates, and taking advantage of the fact that the force vector calculation uses results previously calculated from the inertia matrix calculation, efficient algorithms for the inertia matrix and the force vector have been presented. The algorithms incorporate a connectivity matrix which permits the modeling of branched mechanisms. In the case of a single chain mechanism, the transformation matrix $[C^T \ (\rho \times C)^T]$ is the same as Angeles and Ma's orthogonal complement matrix [Angeles 88]. Buchner [Buchner 86] also referred both the translational and rotational

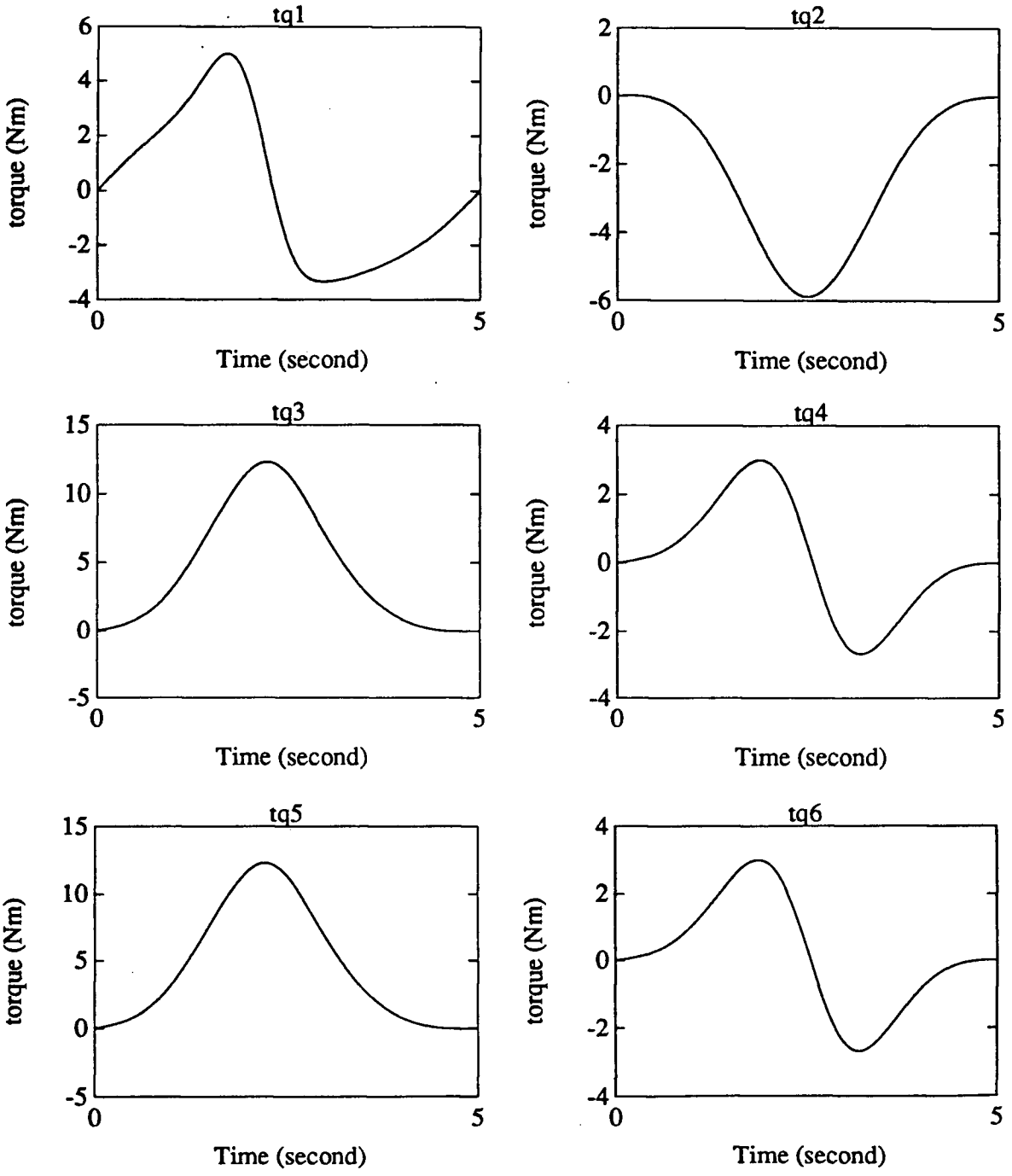

Figure 3.9: τ_d profiles for all 6 joints generated by the inverse dynamics for human torso model

equations to body coordinates, but the equations were not computed in their most efficient form. As a result the simple recursive nature of the vector cross product $(\rho^i_{i,j} \times c^i_{i,j})$ was not

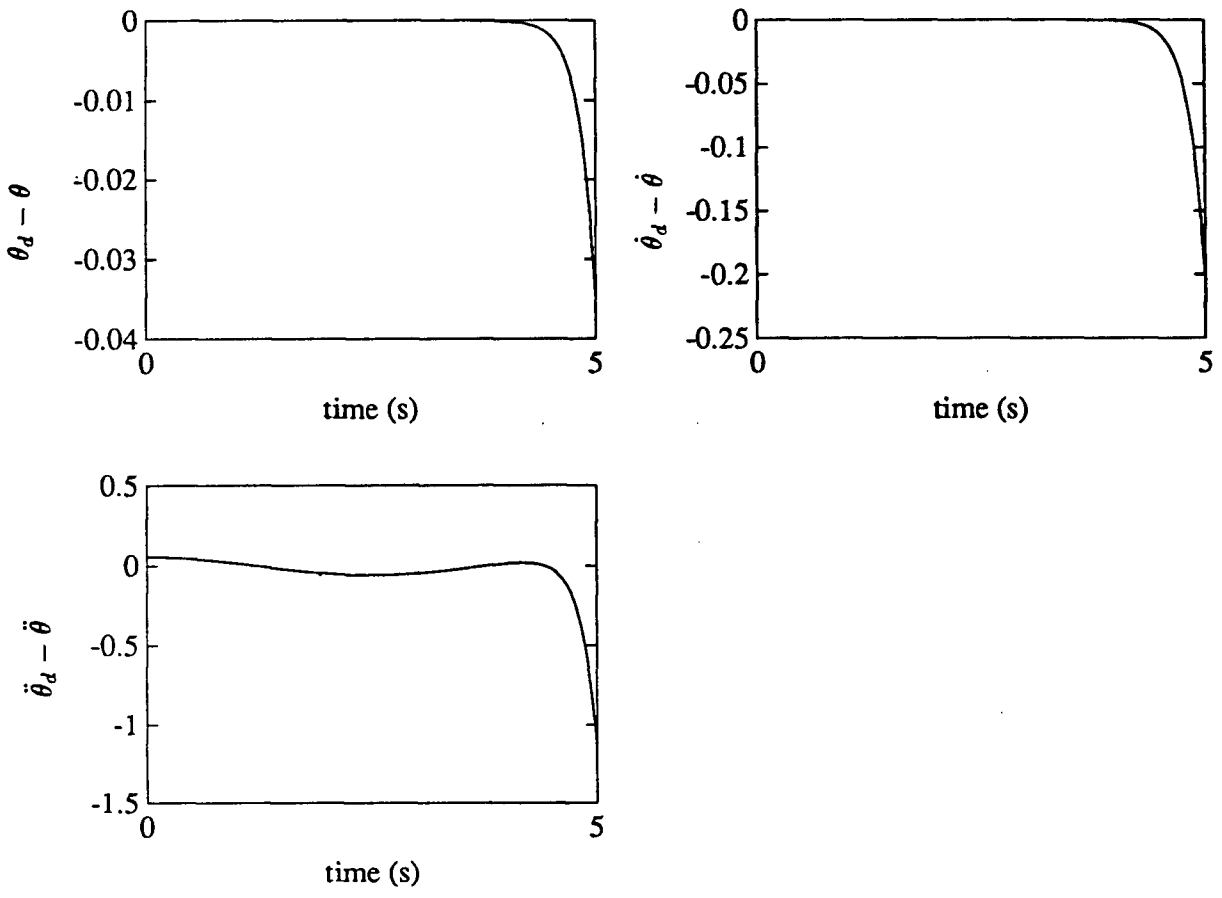

Figure 3.10: Errors $\theta_d - \theta$, $\dot{\theta}_d - \dot{\theta}$, $\ddot{\theta}_d - \ddot{\theta}$ profiles for joint 6 of human torso model

used. In addition, Buchner did not explore the representation of branched mechanisms.

The complexity estimates in Tables 3.1 and 3.5 show that when modeling low degree of freedom ($n < 6$) mechanisms on one cpu, such as an excavator or some of the robots in use today, the algorithms proposed here are of comparable (although not superior) efficiency to those with which we have compared. Table 3.5 lists the number of multiplies and adds for chains of up to twelve dof, while Fig. 3.11 graphs the results from Table 3.5, assuming the cost of an add is half that of a multiply (the complexity is thus measured in μs). Fig. 3.11 shows that the proposed algorithm (P) is very similar to Angeles' algorithms. Brandl's algorithm (B) eventually becomes more efficient due to the $O(n^3)$ complexity of the matrix solver.

In the last part of this chapter, the single chain and branched formulations were verified. The

formulations were implemented on single cpus and the PUMA 600 and a human torso were modeled. Open loop inverse dynamics simulation using joint profiles taken from the literature show good agreement with previous results.

This thesis has thus far established the equations and complexity for the Newton Euler State Space formulation when implemented on a single cpu. In the next chapter a parallel version of this formulation will be presented which is superior in computational complexity for mechanisms with relatively large numbers of links.

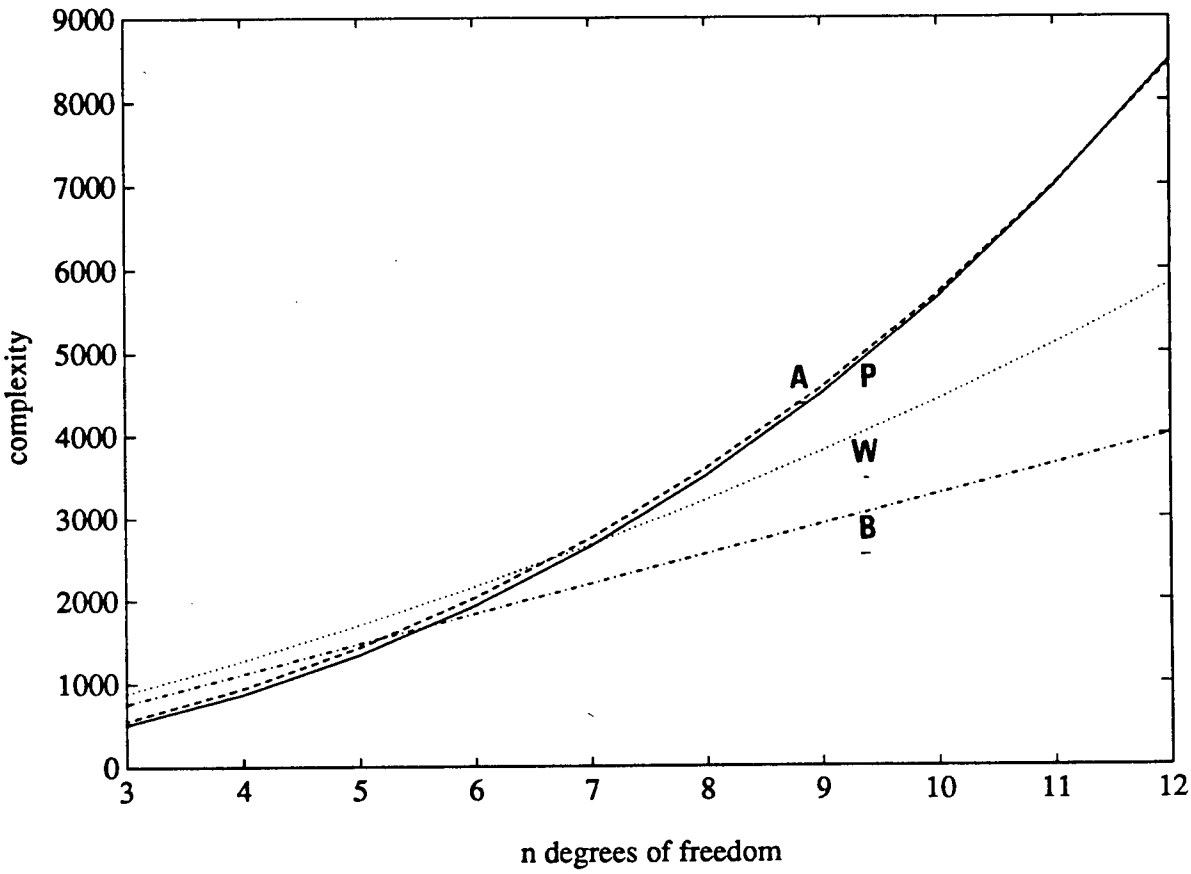

Figure 3.11: Graph of complexity estimates for formulations in Table 3.5. Algorithms are (A) - Angeles, (W) - Walker, (B) - Brandl, and (P) - proposed in thesis

Chapter 4

A Parallel Dynamics Algorithm

In this chapter the algorithms described in Chapter 3 for the inertia matrix, force vector, and solver phases are parallelised and integrated together. New parallel algorithms using a two dimensional, triangular, Multiple Input Multiple Data (MIMD) cpu array (Fig. 4.1) with a systolic/wavefront approach have been developed for the inertia matrix and the force vector calculations. The first row in the array, row 0, calculates the force vector b (gravity, Coriolis, centrifugal and actuator torques). Simultaneously, rows 1 through n calculate the inertia matrix. The leftmost cpu, $\text{cpu}_{0,0}$, connects to the host computer.

Once this is accomplished, b_i resides in row 0 $\text{cpu}_{0,i}$ $(i = 1,..n)$, and the inertia matrix element $\mathcal{M}_{i,j} = \mathcal{M}_{j,i}$ resides in $\text{cpu}_{i,j}$. The data is then shuffled to the edge of the array, and the equation $\mathcal{M}\ddot{\theta} = b$ is solved using a new feedforward systolic solver derived by [Jainandunsing 89] which is more efficient than previous algorithms.

4.1 Integrating the Inertia Matrix and Force Vector Computations

The equations of motion derived in the previous chapter can be stated as

$$[\bar{H}^T H]\ddot{\theta} = \bar{H}^T \begin{bmatrix} \dot{L} \\ m\ddot{p} \end{bmatrix} + \tau_a = f_v + \tau_a \tag{4.1}$$

$$\text{or} \quad \mathcal{M}\ddot{\theta} = b \tag{4.2}$$

where τ_a is the vector of actuator torques. The calculation of both the inertia matrix $\mathcal{M}$ and f_v (τ_a is calculated separately) can be divided into two phases:

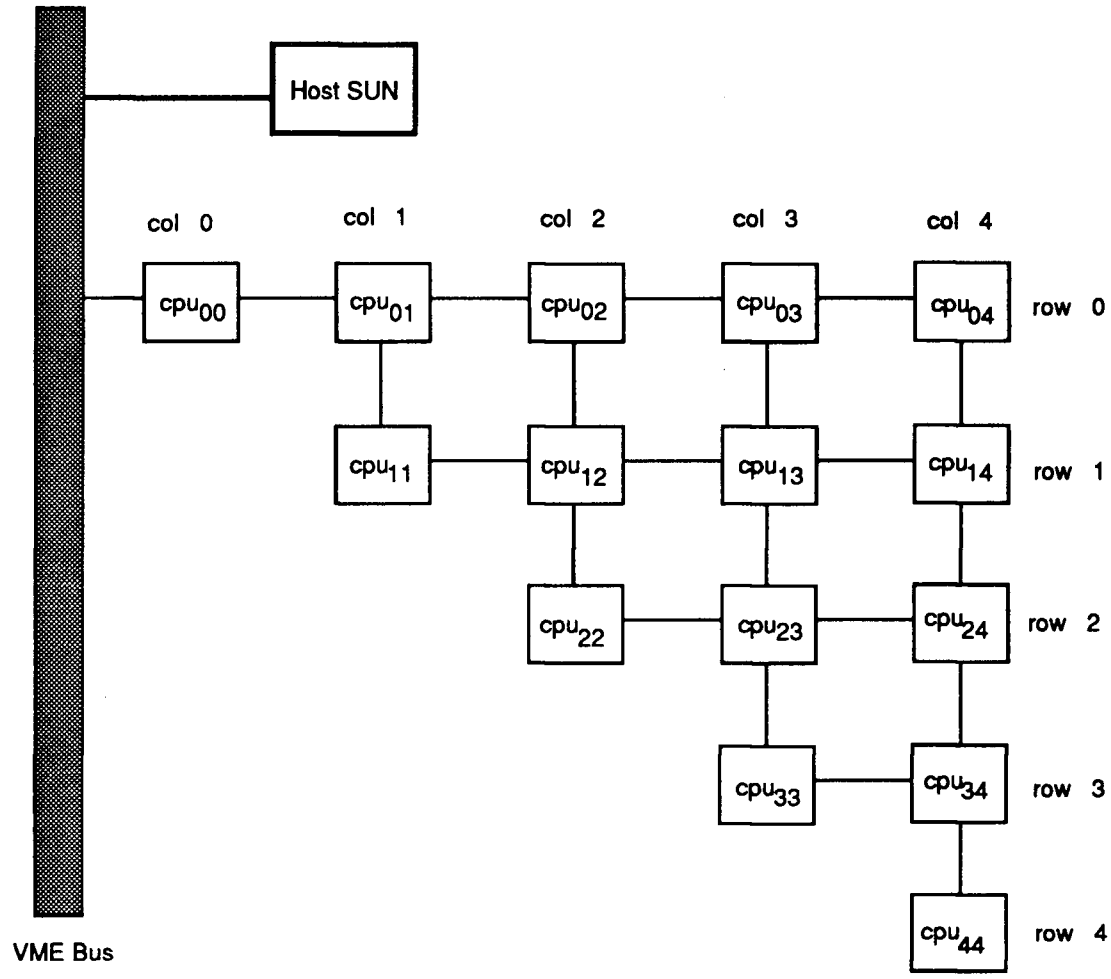

Figure 4.1: The triangular 2d mesh array used to implement the dynamics algorithm

1. Calculating the $6n \times n$ matrices $\bar{H}$ and H ($\bar{H}$ is a component of H), and the $6n \times 1$ momentum derivative vector $[\dot{L}\ m\ddot{p}]^T$. Row 0 of the array in Fig. 4.1 can be used to calculate the derivative vector, while simultaneously rows 1 to n are used to calculate $\bar{H}^T$ and H.

2. Multiplying H and $[\dot{L}\ m\ddot{p}]^T$ by $\bar{H}^T$.

In this chapter the $\mathcal{M}$ and f_v calculations are parallelised so that the computations occur in a balanced manner, where both the computations of H and $[\dot{L}\ m\ddot{p}]$ finish simultaneously.

This is necessary so that phase 2 can proceed in a synchronised manner. Following this, the array is reused to solve the matrix equation. This is easy to implement using the homogeneous architecture described here, in comparison to the architectures of [Lee 88] and [Fijany 89], which require different architectures for the equation formation and for the solver.

4.2 Parallel Calculation of H

The calculation of H requires the recursive evaluation of equation (3.17):

$$H = \begin{bmatrix} JC \\ M(\rho \times C) \end{bmatrix} \tag{4.3}$$

The architecture in Fig. 4.1 can be used to compute equation (4.3) in parallel. The following notes are descriptions of the calculations carried out using this architecture:

calculate A^i_{i-1}

In each cpu in row 1 (Fig. 4.2), $\text{cpu}_{1,i}$, the rotation matrix A^i_{i-1} is calculated from equation (3.18) (operation 1 in Table 4.1). The θ_i are assumed to be resident. No communication is necessary, and the cost of computing sine and cosine functions (1 of each) are excluded from Table 4.1. Note that the Inmos T800 transputer processor can compute the sine and cosine functions, and thus it is not necessary for the host computer to send these values (assumed by [Lee 88]).

calculate $c^i_{i,j}$ and $\rho^i_{i,j}$

Calculation of $c^i_{i,j}$ and $\rho^i_{i,j}$ (equations 3.19 and 3.20) proceeds using row 1 as a systolic pipeline, in which $c^i_{i,j}$ and then $\rho^i_{i,j}$ $j = 1, ...i$ (operations 2 and 3 in Table 4.1) are alternately calculated in $\text{cpu}_{1,i}$ (Fig 4.2). Once a $c^i_{i,j}$ vector is calculated, it is passed eastward (e) for the calculation of $c^{i+1}_{i+1,j}$ by $\text{cpu}_{1,i+1}$ according to equation (3.19). The vector is also sent southward (s) to the lower rows in the mesh in preparation for operations 4 and 5. The communication of $c^i_{i,j}$ (send

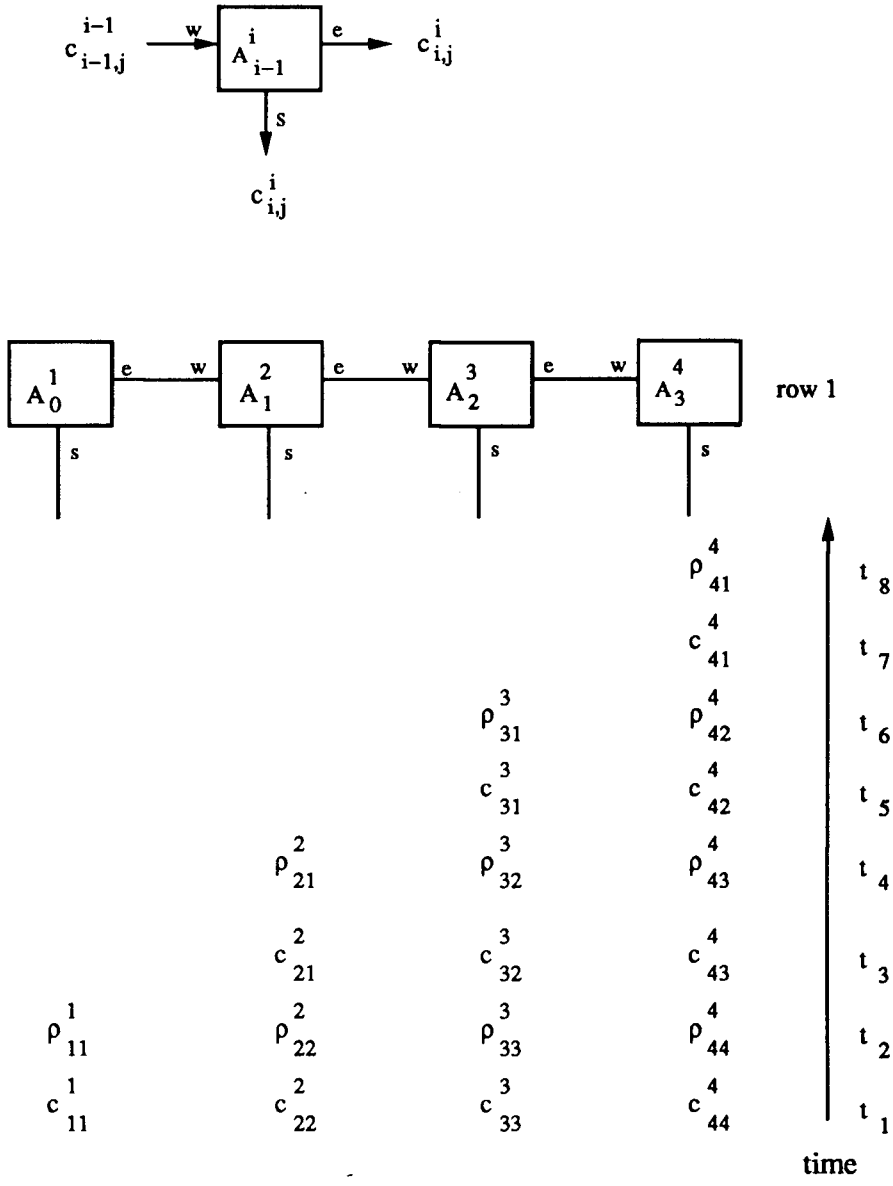

Figure 4.2: Pipeline computation of A^i_{i-1}, $c^i_{i,j}$, and $\rho^i_{i,j}$ in $\text{cpu}_{1,i}$ (row 1)

east and south; receive west) and the computation of $\rho^i_{i,j}$ is assumed to be simultaneous using the Inmos T800 transputer or the TMS320C40 as the cpu, as they both have DMA controllers for each communication link, enabling simultaneous communication and computation. Note that complete overlap is difficult to achieve with current transputer technology (the cpu chosen

for implementing these algorithms) as the amount of data to be transmitted is small, resulting in a communication setup time that becomes relatively significant. Theoretically, however, it has been assumed here that a complete overlap between the multiple communications and the computation is achievable. The time taken to send a vector of three 32-bit floating point numbers is about 12 microseconds. A matrix-vector product or cross product is considered here to be the smallest atom of computation, since the cost is about 12 microseconds (and therefore the computation can overlap the communication of a 3×1 vector).

Fig. 4.3 shows the final distribution of elements in the part of the array used to calculate $\mathcal{M}$ (rows 1 to 4), and Table 4.2 is the schedule for operations 1 to 5 in Table 4.1, for a four link chain. Operations 1 to 11 in Table 4.2 represent atomic operation cycles.

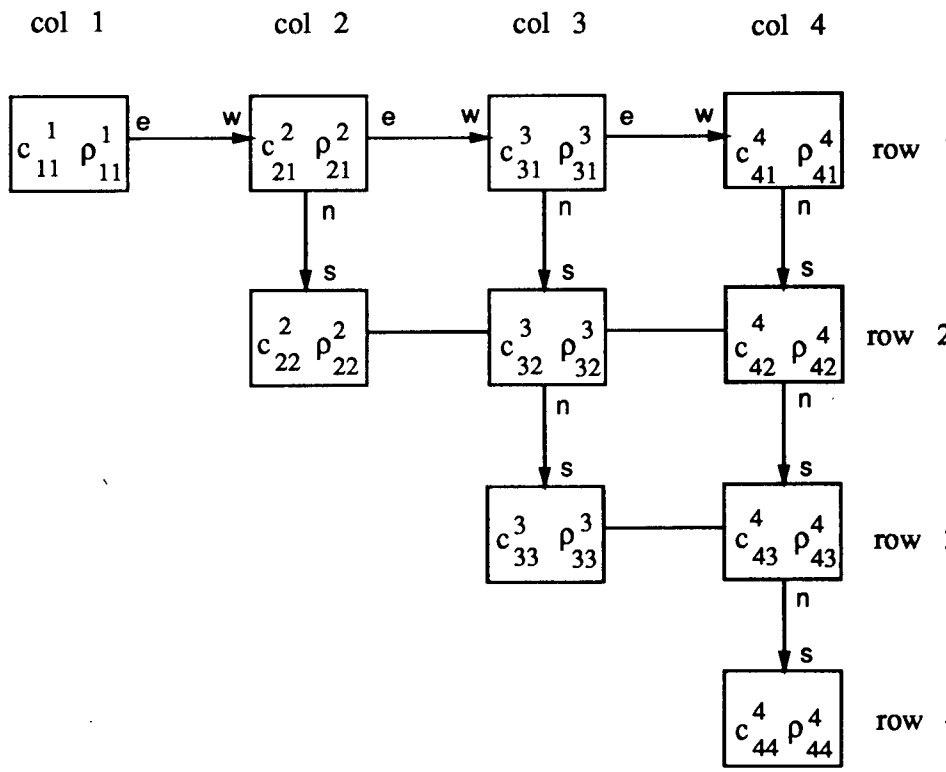

Figure 4.3: Final distribution of $c^i_{i,j}$ and $\rho^i_{i,j}$ vectors

#	operation	cost	comms no. floats	cpus
1	A^i_{i-1}	4×	0	n
2	$c^i_{i,j}$	(n - 1)(8×, 5+)	3n	$\frac{n(n+1)}{2}$
3	$\rho^i_{i,j}$	(n - 1)(8×, 11+)	3n	$\frac{n(n+1)}{2}$
4	$J^i_i c^i_{i,j}$	9×, 6+	0	$\frac{n(n+1)}{2}$
5	$m_i(\rho^i_{i,j} \times c^i_{i,j})$	9×, 3+	0	$\frac{n(n+1)}{2}$
6	$c^i_{i,j}(J^i_i c^i_{i,k})$	n(3×, 2+)	0	$\frac{n(n+1)}{2}$
7	$\rho^j_{j,k} \times c^j_{j,k}$	n(6×, 3+)	0	$\frac{n(n+1)}{2}$
8	$(\rho \times C)^T M(\rho \times C)$	n(3×, 2+)	3n	$\frac{n(n+1)}{2}$
9	$\sum$	n(2+)	n	$\frac{n(n+1)}{2}$

Table 4.1: Computational complexity for parallel H algorithm

cycle	cpu_{11}	cpu_{12}	cpu_{13}	cpu_{14}	cost to cpu_{14}
1	A^1_0	A^2_1	A^3_2	A^4_3	4×
2	$c^1_{1,1}$	$c^2_{2,2}$	$c^3_{3,3}$	$c^4_{4,4}$	0×, 0+
3	$\rho^1_{1,1}$	$\rho^2_{2,2}$	$\rho^3_{3,3}$	$\rho^4_{4,4}$	0×, 0+
4	$J^1_1 c^1_{1,1}$	$c^2_{2,1}$	$c^3_{3,2}$	$c^4_{4,3}$	8×, 5+
5	$m_1(\rho^1_{1,1} \times c^1_{1,1})$	$\rho^2_{2,1}$	$\rho^3_{3,2}$	$\rho^4_{4,3}$	8×, 11+
6		$J^2_2 c^2_{2,1}$	$c^3_{3,1}$	$c^4_{4,2}$	8×, 5+
7		$m_2(\rho^2_{2,1} \times c^2_{2,1})$	$\rho^3_{3,1}$	$\rho^4_{4,2}$	8×, 11+
8			$J^3_3 c^3_{3,1}$	$c^4_{4,1}$	8×, 5+
9			$m_3(\rho^3_{3,1} \times c^3_{3,1})$	$\rho^4_{4,1}$	8×, 11+
10				$J^4_4 c^4_{4,1}$	9×, 6+
11				$m_4(\rho^4_{4,1} \times c^4_{4,1})$	9×, 6+

Table 4.2: H computation schedule for row 1 cpus

calculate $J^i_i c^i_{i,j}$ and $m_i(\rho^i_{i,j} \times c^i_{i,j})$

All cpus $\text{cpu}_{j,i}$ in rows 1 through n calculate equations (3.21) and (3.22) with no communications between the cpus (operations 4 and 5 in Table 4.1, and also the last two operations in each column of Table 4.2).

The computational complexity of operations 1 to 5 in Table 4.1 is

$$\begin{aligned} \sum_{H=1}^{5} &= (n-1)(16\times, 16+) + (22\times, 9+) \\ \sum_{H=1}^{5} &= (16n+6)\times,\ (16n-7)+ \end{aligned} \tag{4.4}$$

where the computations and communications of $c^i_{i,j}$ and $\rho^i_{i,j}$ synchronise the cycle of computation, and all communications have been overlapped by computations.

4.3 Parallel Calculation of $[\dot{L}\ m\ddot{p}]$

In Chapter 3 an $O(n^2)$ algorithm was presented for calculating the vector f_v of gravitational, centrifugal and Coriolis forces. In this section an $O(n)$ parallel algorithm for f_v is developed which integrates well with the parallel H algorithm just presented. The calculation of τ_a, the actuator torques, will not be be discussed until Chapter 5, since they are dependent on the type of drive mechanism employed, and can be considered as a separate computational problem.

From equation (3.34), the generation of f_v reduces to a problem of premultiplying the vector $[\dot{L}\ m\ddot{p}]$ by $\bar{H}$. In this section we calculate $[\dot{L}\ m\ddot{p}]$ in a parallel manner.

Equations (3.27) to (3.31) can be calculated by row 0 in a systolic pipeline. The calculation of $w^i_{i,0}$, $\dot{w}^i_{i,0}$, $W_i l_i$ and $\ddot{p}_i$, computed in $\text{cpu}_{0,i}$, requires $w^{i-1}_{i-1,0}$, $\dot{w}^{i-1}_{i-1,0}$, $W_{i-1}l_{i-1}$, and $\ddot{p}_{i-1}$ to be sent from the left neighbour $\text{cpu}_{0,i-1}$ (l_i, k_i, and p_i are simplified forms of $l^i_{i,i+1}$, $k^i_{i,i}$ and $p^i_{i,0}$ used to simplify the notation in Table 4.3). Note that $\dot{L}_i = b_i + w^i_{i,0} \times a_i$, where $a_i = J^i_i w^i_{i,0}$ and $b_i = J^i_i \dot{w}^i_{i,0}$ (here b_i is a dummy variable, and is not related to equation 4.2).

From Table 4.3, the computational complexity is:

$$\sum_{\dot{L},m\ddot{p}} = (n-1)(18\times, 13+) + (63\times, 50+) = (18n+45)\times, (13n+37)+ \tag{4.5}$$

The computational cost of $w^i_{i,0}$ and $\dot{w}^i_{i,0}$, $(n-1)(18\times, 13+)$, synchronises the computation cycle and dictates the complexity of the calculation. Although $w^{i+1}_{i+1,0}$ is calculated two computational atoms after $w^i_{i,0}$ (due to the one atom delay in sending the $w^i_{i,0}$ vector to $\text{cpu}_{0,i+1}$), the cost of sending $w^i_{i,0}$ to $\text{cpu}_{0,i+1}$ is hidden by the computation of $\dot{w}^i_{i,0}$ in $\text{cpu}_{0,i}$.

4.4 Parallel Multiplication by $\bar{H}^T$

Once the schedules in Tables 4.2 and 4.3 have been completed, both the vector $[\dot{L}\ m\ddot{p}]$ and the rectangular matrix H must be multiplied by $\bar{H}^T$.

cycle	cost to cpu_{01}	cpu_{01}	cost to cpu_{02}	cpu_{02}	cpu_{03}	cpu_{04}
1	$4\times$	A_0^1	$4\times$	A_1^2	A_2^3	A_3^4
2	$0\times, 0+$	$w_{1,0}^1$				
3	$0\times, 0+$	$\dot{w}_{1,0}^1$				
4	$1\times$	W_1	$8\times, 6+$	$w_{2,0}^2$		
5	$9\times, 6+$	$-W_1 k_1$	$10\times, 7+$	$\dot{w}_{2,0}^2$		
6	$9\times, 6+$	$W_1 l_1$	$6\times, 9+$	W_2	$w_{3,0}^3$	
7	$8\times, 11+$	$\ddot{p}_1$	$9\times, 6+$	$-W_2 k_2$	$\dot{w}_{3,0}^3$	
8	$9\times, 6+$	$a_1 = J_1^1 w_{1,0}^1$	$9\times, 6+$	$W_2 l_2$	W_3	$w_{4,0}^4$
9	$9\times, 6+$	$b_1 = J_1^1 \dot{w}_{1,0}^1$	$8\times, 11+$	$\ddot{p}_2$	$-W_3 k_3$	$\dot{w}_{4,0}^4$
10	$9\times, 6+$	$\dot{L}_1, m_1 \ddot{p}_1$	$9\times, 6+$	$a_2 = J_2^2 w_{2,0}^2$	$W_3 l_3$	W_4
11			$9\times, 6+$	$b_2 = J_2^2 \dot{w}_{2,0}^2$	$\ddot{p}_3$	$-W_4 k_4$
12			$9\times, 6+$	$\dot{L}_2, m_2 \ddot{p}_2$	$a_3 = J_3^3 w_{3,0}^3$	$W_4 l_4$
13					$b_3 = J_3^3 \dot{w}_{3,0}^3$	$\ddot{p}_4$
14					$\dot{L}_3, m_3 \ddot{p}_3$	$a_4 = J_4^4 w_{4,0}^4$
15						$b_4 = J_4^4 \dot{w}_{4,0}^4$
16						$\dot{L}_4, m_4 \ddot{p}_4$

Table 4.3: Schedule for calculating $[\dot{L}\ m\ddot{p}]$

The atoms of computation which determine the computation cycle in the inertia matrix algorithm, the $c_{i,j}^i$, $\rho_{i,j}^i$ computations $(16\times, 16+)$ are comparable to their counterparts in the force algorithm, the $(18\times, 13+)$ w_i and $\dot{w}_i$ computations. The constant overhead for calculating $[\dot{L}\ m\ddot{p}]$ $(63\times, 50+)$, however, is significantly larger than the overhead for H $(22\times, 9+)$. This results in idle time in the inertia matrix cpu array, where the $a_{i,j}$ cpus wait for the b_i cpus until both algorithms can proceed with the synchronised multiplication by $\bar{H}^T$.

Minimisation of this idle time can be accomplished by passing some of the $[\dot{L}\ m\ddot{p}]$ calculations from row 0 to row 1. $w_{i,0}^i$ and $\dot{w}_{i,0}^i$ can be sent from row 0 to row 1 cpus, where $a_i = J_i^i w_{i,0}^i$ and $b_i = J_i^i \dot{w}_{i,0}^i$ $(18\times, 12+)$ can be calculated, relieving row 0 cpus of this burden. The results are then sent back to row 0. Although this requires the communication of vectors a_i and b_i from row 1 to row 0, it is possible to hide this communication cost with computations. Table 4.4 presents a schedule of some of the communications and computations between cpu_{03}, cpu_{04} and cpu_{14} (// indicates a parallel process, tx() and rx() indicates transmission and reception). The table shows that all communications are hidden by computations except in the case of $\text{cpu}_{0,4}$. Tables 4.5 and 4.6 are the revised versions of Tables 4.2 and 4.3 (excluding communications)

cpu_{03}	cpu_{04}	cpu_{14}	
$\rightarrow \text{rx}(\ddot{p}_2)//W_3 l_3$	$W_4//\text{rx}(\rho^4_{4,1}) \leftarrow$	$\leftarrow \text{tx}(\rho^4_{4,1})//J^4_4 c^4_{4,1}$	10
$\ddot{p}_3//\text{tx}(W_3 l_3) \rightarrow$	$\rightarrow \text{rx}(W_3 l_3)//-W_4 k_4//\text{tx}(w^4_{4,0}) \rightarrow$	$\rightarrow \text{rx}(w^4_{4,0})//m_4(\rho^4_{4,1} \times c^4_{4,1})$	11
$c_3 = a_3 \times w^3_{3,0}, m_3 \ddot{p}_3//\text{tx}(\ddot{p}_3) \rightarrow$	$\rightarrow \text{rx}(\ddot{p}_3)//W_4 l_4//\text{tx}(\dot{w}^4_{4,0}) \rightarrow$	$\rightarrow \text{rx}(\dot{w}^4_{4,0})//a_4 = J^4_4 w^4_{4,0}$	12
$\dot{L}_3 = c_3 + b_3$	$\ddot{p}_4//\text{rx}(a_4) \leftarrow$	$\leftarrow \text{tx}(a_4)//b_4 = J^4_4 \dot{w}^4_{4,0}$	13
	$c_4 = a_4 \times w^4_{4,0}, m_4 \ddot{p}_4//\text{rx}(b_4) \leftarrow$	$\leftarrow \text{tx}(b_4)$	14
	$\dot{L}_4 = c_4 + b_4$		15

Table 4.4: Redistributed computations and communications for $[\dot{L}_4 \; m_4 \ddot{p}_4]$

	cpu_{11}	cpu_{12}	cpu_{13}	cpu_{14}	cost to cpu_{14}
1	A^1_0	A^2_1	A^3_2	A^4_3	$4\times$
2	$c^1_{1,1}$	$c^2_{2,2}$	$c^3_{3,3}$	$c^4_{4,4}$	$0\times, 0+$
3	$\rho^1_{1,1}$	$\rho^2_{2,2}$	$\rho^3_{3,3}$	$\rho^4_{4,4}$	$0\times, 0+$
4	$J^1_1 c^1_{1,1}$	$c^2_{2,1}$	$c^3_{3,2}$	$c^4_{4,3}$	$8\times, 5+$
5	$m_1(\rho^1_{1,1} \times c^1_{1,1})$	$\rho^2_{2,1}$	$\rho^3_{3,2}$	$\rho^4_{4,3}$	$8\times, 11+$
6	$J^1_1 w^1_{1,0}$	$J^2_2 c^2_{2,1}$	$c^3_{3,1}$	$c^4_{4,2}$	$8\times, 5+$
7	$J^1_1 \dot{w}^1_{1,0}$	$m_2(\rho^2_{2,1} \times c^2_{2,1})$	$\rho^3_{3,1}$	$\rho^4_{4,2}$	$8\times, 11+$
8		$J^2_2 w^2_{2,0}$	$J^3_3 c^3_{3,1}$	$c^4_{4,1}$	$8\times, 5+$
9		$J^2_2 \dot{w}^2_{2,0}$	$m_3(\rho^3_{3,1} \times c^3_{3,1})$	$\rho^4_{4,1}$	$8\times, 11+$
10			$J^3_3 w^3_{3,0}$	$J^4_4 c^4_{4,1}$	$9\times, 6+$
11			$J^3_3 \dot{w}^3_{3,0}$	$m_4(\rho^4_{4,1} \times c^4_{4,1})$	$9\times, 6+$
12				$J^4_4 w^4_{4,0}$	$9\times, 6+$
13				$J^4_4 \dot{w}^4_{4,0}$	$9\times, 6+$

Table 4.5: Revised H computation (row 1) schedule

when a_i and b_i are calculated in row 1.

The complexity for the revised H and $[\dot{L} \;\; m\ddot{p}]$ algorithms prior to multiplying by $\bar{H}$ is

$$\sum_H = (n-1)(16\times, 16+) + (40\times, 21+) \tag{4.6}$$

for the H matrix, and

$$\sum_{\dot{L}, m\ddot{p}} = (n-1)(18\times, 13+) + (45\times, 38+) \tag{4.7}$$

for the $[\dot{L} \;\; m\ddot{p}]$ calculation. The H algorithm in Table 4.5 requires each row 1 cpu to transmit $b_i = J^i_i \dot{w}^i_{i,0}$ to their counterpart cpu in row 0 after all computations have ended (operation 14 in Table 4.4).

	cost to cpu_{01}	cpu_{01}	cost to cpu_{02}	cpu_{02}	cpu_{03}	cpu_{04}
1	4×	A_0^1	4×	A_1^2	A_2^3	A_3^4
2	0×,0+	$w_{1,0}^1$				
3	0×,0+	$\dot{w}_{1,0}^1$				
4	1×	W_1	8×,6+	$w_{2,0}^2$		
5	9×,6+	$-W_1k_1$	10×,7+	$\dot{w}_{2,0}^2$		
6	9×,6+	W_1l_1	6×,9+	W_2	$w_{3,0}^3$	
7	8×,11+	$\ddot{p}_1$	9×,6+	$-W_2k_2$	$\dot{w}_{3,0}^3$	
8	9×,3+	$c_1, m_1\ddot{p}_1$	9×,6+	W_2l_2	W_3	$w_{4,0}^4$
9	3+	$\dot{L}_1 = c_1 + b_1$	8×,11+	$\ddot{p}_2$	$-W_3k_3$	$\dot{w}_{4,0}^4$
10			9×,3+	$c_2, m_2\ddot{p}_2$	W_3l_3	W_4
11			3+	$\dot{L}_2 = c_2 + b_2$	$\ddot{p}_3$	$-W_4k_4$
12					$c_3, m_3\ddot{p}_3$	W_4l_4
13					$\dot{L}_3 = c_3 + b_3$	$\ddot{p}_4$
14						$c_4, m_4\ddot{p}_4$
15						$\dot{L}_4 = c_4 + b_4$

Table 4.6: Revised schedule for calculating $[\dot{L}\ m\ddot{p}]$

4.4.1 Multiplying H and $[\dot{L}\ m\ddot{p}]$ by $\bar{H}^T$

By reorganising the computations as discussed above, the momentum derivative vector $[\dot{L}\ m\ddot{p}]$ and the H matrix are completed at approximately the same time. $\bar{H}^T = [C\ \ (\rho \times C)]^T$ can now be multiplied simultaneously with H and $[\dot{L}\ m\ddot{p}]$. The rectangular matrix JC and the vector $\dot{L}$ are premultipied with C^T, and the matrix $M(\rho \times C)$ and vector $m\ddot{p}$ are premultiplied with $(\rho \times C)^T$.

For the inertia matrix calculations, the elements $c_{i,j}^i$, $(\rho_{i,j}^i \times c_{i,j}^i)$, $J_i^i c_{i,j}^i$ and $m_i(\rho_{i,j}^i \times c_{i,j}^i)$ are already in $cpu_{i,j}$ (Fig. 4.3) and so there is no loading phase. The $\bar{H}^T H$ multiplication can be accomplished by systolically sweeping $\bar{H}_{ij}$ vectors (6×1) left in Table 4.7 according to the following algorithm:

for $i = 0$ to n

{

1. cpus in columns 1 to $n - i$ calculate dot products with $\bar{H}$ vector and resident H vectors.
2. shift scalar dot product result in rows n to $n - i$ up to row above and accumulate

3. shift $\bar{H}$ vectors in columns 1 to $n - i$ left

} (4.8)

where the H vector in $\text{cpu}_{j,k}$ is $[J_j^j c_{j,k}^j \;\; m_j(\rho_{j,k}^j \times c_{j,k}^j)]^T$. Table 4.7 graphically illustrates the first four computations of this algorithm. Each matrix element represents a cpu, and the contents of each element represent a computation (– indicates no computation). Note that the grid is an $\frac{n(n+1)}{2} + n$ triangular mesh, so the cpu in element (col 1, row 2) does not exist.

Multiplying the momentum derivative vector $[\dot{L} \;\; m\ddot{p}]$ by $\bar{H}$ requires the $[c_{i,j}^i \;\; (\rho_{i,j}^i \times c_{i,j}^i)]^T$ vectors to be shifted into row 0 from row 1. This is done as part of the algorithm in equation 4.8. Table 4.7 shows row 0 $[\dot{L}]$ appended to the rows calculating H (in computation cycle 0). The vectors in the $\bar{H}$ matrix, $\bar{H}_{i,k}$, are then systolically shuffled leftward through the array, each computation phase involving dot products and/or accumulations.

The first four computations (two cycles) of the algorithm are demonstrated in the sequence presented in Table 4.7 (only rotational momentum computations $c^T \cdot (Jc)$ and $c^T \cdot \dot{L}$ are shown). Between computation phases (1) and (2) in Table 4.7, the dot products calculated in (1) are also passed up from column 4 to column 3 in an accumulative sweep of the dot product results, and the $\bar{H}_{ij}$ (the $c_{i,j}^i$ in Table 4.7) are simultaneously passed left from rows 1 through n to rows 0 through $n - 1$. Computation cycles (2) and (3), the accumulation (2) and the next dot product (3), are then computed as one phase, followed by another communication phase between (3) and (4).

By simply passing $\bar{H}_{i,j}^T = [c_{i,j}^i \;\; (\rho \times c_{i,j}^i)]^T$ as 6×1 vectors, the communication phases are not hidden, as the $\bar{H}_{i,j}^T \cdot H_{i,k}$ dot product must be completed before the results can be passed upward and accumulated. However, if the dot product computations and the vector communications are split up so that $c_{i,j}^i$ and $(\rho_{i,j}^i \times c_{i,j}^i)$ are sent separately as 3×1 vectors, and the dot products for each of these vectors are calculated separately, the communications can be hidden (although a 3×1 vector dot product computation is a less costly atomic action than a 3×1 vector communication). Furthermore, it is possible to partially or completely eliminate the communication of $\bar{H}_{i,j}$ by taking advantage of two facts:

1. $c^i_{i,j}$ need not be transmitted left by rows 1 to n in Table 4.7 (south to north in Fig. 4.3), since it is already present in rows to the left, having passed through during operation 2 of Table 4.1. Thus, in $\text{cpu}_{k,i}$ while $(\rho^i_{i,j} \times c^i_{i,j})$ is being received from the right, the dot product partial sum

$$p_{ki,j-1} = (c^i_{i,j-1})^T (J^i_i c^i_{i,k}) + (\rho^i_{i,j-1} \times c^i_{i,j-1})^T m_i (\rho^i_{i,k} \times c^i_{i,k}) \qquad (6\times, 5+) \qquad (4.9)$$

is calculated, and the moving sum of other partials $\sum_{ki}$ is passed up from below. in the next operation, when $c^i_{i,j}$ would normally be sent, $(\rho^i_{i,j+1} \times c^i_{i,j+1})$ can be sent and

$$p_{ki,j} = (c^i_{i,j})^T (J^i_i c^i_{i,k}) + (\rho^i_{i,j} \times c^i_{i,j})^T m_i (\rho^i_{i,k} \times c^i_{i,k}) \qquad (4.10)$$

computed. A moving sum $\sum_{ki}$ is one which originates in $\text{cpu}_{i,n}$ and migrates to $\text{cpu}_{i,k}$ after k cycles, accumulating the partial sums $p_{ki,j}$, $j = 1, ...k$ relevant to that cpu's matrix multiplication at each step.

Previously in $\text{cpu}_{i,k}$, while $(\rho^i_{i,j} \times c^i_{i,j})$ was being received, only the $c^{iT}_{i,j}(J^k_k c^i_{i,k})$ computation $(3\times, 2+)$ could simultaneously occur since $c^i_{i,j}$ had to be received in the prior communication. With $c^i_{i,j}$ now resident, equation (4.9) can be computed at the time that $(\rho^i_{i,j} \times c^i_{i,j})$ is transmitted, and the $(\rho^i_{i,j} \times c^i_{i,j})$ vectors can be sent at twice the rate since the $c^i_{i,j}$ vectors are not interleaved. In fact, the vector transmission is more easily hidden since the cost of the dot products in equation 4.9 is $(6\times, 5+)$, approximately as expensive as a 3×1 vector communication.

The calculation of f_v can similarly be improved by passing the $c^i_{i,j}$ vector north to row 0 as it is being produced by row 1 cpus (in operation 3 of Table 4.1). The $c^i_{i,j}$ vectors will thus be present when phase two begins, and only $(\rho^i_{i,j} \times c^i_{i,j})$ needs to be passed in.

2. The communication of $(\rho^i_{i,j} \times c^i_{i,j})$ can also be avoided by noting that the $\rho^i_{i,j}$ have also passed through the array (rows 1 to n) due to operation 3 in Table 4.1 (and may also be passed from row 1 cpus to row 0 cpus during operation 3 (in a similar manner to $c^i_{i,j}$) in note 1 above). However, the cross product $(\rho^i_{i,j} \times c^i_{i,j})$ (operation 7) must then be computed at each cpu before the dot product in the second term of equation 4.9 is calculated.

The computational complexity of the systolic multiplication by $\bar{H}$ is $n(6\times,6+)$ (the extra n adds occur due to the accumulating moving sum).

The overall complexity, assuming the cross product result of $(\rho^i_{i,j} \times c^i_{i,j})$ is passed through the array, is then:

$$\begin{aligned}\sum_{\mathcal{M}} &= (n-1)(16\times,16+) + n(6\times,6+) + (40\times,21+) \\ &= (22n+24)\times,(22n+5)+ \qquad (4.11)\\ \sum_{f_v} &= (n-1)(18\times,13+) + n(6\times,6+) + (45\times,38+) \\ &= (24n+27)\times,(19n+25)+ \qquad (4.12)\end{aligned}$$

and if the $\rho^i_{i,j} \times c^i_{i,j}$ vectors are calculated rather than passed in, a cost of $n(6\times,5+)$ is incurred, and so the overall cost is:

$$\begin{aligned}\sum_{\mathcal{M}} &= (n-1)(16\times,16+) + n(6\times,6+) + n(6\times,3+) + (40\times,21+) \\ &= (28n+24)\times,(25n+5)+ \qquad (4.13)\\ \sum_{f_v} &= (n-1)(18\times,13+) + n(6\times,6+) + n(6\times,3+) + (45\times,38+) \\ &= (30n+27)\times,(22n+25)+ \qquad (4.14)\end{aligned}$$

The communications for operation 9 in Table 4.1 (the accumulation of partial sums by the moving sum), represented by the vertical data shift in Table 4.7, must still be performed, but the cost is hidden by the dot product computations. The cost of the extra cross product $\rho \times c$, done to avoid communicating, is computationally expensive, but may be worthwhile if the cost of communication is more than a vector cross product, or if there is difficulty in hiding the communications within the computations.

	row 0	row 1	row 2	row 3	row 4
col 1	$\dot{L}_1$	$J_1^1 c_{1,1}^1$			0
col 2	$\dot{L}_2$	$J_2^2 c_{2,1}^2$	$J_2^2 c_{2,2}^2$		
col 3	$\dot{L}_3$	$J_3^3 c_{3,1}^3$	$J_3^3 c_{3,2}^3$	$J_3^3 c_{3,3}^3$	
col 4	$\dot{L}_4$	$J_4^4 c_{4,1}^4$	$J_4^4 c_{4,2}^4$	$J_4^4 c_{4,3}^4$	$J_4^4 c_{4,4}^4$

	row 0	row 1	row 2	row 3	row 4
col 1	-	$(c_{1,1}^1)^T J_1^1 c_{1,1}^1$			1
col 2	-	$(c_{2,1}^2)^T J_2^2 c_{2,1}^2$	$(c_{2,2}^2)^T J_2^2 c_{2,2}^2$		
col 3	-	$c_{3,1}^3)^T J_3^3 c_{3,1}^3$	$(c_{3,2}^3)^T J_3^3 c_{3,2}^3$	$(c_{3,3}^3)^T J_3^3 c_{3,3}^3$	
col 4	-	$(c_{4,1}^4)^T J_4^4 c_{4,1}^4$	$(c_{4,2}^4)^T J_4^4 c_{4,2}^4$	$(c_{4,3}^4)^T J_4^4 c_{4,3}^4$	$(c_{4,4}^4)^T J_4^4 c_{4,4}^4$

	row 0	row 1	row 2	row 3	row 4
col 1	-	-			2
col 2	-	-	-		
col 3	-	$\sum_{11} = (c_{3,1}^3)^T J_3^3 c_{3,1}^3 + (c_{4,1}^4)^T J_4^4 c_{4,1}^4$	$\sum_{22} = (c_{3,2}^3)^T J_3^3 c_{3,2}^3 + (c_{4,2}^4)^T J_4^4 c_{4,2}^4$	$\sum_{33} = (c_{3,3}^3)^T J_3^3 c_{3,3}^3 + (c_{4,3}^4)^T J_4^4 c_{4,4}^4$	
col 4	-	-	-	-	-

	row 0	row 1	row 2	row 3	row 4
col 1	$(c_{1,1}^1)^T \dot{L}_1$	-			3
col 2	$(c_{2,1}^2)^T \dot{L}_2$	$(c_{2,2}^2)^T J_2^2 c_{2,1}^2$	-		
col 3	$(c_{3,1}^3)^T \dot{L}_3$	$(c_{3,2}^3)^T J_3^3 c_{3,1}^3$	$(c_{3,3}^3)^T J_3^3 c_{3,2}^3$	-	
col 4	$(c_{4,1}^4)^T \dot{L}_4$	$(c_{4,2}^4)^T J_4^4 c_{4,1}^4$	$(c_{4,3}^4)^T J_4^4 c_{4,2}^4$	$(c_{4,4}^4)^T J_4^4 c_{4,3}^4$	-

	row 0	row 1	row 2	row 3	row 4
col 1	-	-			4
col 2	-	$\sum_{11} = (c_{2,1}^2)^T J_2^2 c_{2,1}^2 + \sum_{11}$	$\sum_{22} = (c_{2,2}^2)^T J_2^2 c_{2,2}^2 + \sum_{22}$		
col 3	$\sum_{10} = (c_{3,1}^3)^T \dot{L}_3 + (c_{4,1}^4)^T \dot{L}_4$	$\sum_{21} = (c_{3,2}^3)^T J_3^3 c_{3,1}^3 + (c_{4,2}^4)^T J_4^4 c_{4,1}^4$	$\sum_{32} = (c_{3,3}^3)^T J_3^3 c_{3,2}^3 + (c_{4,3}^4)^T J_4^4 c_{4,2}^4$	-	
col 4	-	-	-	-	-

Table 4.7: First four computations of $\bar{H}^T H$ (rows 1 to 4) and $\bar{H}^T[\dot{L}]$ (row 0) (rotational components only)

4.5 Parallel Matrix Solver

Once the inertia matrix $\mathcal{M}$ and the driving vector b ($b = f_v + \tau_a$) have been assembled, the system of equations must be solved for $\ddot{\theta}$. An efficient feedforward wavefront matrix solver has been implemented for the equation

$$\begin{aligned} \mathcal{M}\ddot{\theta} &= b \\ \text{or, using matrix terminology, } Ax &= b \end{aligned} \tag{4.15}$$

using a feedforward algorithm proposed by [Jainandunsing 89]. Although Jainandunsing's algorithm is a systolic array, it can be used by the wavefront array employed here (a wavefront cpu such as the transputer is activated by the arrival of data, whereas a systolic cpu is synchronised by a global controller, and calculates regardless of the presence of data).

For dense systems of equations (when most elements of A are nonzero), the direct method solver is the best method of solution. Direct methods can be divided into backsubstitution and feedforward methods.

factorisation and backsubstitution method

Backsubstitution methods compute an LU, QR or Cholesky factorisation of the A matrix, followed by a backsubstitution to derive the solution vector x. The factorisation is performed by premultiplying the A matrix with embeddings of 2x2 rotations, which can be linear, trigonometric/orthogonal, or hyperbolic. The multiplication triangularizes A to an upper triangular matrix. For the linear rotation, the 2x2 matrix has the form:

$$\begin{bmatrix} 1 & 0 \\ \alpha_{i,j} & 1 \end{bmatrix} \tag{4.16}$$

which gives the following $N \times N$ embedding:

$$\Theta_{ij}(0) = \begin{bmatrix} I_{i-1} & & & & \\ & 1 & & 0 & \\ & & I_{j-i-1} & & \\ & \alpha_{i,j} & & 1 & \\ & & & & I_{N-j} \end{bmatrix} \tag{4.17}$$

For an appropriate $\alpha_{i,j}$, the product $\Theta_{ij}(0)A$ eliminates the $a_{i,j}$ element in the A matrix. A cascaded backward series of these embeddings $N, ...i$, (N is the size of the matrix A) produces a product

$$\Theta(0) = \Pi_{i,j}\Theta_{i,j}(0) = (\Pi_{i=1}^{N}\Theta_{i,N}(0))...(\Pi_{i=1}^{N}\Theta_{i,2}(0)) \cdot (\Pi_{i=1}^{N}\Theta_{i,1}(0)). \tag{4.18}$$

When the α_{ij} are calculated such that the strictly lower part of A is eliminated, the resulting product $\Theta(0)$ is L^{-1}, the inverse of the factor L in the LU factorisation. When $\Theta(0)$ is multiplied with A, the result is the factor U ie.

$$\begin{aligned} A &= LU \\ \Theta(0)A &= \Theta(0)LU. \\ \text{If } \Theta(0) &= L^{-1}, \\ \Theta(0)A &= L^{-1}LU = U. \end{aligned} \tag{4.19}$$

Thus, if each cpu in the array has calculated the appropriate $\alpha_{i,j} = a_{ij}^{(j)}/a_{jj}^{(j)}$ and formed the 2×2 rotation ($a_{ij}^{(j)}$ is the value of element $A_{i,j}$ after j successive rotations), then feeding A into the systolic array produces U. To solve $Ax = b$, let $y = Ux$ and $A = LU$. Then, if

$$\begin{aligned} Ax &= b, \\ LUx &= b. \\ \text{If } Ly &= b, \\ y &= L^{-1}b = \Theta(0)b \end{aligned} \tag{4.20}$$

From equation (4.19) it can be observed that when b is premultiplied with $\Theta(0)$ (ie when b is also fed into the systolic array), y is obtained, from which x may be calculated through the backsubstitution:

$$\begin{aligned} Ux &= y \\ x &= U^{-1}y \end{aligned} \tag{4.21}$$

Once $U = \Theta(0)A$ and $y = \Theta(0)b$ are obtained, they are passed to the backsubstitution algorithm, which is architecturally organised as a pipeline. This pipeline calculates the x vector in reverse order $(x_n, ...x_1)$, with y and U fed simultaneously into the end and the side of the pipe respectively. Unfortunately, the elements of U, which are produced a column at a time starting with column 1, are in the reverse order to that desired by the backsubstitution. As a result, a systolic implementation cannot overlap the factorisation and backsubstitution, as the factorisation must be completed before the backsubstitution begins. Systolic implementations such as Liu and Young's [Liu 83] have a complexity of $3n-2$ cycles for the factorisation, and $4n-3$ cycles for the backsubstitution (loading, computation, unloading of solution), for a total of $7n-5$ cycles [Jainandunsing 89]. Each cycle requires $(1\times, 1+)$.

feedforward method

Feed forward techniques solve a system using LU factorisation without backsubstitution. Faddeev [Faddeev 59] proposed an algorithm which augments the matrix A with an identity matrix $-I_N$. The factorisation of the augmented matrix $[A^T \quad -I_N]$ which is accomplished by feeding A and I into the augmented cpu array, produces $\Theta(0) = (L')^{-1}$, which is the inverse of L', the lower factor in the factorisation result $[L' \quad U']$. When b is also fed into the array after A, it is transformed by $\Theta(0)b = x$ to the solution vector x. While this algorithm is fast (complexity $3n$ cycles) it requires pivoting to maintain stability, which is difficult to implement for a systolic architecture. Additionally, the algorithm requires cpus for processing $-I_N$, demanding almost twice the number of cpus compared to other systolic algorithms.

4.5.1 The Jainandunsing feedforward algorithm

The algorithm proposed by Jainandunsing [Jainandunsing 89] is a feedforward direct procedure involving a reorganisation of the backsubstitution into a feedforward sequence, so that the sequence of results from the factorisation (which is itself different from the previous algorithm) is immediately processed by the new feedforward phase that follows (in fact, the backsubstitution has been rephrased as an LU-type factorisation). The factorisation, instead of producing U and y starting from column 1 of U, produces $[L^T \;\; U^{-T}]$ starting from row 1. The feedforward algorithm that follows accepts the $[L^T \;\; U^{-T}]$ values in the order they are generated. Non-singular matrices are assumed.

Variations have been developed by Jainandunsing for Cholesky factorisation (hyperbolic rotations) and QR factorisation (orthogonal trigonometric rotations). Here, we use the LU variation because it uses linear rotations, which are cheaper to compute than hyperbolic or trigonometric rotations. Although the LU factorisation is susceptible to numerical stability problems due to roundoff errors [Jainandunsing 1989], it was chosen because the matrices dealt with in this thesis are positive definite and very small in size, for which numerical problems are unlikely to occur. The number of cpus required is also reasonable, as apart from the $\frac{n(n+1)}{2}$ array for the A matrix, only one extra row of n cpus is needed (to process the b vector). Thus the architecture in Fig. 4.1 is ideal for this solver, since row 0 is already used to calculate b while rows 1 through n are used to calculate $\mathcal{M}$.

If instabilities are likely to occur, the QR factorisation variation should be chosen. For the QR factorisation, the computer architecture remains the same as for the LU factorisation, but the linear rotations are replaced by trigonometric rotations.

factorisation

Consider the augmented matrix $\left[\begin{array}{cc} A^T & I_N \end{array} \right]$. If this matrix is premultiplied by U^{-T}, it will be factorised to $[L^T \;\; U^{-T}]$. U^{-T} can be defined as a cascaded backward series $(N, ...1)$ of embedded rotations in a manner similar to the previous factorisation matrix $\Theta(0)$. If the 2x2

embedding is a premultiplier of the augmented matrix, it is

$$U_{i,j} = \begin{bmatrix} I_{j-1} & & & & \\ & 1 & & 0 & \\ & & I_{i-j-1} & & \\ & \alpha_{i,j} & & 1 & \\ & & & & I_{N-i} \end{bmatrix} ; \; i,j = 1,...N-1. \tag{4.22}$$

For j= 1, $\Pi_{i=1}^{N-1} U_{i,1}$ is a cascaded backward series $(U_{N-1,1})(U_{N-2,1})...(U_{1,1})$ of rotations which zeros the below-diagonal elements of the first column in A^T. The appropriate choice of $\alpha_{i,1}$ is $a_{i,1}^{(1)}/a_{1,1}^{(1)}$. The resulting matrix is then $A^{T(1)}$. This can be repeated for $j = 2,...N-1$ where $\alpha_{i,j} = a_{i,j}^{(j)}/a_{j,j}^{(j)}$ such that we get

$$(\Pi_{i=N-1}^{N-1} U_{i,N-1})...(\Pi_{i=2}^{N-1} U_{i,2}) \; (\Pi_{i=1}^{N-1} U_{i,1}) \; \left[A^T \; I_N\right] = \left[L^T \; U^{-T}\right] \tag{4.23}$$

The above multiplication yields a factorisation in which the rows of $\left[L^T \; U^{-T}\right]$ are produced in row order starting from row 1. By producing $\left[L^T \; U^{-T}\right]$ in this order, the rows can be immmediately sent to the feedforward algorithm.

backsubstitution redefined as a feedforward algorithm

Assume that the factorisation has occurred and the factors $\left[L^T \; U^{-T}\right]$ are available. Consider $Ax = b$ where $A = LU$. If $Ly = b$ and $x = U^{-1}y$, then

$$[y^T \; 1] \begin{bmatrix} L^T \\ -b^T \end{bmatrix} = 0 \tag{4.24}$$

$$[y^T \; 1] \begin{bmatrix} U^{-T} \\ 0^T \end{bmatrix} = x^T \tag{4.25}$$

The matrix $[L \; -b]^T$ can be reduced to an upper triangular form $[R^T(0) \; 0]^T$ in which the last row is zero (i.e. the $-b^T$ row is zeroed) by applying a sequence of rotations $\Theta(0) = \Pi_{i=1}^{N} \Theta_i(0)$

where

$$\Theta_i(0) = \begin{bmatrix} I_{i-1} & & & \\ & 1 & & 0 \\ & & I_{N-i-1} & \\ & \alpha_i & & 1 \end{bmatrix} \tag{4.26}$$

$$\Theta(0) \begin{bmatrix} L^T \\ -b^T \end{bmatrix} = \begin{bmatrix} R(0) \\ 0^T \end{bmatrix} \tag{4.27}$$

$R(0)$ is an upper triangular remainder matrix equal to U. Equations (4.23) and (4.26) both produce 0, suggesting that the last row of $\Theta(0)$ is proportional to $k[y^T\ 1]$, where k is a scalar proportionality constant (k=1 for the LU method [Jainandunsing 89]). This allows equation (4.24) to be rewritten as

$$\Theta(0) \begin{bmatrix} U^{-T} \\ 0^T \end{bmatrix} = x^T \tag{4.28}$$

since the last row of $\Theta(0)$ is $k[y^T\ 1]$. Thus, grouping (4.26) and (4.27),

$$\Theta(0) \begin{bmatrix} L^T & U^{-T} \\ -b^T & 0^T \end{bmatrix} = \begin{bmatrix} R(0) & ? \\ 0^T & x^T \end{bmatrix}, \tag{4.29}$$

where ? implies an unused result.

interleaving

Premultiplying the matrix $[A^T\ I_N]$ by $\Pi_{i=1}^{N-1} U_{i1}$ eliminates the first column of the strictly lower part of the augmented matrix. The output is the first row of $[L^T\ U^{-T}]$, which is then fed directly into the rephrased 'backsubstitution' calculator. This feedforward 'backsubstitution' algorithm applies $\Theta_1(0)$ to the just-received first row of $[L^T\ U^{-T}]$ and to the first element of vector $[-b^T\ 0]$, causing the first element b_1 to be zeroed (the first 0 in the bottom row of the result matrix in equation (4.26)). Continued interleaving of the $\Pi_{i=1}^{N-1} U_{ij}$ and $\Theta_j(0)$ eventually results in the solution vector x^T being produced after $4n$ cycles, x_1 being first after $3n$ cycles. A systolic array for this algorithm is shown in Fig. 4.4.

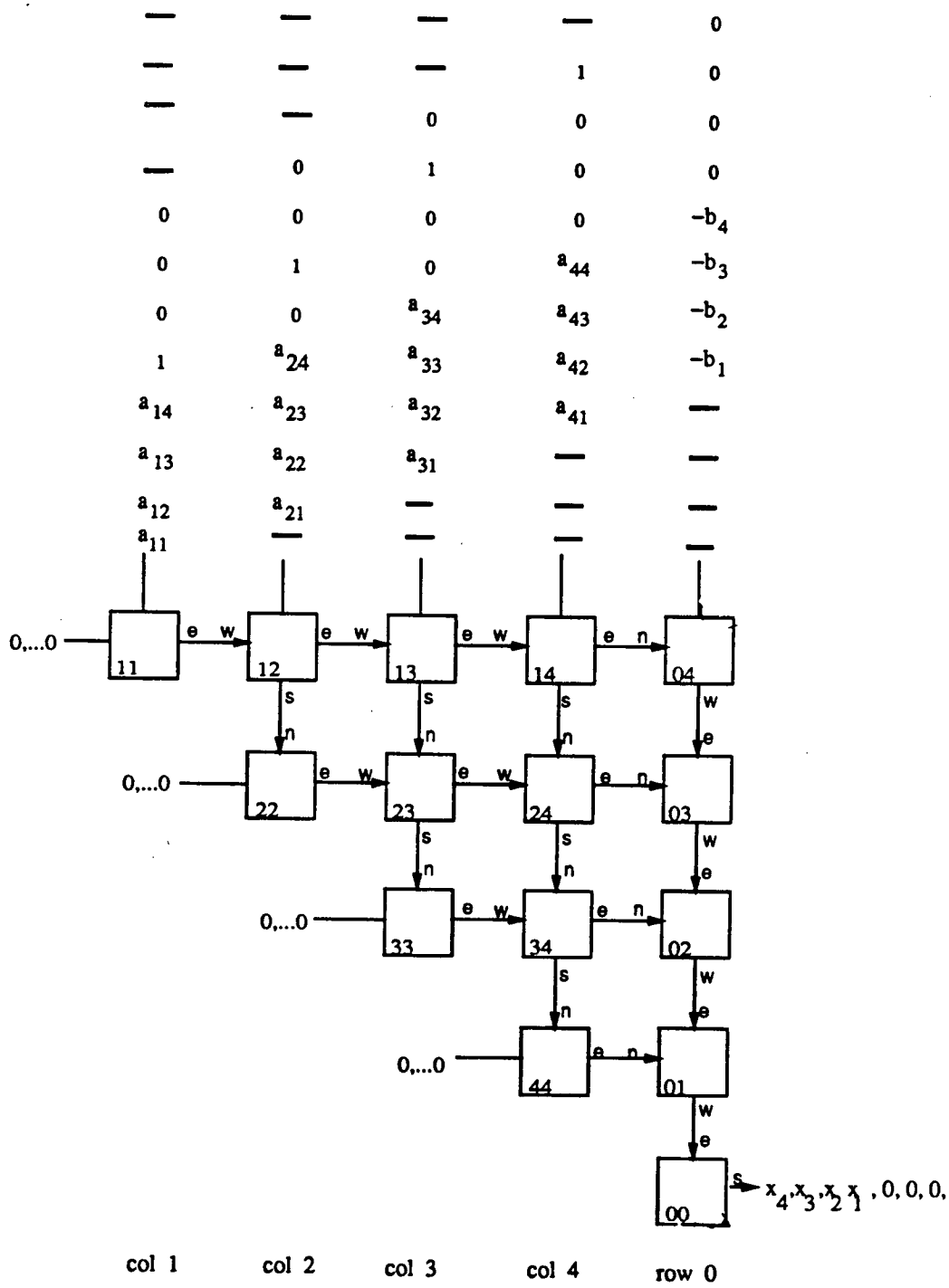

Figure 4.4: Systolic array for solving the matrix equation $Ax = b$

integration of formation and solving phases

The elements of A and b are distributed throughout the array by the inertia matrix and force vector algorithms during calculation. The solver algorithm in Fig. 4.4 requires that A and b

be moved to the edges of the array before beginning. Fig. 4.5 shows the movement of the data to the cpus of row 1 e.g. A_{44}, A_{34}, A_{24} and A_{14} are shifted to cpu_{14}. Similarly, A_{34}, A_{33}, A_{23} and A_{13} are shifted to cpu_{13}. Note that $A_{ij} = A_{ji}$ i.e. $A = A^T$. This data movement phase requires $n - 1$ data shifts, each one moving a single float value. These communications cannot be hidden by computations. The b cpus (cpus 01 to 04 in row 0 of Fig. 4.5) are connected not

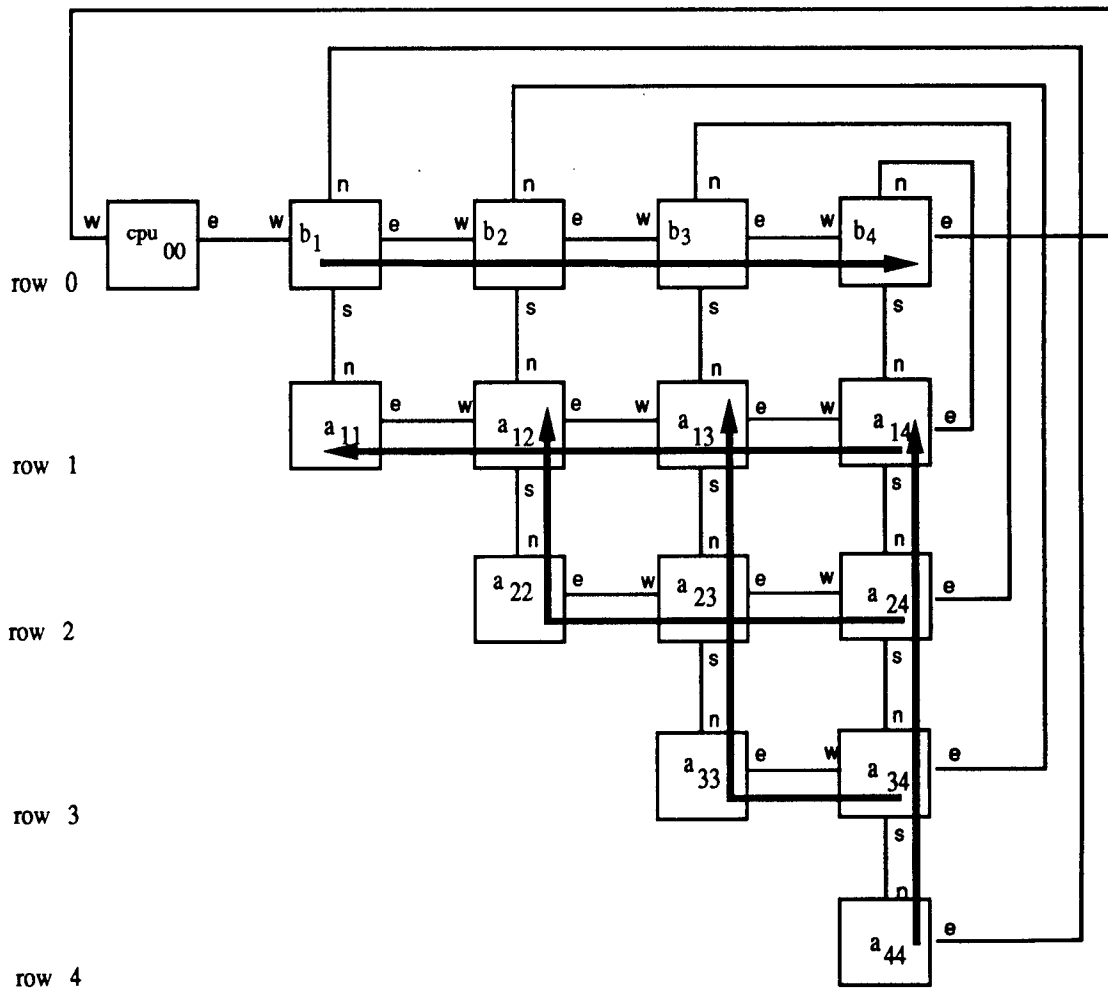

Figure 4.5: Data shifting phase prior to solver

only to cpus 11, 12, 13 and 14, but also to 44, 34, 24 and 14 respectively through north-to-east links (see Fig. 4.5). To derive a simple and efficient data shifting phase that integrates well

with the solver, the b (row 0) cpus are aligned as column '5' and placed next to the 4th column of cpus, as shown in Fig. 4.4. If this rearranged architecture is not used, an alternative data shifting algorithm (bringing the data in row i out to cpu_{i4}) is twice as long, since the elements a_{ij} collected in cpu_{i4} must be moved to $\text{cpu}_{4-i,4}$ after they are brought to the edge of the array, in order to get the data arrangement required in Fig. 4.4. Fig. 4.5 shows how the cpus are arranged in a cone-like mesh architecture.

```
/* row_id is the row in which this cpu resides in the mesh   */
    for( i = 1; i <= twoN; i++ )
    {
        ChanIn( northin, (char *)&yold, 4 ) ;
        ChanIn( westin, (char *)&xold, 4 ) ;
        if( i == row_id ) alpha = -yold/xold ;
        if( i < row_id  )
        {
            xnew = 0.0 ;
            ynew = 0.0 ;
        }
        else
        {
            xnew = xold ;
            ynew = yold + alpha*xold ;
        }
        ChanOut( southout, (char *)&ynew, 4 ) ;
        ChanOut( eastout, (char *)&xnew, 4 ) ;
    }
```
(4.30)

producing and using $\ddot{\theta}$

The solution vector $x = \ddot{\theta}$ is produced at cpu_{00} (see Fig. 4.4), and the $\ddot{\theta}_i$ are redistributed back to the b_i cpus ($\text{cpu}_{0,i}$) via the west link of cpu_{00} to the east link of b_4 as they are produced. The $\ddot{\theta}_i$ are integrated by each $\text{cpu}_{0,i}$ to obtain $\dot{\theta}_i$ (necessary for calculating the angular velocity), and integrated again to get θ_i, which is used by the graphics display of the dynamics simulator (further discussed in Chapter 5). The θ_i has to be fed back along the row 0 pipeline to $\text{cpu}_{0,0}$ and on to the host or the graphics computer. Each cycle in the systolic algorithm requires

$(1\times, 1+)$ for the rotation (a multiply and accumulate). One $\div$ (equivalent in cost to a $\times$) is necessary to calculate $\alpha_{i,j} = a_{i,j}^{(j)}/a_{j,j}^{(j)}$ (or $\alpha_{0,j} = -b_j^{(j)}/a_{j,j}^{(j)}$ for the force vector cpus) once during the algorithm for each cpu. The C code for a cpu implementing this solver is given above in (4.30). The last value of $\ddot{\theta}_i$ is produced after $4n$ cycles, in contrast to $7n - 5$ cycles for the previous factorisation and backsubstitution algorithm. The communications of intermediate data and the rotation computations cannot be overlapped, and so the communication cost is $4n$ cycles (passing single floating point values), plus n cycles for the initial data shifting phase described earlier.

4.6 Summary of the Proposed Forward Dynamics Algorithm

To complete one cycle of the forward dynamics algorithm, it is necessary to

1. Calculate the force vector f_v
2. Calculate the inertia matrix $\mathcal{M}$
3. Solve the matrix equation $\mathcal{M}\ddot{\theta} = f_v + \tau_a$.

This thesis has developed $O(n)$ multiprocessor algorithms for all three of these phases. The n cpus in row 0 and the $\frac{n(n+1)}{2}$ cpus in rows 1 to n simultaneously calculate $[\ddot{L} \;\; m\ddot{p}]$ and H, respectively, and then multiply by $\bar{H}^T$ to compute f_v and $\mathcal{M}$. These two phases require a 2d mesh architecture as data must be passed between nearest neighbours. Following this, the elements of $f_{v(i)}$ reside in cpu_{0i} and $\mathcal{M}_{ij}$ reside in cpu_{ij}. This data is shifted to the edge (row 1 cpus cpu_{1i} and cpu_{04}) and the system is then solved using a new systolic solver derived by [Jainandunsing 89]. The costs associated with each phase of the proposed algorithm are given in Table 4.8 for the W (Wong) algorithm.

4.7 Parallelism in Other Formulations

In this section a discussion on the parallel formulations mentioned in Chapter 2 is presented in which the computational complexity is considered in more detail. The complexities of the

		f_v	$\mathcal{M}$	solver $\mathcal{M}^{-1}$	$\sum_{\mathcal{M}+\mathcal{M}^{-1}}$
L	×	87	$9\lceil\frac{n+1}{2}\rceil + 31\lceil\log_2 n\rceil + 116$	$7n-5$	$9\lceil\frac{n+1}{2}\rceil + 31\lceil\log_2 n\rceil + 7n + 111$
	+	$15\lceil\log_2 n\rceil + 65$	$5\lceil\frac{n+1}{2}\rceil + 28\lceil\log_2 n\rceil + 3n + 92$	$7n-5$	$5\lceil\frac{n+1}{2}\rceil + 28\lceil\log_2 n\rceil + 10n + 87$
	cpu	n (c)	n (c)	$\frac{n(n+1)}{2}$ (m)	
W	×	$24n+27$	$22n+24$	$4n+1$	$26n+25$
	+	$19n+25$	$22n+5$	$4n$	$26n+5$
	cpu	n (p)	$\frac{n(n+1)}{2}$ (m)	$\frac{n(n+1)}{2}+n$ (m)	
A	μs		$27n + 99\log_2 n - 1$		
	cpu		n (c)		
F	×		$48\lceil\log_2 n\rceil + 102$		
	+		$63\lceil\log_2 n\rceil + 66$		
	cpu		$\frac{n(n+1)}{2}$ (m)		
H	×		$27\lceil\log_2 n\rceil + 6n + 109$	$7n-5$	$27\lceil\log_2 n\rceil + 13n + 104$
	+		$31\lceil\log_2 n\rceil + 5n + 92$	$7n-5$	$31\lceil\log_2 n\rceil + 12n + 87$
	cpu		n (p)	$\frac{n(n+1)}{2}$ (m)	

Table 4.8: Parallel computational complexity

inertia matrix, force vector, and solver algorithms developed in this thesis are summarised in Table 4.8. In the following discussion the communication complexity is discussed but not explicitly estimated in the Table. This is because the authors of the various formulations that are listed in Table 4.8 either excluded or completely included all communication costs, and did not consider the case when some communications could be hidden. Thus, to compare the algorithms under similar circumstances, a comparison is made solely on computational costs. It should also be noted that the complexity estimates in Table 4.8 are not exactly as presented in the relevant papers cited. It has been necessary to assume that extra cpus are available to compute the force vector values, so that this cost is masked by the inertia matrix. Additionally, in cases where only the inertia matrix formation is discussed, Liu's systolic solver has been assumed to calculate the matrix solution.

In Table 4.8, (p), (c), and (m) indicate pipe, cube and mesh architectures respectively. L, W, A, F, and H refer to the Lee [Lee 88], Wong (this thesis), Amin-Javaheri [Amin-Javaheri 88], Fijany[Fijany 89], and Hwang [Hwang 88] formulations, while *cpu* refers to the number of cpus used.

4.7.1 Lee and Chang

Lee and Chang [Lee 88] implemented their composite rigid body based force vector and inertia matrix algorithms on an n processor generalised cube network architecture. The force vector was computed in $O(\lceil \log_2 n \rceil)$ time and the inertia matrix algorithm in $O(n + \lceil \log_2 n \rceil)$ time. To achieve this, the recursive doubling algorithm (RDA), which is used for many of the calculations (e.g. calculating the rotational matrices A_i^0, the composite centre of mass $\hat{c}_i$, and the composite body inertia $\hat{J}_i$ $(i = 1,...n)$), computes each variable in $O(\lceil \log_2 n \rceil)$ time. It is implemented using the generalised cube as a $\lceil \log_2 n \rceil$ stage switching network for distributing the intermediate matrix data among the cpus (Fig. 4.6). The RDA algorithm computes A_i^0 in $O(\lceil \log_2 n \rceil)$

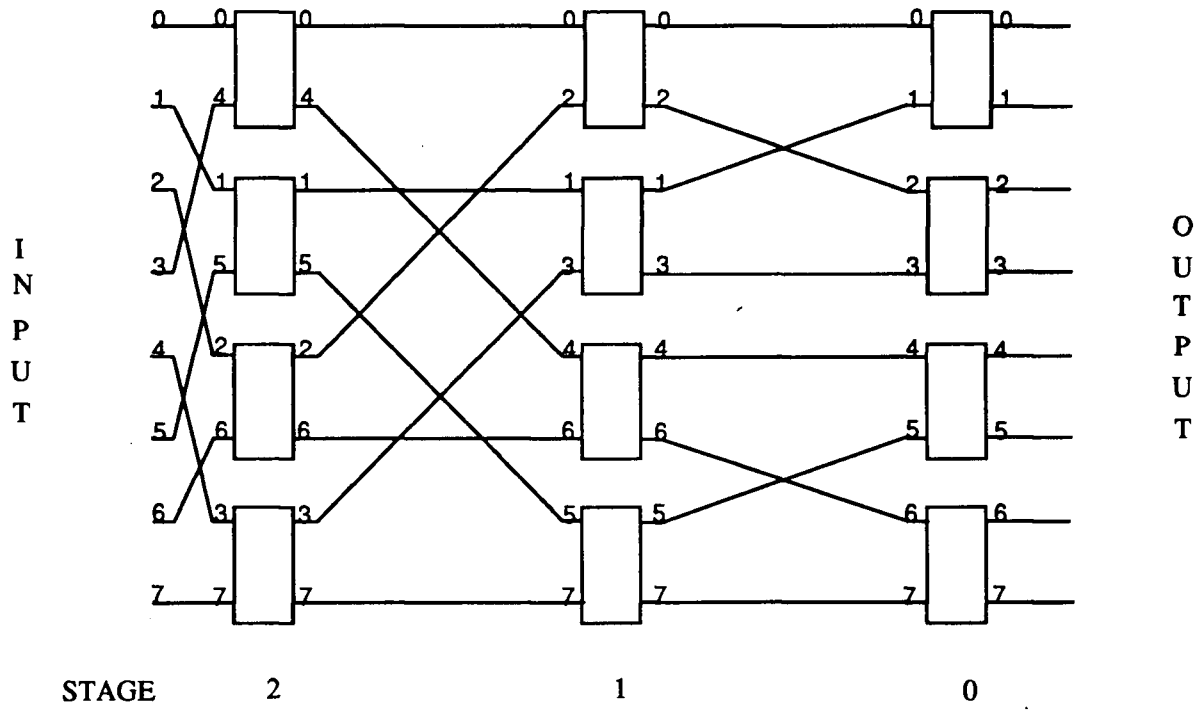

Figure 4.6: Generalised cube network topology for $n = 8$. Adapted from [Lee 88]

time by subdividing the computations into pairs of matrix-matrix multiply operations over $\lceil \log_2 n \rceil$ stages. To hide the communications cost of sending or receiving A_i^j matrices, A_i^j must be divided into three 3×1 vectors, each sent one at a time so that the matrix-matrix

multiply can be computed as three matrix-vector multiplies. This only hides two of the three vectors, but the beginning of the next matrix-matrix computation can begin while the last result vector is being transmitted. This method of hiding communications also applies to the computation of $\hat{J}_i$. Another instance of the RDA algorithm, however, the recursive vector addition of the composite body centre of mass $\hat{c}_i$ (a 3×1 vector), cannot overlap communications with computations since the complete vector result (a computation 'atom') must be computed before being communicated to the next stage.

Lee's $\frac{n(n+1)}{2}$ mesh algorithm, which uses an n cpu cube architecture for the inertia matrix and an $\frac{n(n+1)}{2}$ systolic mesh to solve the matrix equation, is presented in Table 4.8 as formulation (L). Lee uses Liu and Young's systolic Cholesky solver algorithm [Liu 83]. In (L) we assume that an extra n cpus is available to calculate the force vector simultaneously with the inertia matrix n cpu pipeline. Thus, Lee's architecture requires $2n$ cpus to form $\mathcal{M}$ and f_v, each n forming a generalised cube network, followed by an $\frac{n(n+1)}{2}$ mesh to solve the equations. The complexity of Lee's formulation is presented in Table 4.8. Note that we have assumed Jainandunsing's complexity estimate of $7n - 5$ cycles for Liu's systolic Cholesky factorisation (plus backsubstitution) algorithm. Lee's inertia matrix complexity has an added $3(n-1)$ additions which were left out in Lee's complexity polynomial (equation 25 in his algorithm). The last column of the table is the sum of the inertia matrix and solver complexities, since it is assumed that the force vector complexity is approximately the same as the inertia matrix complexity, and is therefore hidden when calculated simultaneously.

4.7.2 Amin-Javaheri and Orin

Amin-Javaheri and Orin [Amin-Javaheri 88] developed a number of parallel algorithms for the inertia matrix, the most efficient of which used a hypercube architecture. The composite body variables $\hat{c}_i$, $\hat{J}_i$ and A_i^0 were calculated in $O(\lceil \log_2 n \rceil)$ time using the RDA algorithm and the architecture of Fig. 4.7. To reduce the communication delay in the RDA algorithm, computations are duplicated by source and destination cpus so that the data for the next level of computation is at most only one hop from its destination. By doing this, the computational

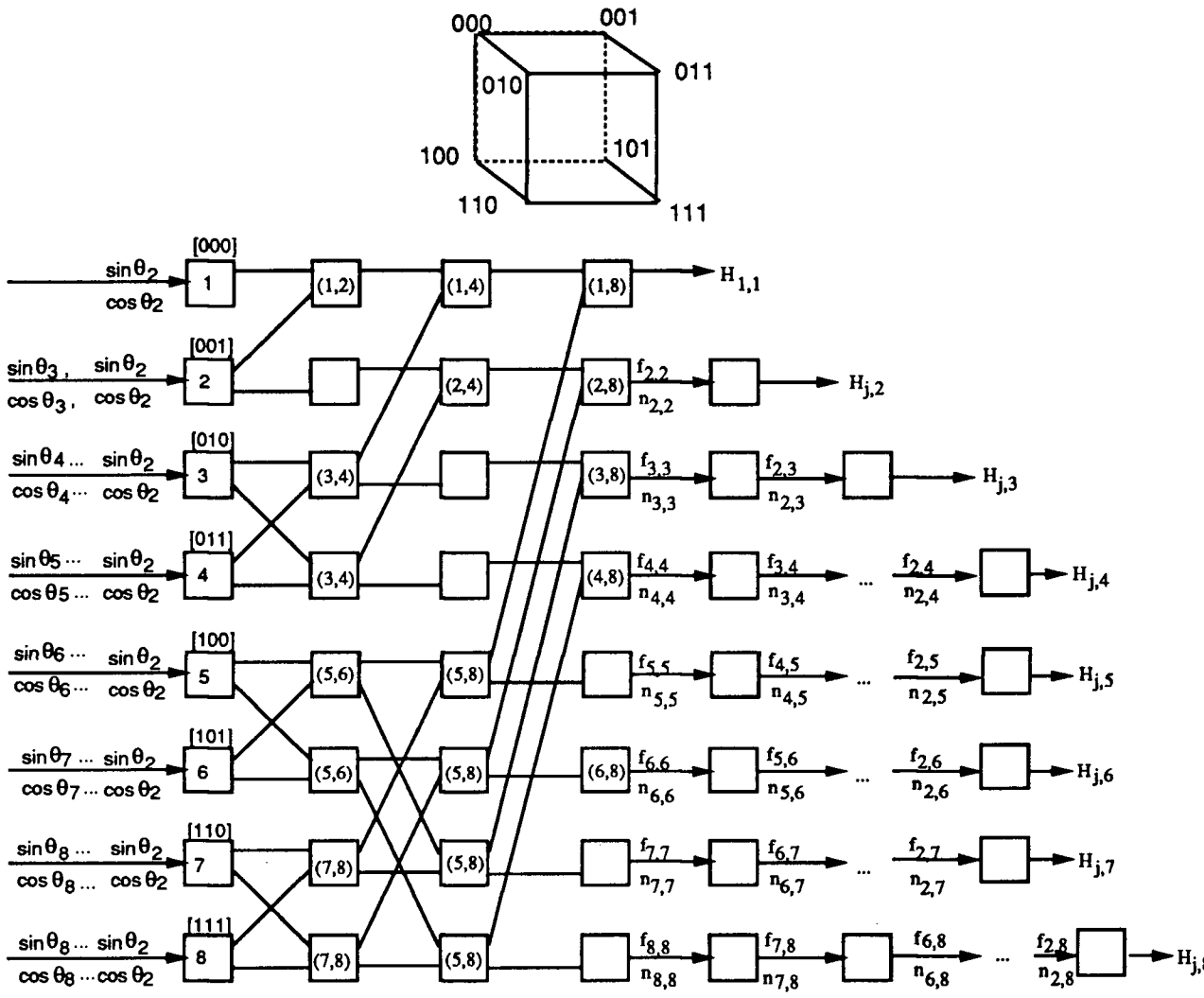

Figure 4.7: N=8 processor cube configuration. Adapted from Amin-Javaheri 1988.

complexity is not increased (the cpus would be idle otherwise), but the communication complexity is minimised to $2\lceil \log_2 n \rceil$ time rather than the $\sum_{i=1}^{\lceil \log_2 n \rceil} i$ cost normally incurred by the hypercube (e.g. if (5,8) in Fig. 4.7 were to be calculated only in cpu 5, it would have to be passed to cpus 1,2,3 and 4 in three hops). Data has to be passed bidirectionally, so there is a factor of 2 in the communication cost (assuming only one bidirectional link between any pair of cpus). Once the $f_{j,j}$ and $n_{j,j}$ of equation 2.8 have been calculated (see Fig. 4.7), $f_{i,j}$ and $n_{i,j}$

($i < j$), and column j of the inertia matrix are calculated by the jth cpu in O(n) time without any further communications.

A direct comparison of computational complexity is difficult with Amin-Javaheri's algorithm since the cost is not given as a function of multiplies and adds. The computational complexity polynomial (A) in Table 4.8 gives the number of 1 μs cycles. Since Amin-Javaheri and Lee have very similar algorithms, it is expected that Lee's complexity is also representative of Amin-Javaheri's algorithm (e.g. assuming multiplies and adds are 1 μs each, for $n = 8$ the compute times are 473 cycles for Lee and 512 cycles for Amin-Javaheri, which are very close).

4.7.3 Fijany and Bejczy

Fijany, like Amin-Javaheri, presented an algorithm for just the inertia matrix [Fijany 89]. He used the RDA in a similar manner to Lee and Amin-Javaheri to compute the A_i^0 and other composite body variables in O($\lceil \log_2 n \rceil$) time. Following this, Fijany computed the inertia matrix terms in O($\lceil \log_2 n \rceil + 1$) rather than the O($n$) time of Lee, by using the RDA algorithm to compute $P_{i,j}$, the vector from hinge i to hinge j. $P_{i,j}$ is used in the Composite Rigid-Body Spatial algorithm (but not in the Composite Body algorithm) according to the following equations:

$$\begin{aligned} \text{for } i &= n, \ldots 1 \\ \text{for} &\quad j = i-1, \ldots 1 \\ &\quad P_{i,j} = P_{i,j+1} + P_{j+1,j} \qquad (4.31) \\ &\quad \mathcal{M}_{j,i} = z_j^T (n_i + P_{i,j} \times f_i) \qquad (4.32) \end{aligned}$$

The computational complexity of equations (4.31) and (4.32) is O($\lceil \log_2 n \rceil$) on a hypercube and O(1) on an $\frac{n(n+1)}{2}$ mesh, respectively.

The complete algorithm is mapped to a 2d $\frac{n(n+1)}{2}$ nearest neighbour mesh. By choosing this architecture, the RDA communication cost increases to O($\sum_{i=1}^{\lceil \log_2 n \rceil} 2^{i-1}$), as the data must be passed from neighbour to neighbour in order to imitate the hypercube structure. This increases

the communication cost of the RDA by 150% [Fijany 89], but the communication architecture can be reused for matrix solving (whereas the n cpu hypercube architecture does not implement a systolic solver as efficiently as a mesh architecture).

4.7.4 Hwang, Bae, Haug and Kuhl

The orthogonal complement algorithm proposed by Bae [Bae 88a] has been implemented on an Alliant FX/8 multiprocessor, an n cpu shared memory bus architecture. Because the formulation deals with closed loop constraints, which require the identification of dependent and independent coordinates and the maintenance of constraints, the computations are much more complex than those for an open loop chain. As a consequence, the computational complexity polynomial was not given by Hwang and Bae [Hwang 88].

To directly compare the Hwang's algorithm we have estimated the performance of the algorithm for a single open chain in Table 4.9, making a number of assumptions. To take advantage of the fact that the variables are all referred to the inertial frame (and therefore A_i^0 must be calculated) we have assumed a hypercube architecture and the RDA algorithm for calculating A_i^0 and K_i. An $\frac{n(n+1)}{2}$ mesh has been added to solve the matrix systolically using Liu's algorithm, and the force vector is assumed to be simultaneously calculated on another n cpu cube (and therefore will not be counted in the complexity estimate). Table 4.9 presents the operations described in Chapter 2.

The calculation of A_i^0, $x_{i,0}^0$ and K_i are performed using the RDA algorithm. Operation 10, $K_i = \sum_{i=1}^{n} \hat{M}_i$, requires that $\hat{M}_i$ be sent to neighbouring cpus. Only 10 values need to be sent in each direction for each data transfer of operation 10 because of the symmetry in $\hat{M}_i$. It is possible to reduce the computations of operation 12 to $(6\times, 5+)$ by distributing the $(B_i^T K_i)$ and B_j vectors throughout the mesh. This, however, would require a complex algorithm to distribute the vectors, followed by another phase in which the inertia matrix elements must be brought to the edge of the mesh before the solver can begin. It is thus not worth the extra cost in communications to do this unless data communications becomes much cheaper, and so

#	operation	cost	comms no. floats	cpus
1	A^i_{i-1}	$4\times$	0	n
2	A^0_i	$\lceil\log_2 n\rceil(27\times,18+)$	$6\lceil\log_2 n\rceil$	n
3	$A^0_i J^i_i A^0_i$	$36\times,24+$	0	n
4	$v_i = A^0_{i-1} l^{i-1}_{i-1,i} - A^0_i k^i_{i,i}$	$18\times,15+$	3	n
5	$x^0_{i,0} = x^0_{i-1,0} + v_i$	$3\lceil\log_2 n\rceil+$	$6\lceil\log_2 n\rceil$	n
6	$m_i x^0_{i,0}$	$3\times$	0	n
7	$m_i(\tilde{x}^0_{i,0})^2$	$6\times,3+$	0	n
8	$J^0_i - m_i(\tilde{x}^0_{i,0})^2$	$9+$	0	n
9	$B^T_i = [(\tilde{x}^i_{i,0} + \tilde{k}^i_{i,0})z^0_{i,i} \;\; z^0_{i,i}]^T$	$6\times,8+$	0	n
10	$K_i = \sum_{j=i}^n \hat{M}_i$	$10\lceil\log_2 n\rceil+$	$20\lceil\log_2 n\rceil$	n
11	$B^T_i K_i$	$36\times,30+$	0	n
12	$(B^T_i K_i)B_j$	$n(6\times,5+)$	$6(n-1)$	n
13	$\mathcal{M}^{-1}$	$(7n-5)\times,+$	$7n-5$	$\frac{n(n+1)}{2}$

Table 4.9: Estimated computational complexity for Hwang's algorithm
operation 12 is assumed to be executed by the n cpu cube architecture in O(n) complexity.

4.7.5 Complexity Comparison

Fig. 4.8 is a plot of the computational complexity for forming the inertia matrix alone (an + is 0.5 μs, half of the cost of a $\times$, using an Inmos T800 transputer). The performance of the proposed O(n) inertia matrix algorithm (W) is linear. The algorithm is more efficient at lower degrees of freedom than the other algorithms in Table 4.8. The exact crossover point is sensitive to assumptions made about the cost of implementation for each method. The implementation assumptions used in Section 4.7 indicate that the proposed algorithm is more efficient for up to 13 degrees of freedom (excluding communication costs). When the computational cost of the solver is added to the cost of the inertia matrix algorithm (Fig. 4.9), W apeears to be more efficient for chains of up to 15 degrees of freedom. It is assumed in algorithm W that the Jw and $J\dot{w}$ computations are executed by row 1 cpus in algorithm W, and so their cost is included in the $\mathcal{M}$ estimate. Algorithm F is included in the figure by adding the complexity of Liu's solver.

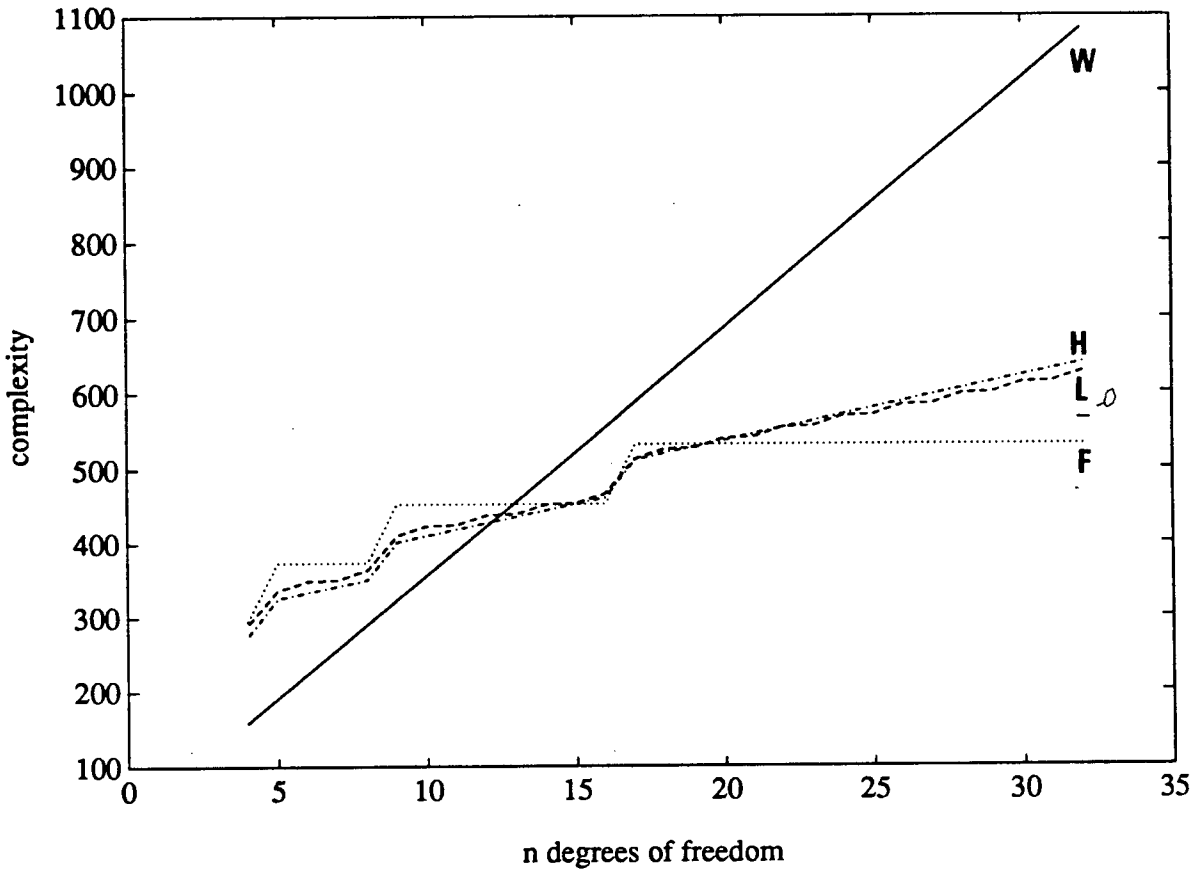

Figure 4.8: Computational complexity for the parallel algorithms of Table 4.8, assuming only $\mathcal{M}$ is calculated. L - Lee, F - Fijany, H - Hwang, W - Wong (proposed algorithm).

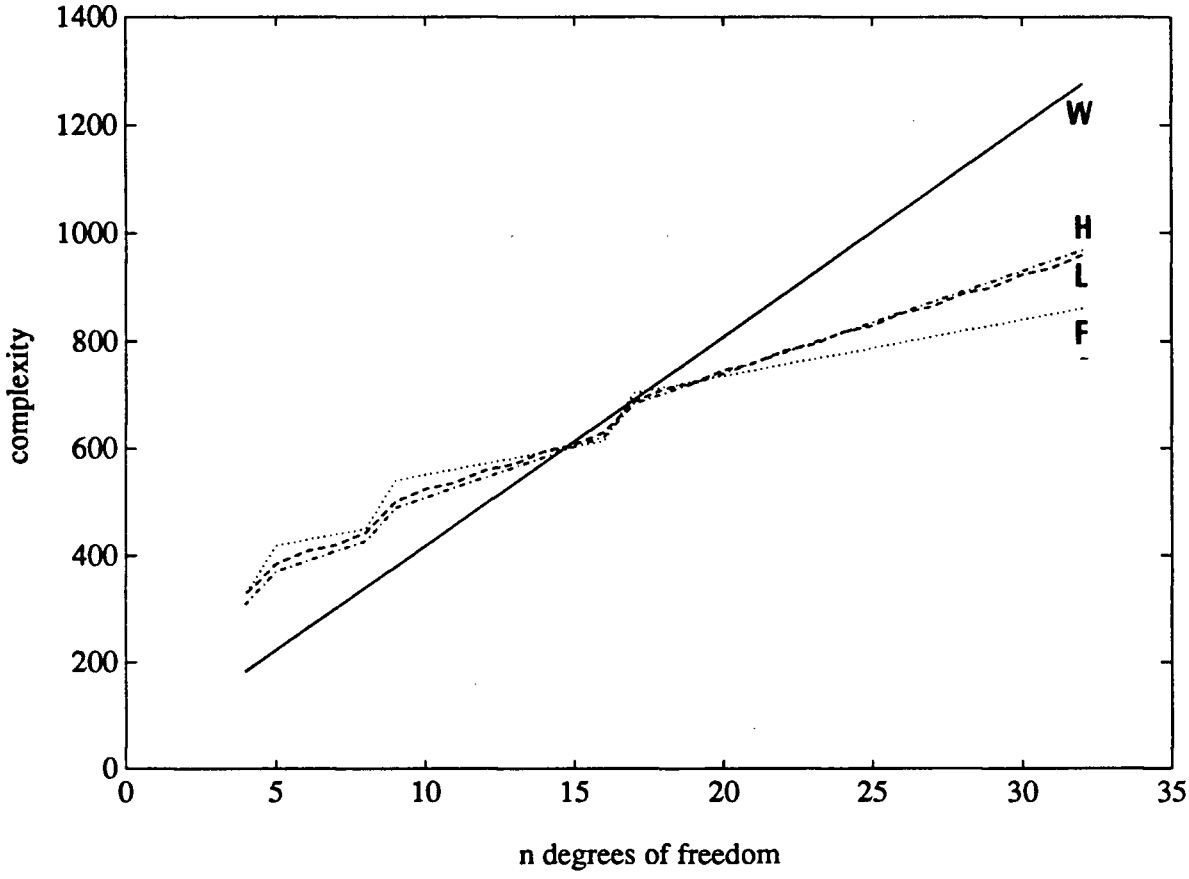

Figure 4.9: Computational complexity for the parallel algorithms of Table 4.8, including $\mathcal{M}$ and feedforward solver. L - Lee, F - Fijany, H - Hwang, W - Wong (proposed algorithm).

4.8 Summary

The orthogonal complement formulation derived in Chapter 3 has been parallelised. New parallel algorithms for the inertia matrix and the force vector have been developed, producing $O(n)$ parallel algorithms for single chain mechanisms which are conceptually very simple.

The proposed algorithms map onto a 2d $\frac{n(n+1)}{2} + n$ nearest-neighbour mesh architecture (an extra cpu is used to connect to the host). The connections are wrapped around to form a 3d cone-like architecture (this aids in the data shifting phase of the algorithm, when data must be routed to the edge of the array in preparation for the matrix solver). The algorithms make full use of the computational resources i.e. the pipeline used by the force vector algorithm and the triangular mesh used by the inertia matrix algorithm are reused by the systolic matrix solver. The systolic feedforward solver is very efficient, and does not require the storage and reordering of the intermediate results between factorisation and backsubstitution phases, which is required by the algorithm proposed by Liu.

In comparison to other parallel dynamics formulations, which are of $O(\lceil \log_2 n \rceil)$ or $O(\lceil \log_2 n \rceil + n)$ complexity, the proposed $O(n)$ formulation is more efficient for mechanisms of 15 degrees of freedom or less (this will vary, however, depending on the assumptions made for each of the implementations). The $O(\lceil \log_2 n \rceil + n)$ algorithms are less efficient for low numbers of degrees of freedom because of the high initial cost of calculating the rotation matrices that refer each body's variables to the inertial frame. The proposed formulation uses nearest-neighbour communications routing, which makes expansion of the architecture to accomodate larger n very simple. The $O(\lceil \log_2 n \rceil + n)$ algorithm proposed by Lee and Amin-Javaheri uses a hypercube architecture, which is not amenable to VLSI integration, and requires relatively complex communication strategies to minimise the communication cost (this not only includes redundant computations in source and destination cpus, as discussed before, but also includes the addition of extra communication links when $n > 16$). The $O(\lceil \log_2 n \rceil)$ algorithm proposed by Fijany uses a mesh architecture, but extra communication costs are incurred when the RDA algorithm is implemented. One drawback of the formulation proposed here is that the $\bar{H}^T$ multiplication

phase in the inertia matrix and force vector algorithms requires data to be sent in the reverse direction (west to east and north to south) compared to that of the previous phase (when $[\dot{L}\ m\ddot{p}]$ and $[C\ \rho \times C]$ were calculated) (east to west and south to north). This means bidirectional links are necessary, in place of the simple unidirectional links which have been proposed by the designers of VLSI systolic arrays.

Chapter 5

The Real-Time Excavator Simulator

5.1 The UBC Teleoperation Project

This chapter describes the development of the real-time simulator for a Caterpillar 215B heavy-duty excavator. The 215B is typically used in primary industries such as forestry, mining, and construction. The objective of the UBC teleoperation project is to convert this type of machine, with minimum changes, to a semi-automated, human-supervisory control system. The simulator developed in this thesis is a fundamental part of this effort. Semi-automated control uses both human supervision and efficient robotic power for tasks in which these characteristics can simplify operations, such as tree harvesting and materials handling. Present operation of excavators and other similar hydraulic equipment is heavily influenced by the skill and smoothness of the human operator. Computerised joystick control can improve the ergonomics of the present lever per joint arrangement by, for example, having the joystick mapped to Cartesian-like workspace (the world coordinates are inertially measured x, y, and z) rather than joint space. The operator thus controls the end effector of the excavator, letting the computer calculate the control signals needed to achieve the desired endpoint motion. This design change can improve the ease of use, which should decrease the time necessary to attain a high level of skill, and simplify the low level decision making required from the operator.

The excavator is modeled as a large system consisting of two subsystems – a complex hydraulic actuation system, and a rigid multibody mechanism. The multibody dynamics is represented as a four link mechanism forming a single chain, with each link having one rotary degree of freedom. Although most heavy-duty manipulators use piston hydraulics with a complex arrangement of coupled hoses and pressure sources, for this thesis we have simulated simple rotary hydraulics.

This is because the true actuation dynamics are highly nonlinear and interconnected, with a mixture of fast and slowly changing state variables, causing the system to be mathematically stiff and inefficient. Overcoming the problem of stiffness, so that the integration technique for the hydraulics can be matched to the integration algorithm used by the multibody dynamics, is a thesis topic outside the scope of this thesis. Another participant in the project has simulated the true hydraulics equations [Sepehri 90], and this work is currently being integrated into the simulator.

As the excavator is a single chain mechanism, the $O(n)$ parallel algorithms developed in Chapter 4 are suitable for the simulator. In the next section, the mechanism is described based on information obtained from the specifications of the Caterpillar 215B and from [Sepehri 90].

The discussion then focuses on the hardware requirements necessary for the simulator, as determined by the algorithms of Chapter 4 and by the computer architecture needed to integrate other components of the simulator, such as the graphics display and the controller.

Next, the implementation of the simulator is discussed. This includes the organisation of a parallel fixed stepsize Runge Kutta integration algorithm, the development of equations for simple rotary hydraulic actuators, and the organisation of the input and output between the dynamics simulator and the display and joystick devices.

Following this, some experiments will be presented which demonstrate the operation of the simulator, and its use in experiments on joint and resolved mode joystick control. A timing and complexity comparison is made between single and multiple cpu versions of the simulator as a measure of the effectiveness of the implementation.

5.2 A Description of the Caterpillar 215B

The Caterpillar 215B is a tracked excavation mechanism with four rotary links named the cab, boom, stick and bucket (Fig. 5.1). The cab sits upon the base and swivels about a vertical axis. The operator sits within the cab, controlling the excavator with a pair of two degree of freedom joysticks. Each degree of freedom controls one link. The boom is the anatomical equivalent to

the upper arm of the human body; it rotates about a horizontal axis and provides the ability to lift the bucket. The stick is analogous to the forearm, and is used with the boom when reaching. The bucket is the end link which is used to dig or carry the load, and is analogous to the human hand. Both the stick and the bucket also rotate about horizontal axes.

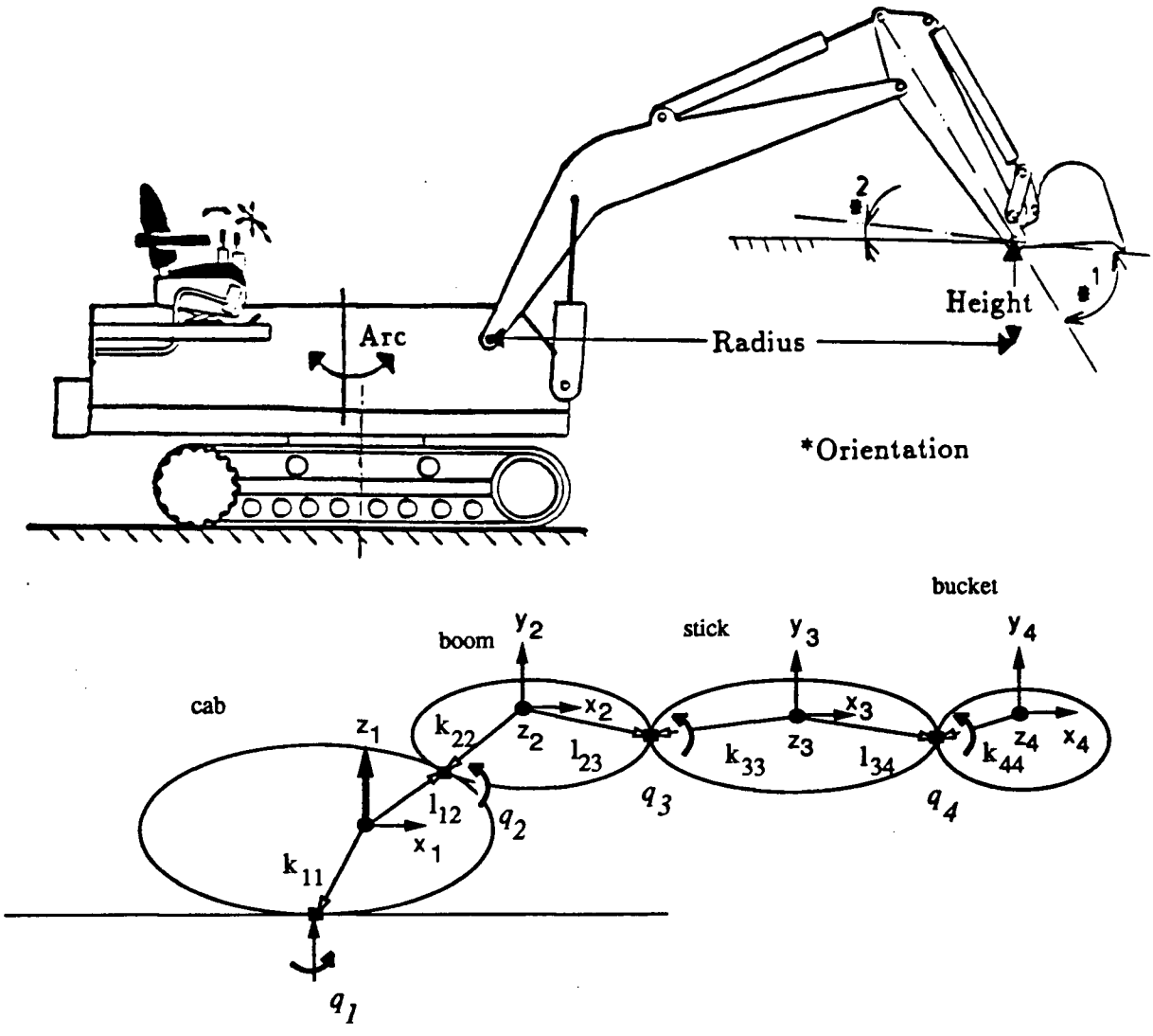

Figure 5.1: Initial coordinate arrangement for Caterpillar 215B and resolved mode parameters

Fig. 5.1 is a diagram showing the coordinate frames attached to the mechanism in its initial

position. The coordinate frames are attached to the centre of gravity of each body. The *cab*, *boom*, *stick*, and *bucket* are labelled links 1 to 4, respectively. The parameters used for the simulation are in metric units of kg for mass, m for length, $kg \cdot m^2$ for inertia, and deg for angle:

$$
\begin{aligned}
m_1 &= 8031, \quad k_{1,1}^T = [-1.05 \;\; 0.04 \;\; -0.5], \quad l_{1,2}^T = [0.35 \;\; -0.12 \;\; 0.5] \\
m_2 &= 1830, \quad k_{2,2}^T = [-2.3 \;\; -0.2 \;\; 0], \quad l_{2,3}^T = [2.9 \;\; -0.2 \;\; 0] \\
m_3 &= 688, \quad k_{3,3}^T = [-0.9 \;\; -0.1 \;\; 0], \quad l_{3,4}^T = [0.9 \;\; -0.1 \;\; 0] \\
m_4 &= 512, \quad k_{4,4}^T = [-0.5 \;\; -0.5 \;\; 0]
\end{aligned}
$$

The twist angles are $\alpha_1 = 0$, $\alpha_2 = 90$, $\alpha_3 = 0$, and $\alpha_4 = 0$, and the inertia matrices are:

$$
J_1^1 = \begin{bmatrix} 8000 & 0 & 0 \\ 0 & 8000 & 0 \\ 0 & 0 & 15700 \end{bmatrix}, J_2^2 = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 15400 & 0 \\ 0 & 0 & 15400 \end{bmatrix},
$$

$$
J_3^3 = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 600 & 0 \\ 0 & 0 & 600 \end{bmatrix}, J_4^4 = \begin{bmatrix} 130 & 0 & 0 \\ 0 & 130 & 0 \\ 0 & 0 & 130 \end{bmatrix}
$$

5.3 Architecture and Hardware Considerations

Robot systems often have many diverse tasks that can run in parallel, such as the dynamics (inertia matrix and force vector formation), the inverse kinematics, control algorithms, and sensor fusion algorithms (e.g. image understanding and touch sensing). Although some tasks such as image processing operate well using an SIMD (Single Instruction, Multiple Data) architecture, the simultaneous operation of many of the above tasks necessitates an MIMD (Multiple Instruction, Multiple Data) approach, so that different computational structures can be accomodated. As an example, image processing and dynamics are two tasks that can run in parallel, but may be processed by different computational units. Expandability, connectivity and global access to sensors are also important considerations. Expandability is an issue because mechanisms

with other topologies will eventually be simulated. Connectivity must be considered so that communication latency is minimised.

These requirements have directed our choice of computer architecture to a crossbar+bus based multiprocessor system. The crossbar, a 2d mesh of switches providing a non-conflicting path between any pair of processors, has a high data bandwidth for local task computations in which cpus are interconnected by the crossbar to perform a common task. The bus provides a path between clusters (each cluster forming around a small crossbar network and performing a different task e.g. dynamics). The architecture chosen for this project consists of a 32 bit global VME bus using a SUN 3/280 work station as the file-serving host running UNIX. A cluster of cpus made up of thirty four Inmos T800 transputers is connected together in a 2d mesh and to the host by a crossbar network. The crossbar is used to imitate the mesh as it has great flexibility in connectivity and can adopt architectures that are suited to many tasks. By forming a mesh the expansion to larger n is easily accomplished by adding new columns of n cpus.

For the excavator simulation, the crossbar forms a cluster of fourteen transputers configured as a 2d triangular mesh (Fig. 5.2). Transputer $\text{cpu}_{0,0}$ connects the mesh to both the host and to another transputer which controls the spool response. The control transputer also connects to the display and joystick subsystems.

A Silicon Graphics IRIS 4D70GT is used to present a real time, hidden-surface-removed display of the external environment as viewed through the front window of the excavator cab. The IRIS is connected to the control transputer via an RS232C cable, which is then converted to the RS422 standard for the transputer link. Two joysticks are connected by an A/D board to the IRIS. In normal operation (JOINT mode), each joystick has two degrees of freedom, while in Cartesian-like endpoint control, one joystick has three degrees of freedom and the other has one. Data from the joysticks is read by the IRIS computer and sent on the RS232C/RS422 link to the control transputer. Data is also passed from the mesh via $\text{cpu}_{0,0}$ through the control cpu to the IRIS.

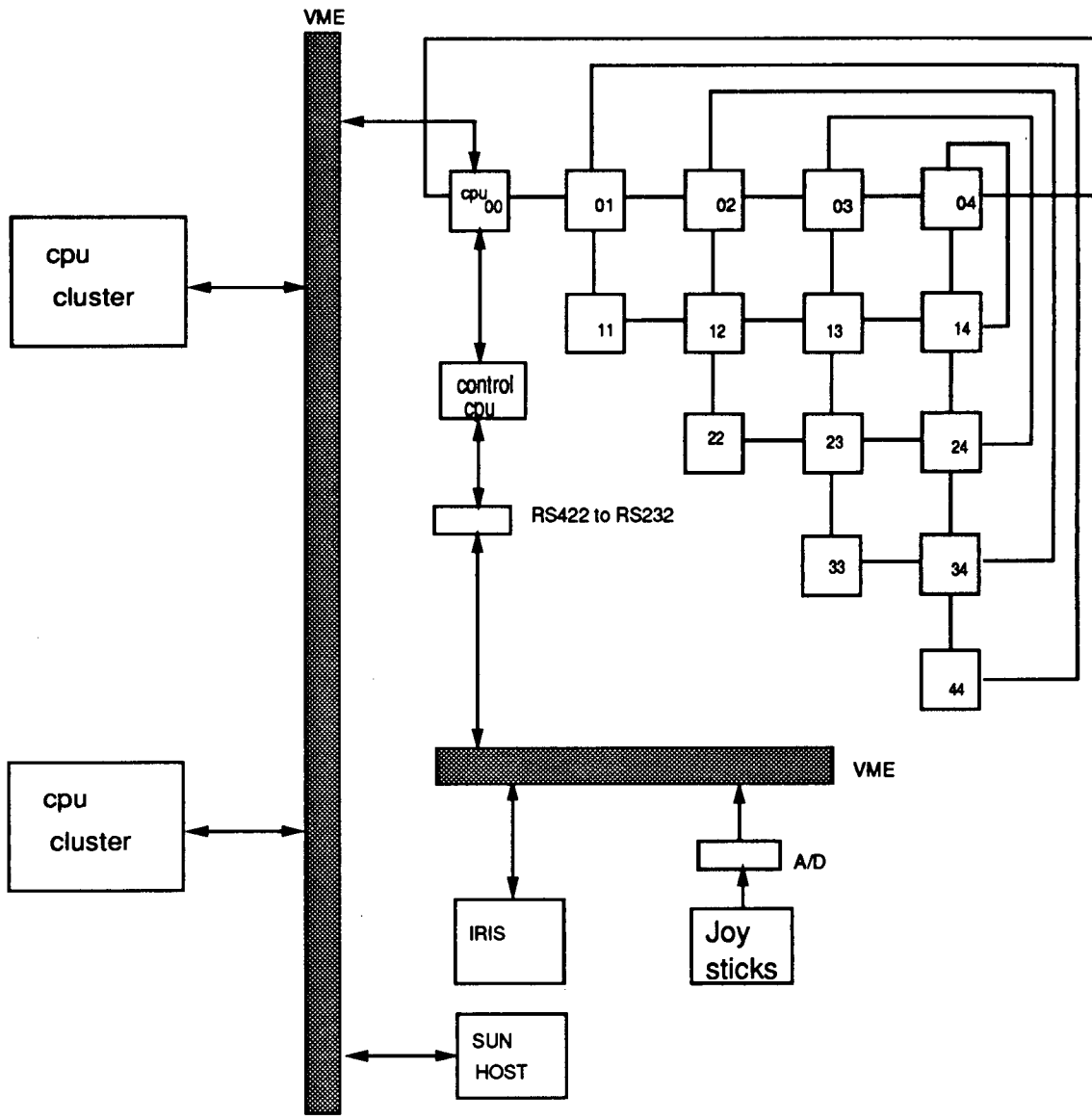

Figure 5.2: Computer architecture for the excavator simulator

5.4 Implementation of the Simulator

5.4.1 Multibody dynamics and distributed Runge Kutta

In Fig. 5.2, row 0 is used to calculate the force vector f_v (3.27 to 3.31, and 3.34), and rows 1 through n (n=4) are used to calculate the inertia matrix (3.17). $\text{cpu}_{0,0}$ provides a link to the

#	row 0	rows 1 to n
	Do $j = 1$ to 4 {	
1	Calculate actuator torque τ_a	no activity
2	Calculate $[\dot{L}\ m\vec{p}]$ (eqs. 3.27 to 3.31 sec. 4.3)	Calculate $[JC\ M(\rho \times C)]$ (eq. 3.17, sec. 4.2))
3	Multiply by H^T to get f_v (eq. 4.8, sec. 4.4.1)	Multiply by H^T to get $\mathcal{M}$ (eq. 4.8, sec. 4.4.1)
4	Move $b = f_v + \tau_a$ to $\text{cpu}_{0,4}$ (Fig. 4.5)	Move $\mathcal{M}_{i,j}$ to $\text{cpu}_{1,i}$ (Fig. 4.5)
5	Solve for $\ddot{\theta}$, results reside in $\text{cpu}_{0,0}$ (Fig. 4.4)	Solve for $\ddot{\theta}$ (Fig. 4.4)
6	Move $\ddot{\theta}_i$ from $\text{cpu}_{0,0}$ to $\text{cpu}_{0,i}$	no activity
7	Integrate $\ddot{\theta}_i$ to get $\dot{\theta}$ and θ (5.1), and then transmit θ_i from $\text{cpu}_{0,i}$ to $\text{cpu}_{1,i}$.	no activity Receive θ_i from $\text{cpu}_{0,i}$
	}	
8	Send θ_i from $\text{cpu}_{0,i}$ to $\text{cpu}_{0,0}$ and then to IRIS	no activity
9	Send x_i from control cpu to $\text{cpu}_{0,0}$ and then to $\text{cpu}_{0,i}$	no activity

Table 5.1: Tasks required for the dynamics simulation

host for program development, down loading, and data storage.

Table 5.1 lists the tasks performed by row 0, and rows 1 through 4. Tasks 1 to 7 are evaluated in a loop four times, and then the θ are sent out to the display in task 8. The spool valve openings x, calculated by the control transputer from the joystick data, are sent to $\text{cpu}_{0,0}$ and on to the array (tasks 8 and 9).

The calculation of the actuator torques τ_a will be discussed later. Tasks 2 and 3 use the algorithms described in sections 4.2, 4.3, and 4.4 of Chapter 4 to derive $\mathcal{M}$ and f_v. In this implementation of $\mathcal{M}$, the cross product $(\rho^i_{i,j} \times c^i_{i,j})$ is calculated by each cpu before the dot product (method 2 in section 4.3), since the cost of communications using transputers is not small. Task 4 is the data shifting phase discussed in section 4.5 and shown in Fig. 4.5, which moves the elements of b and $\mathcal{M}$ to the edge of the array. Task 5 uses the Jainandunsing systolic solver algorithm to calculate $\ddot{\theta}$. Task 6 redistributes the $\ddot{\theta}_i$, since the values are resident in $\text{cpu}_{0,0}$ after the solver has finished.

Task 7, the Runge Kutta integration algorithm, is distributed among the cpus of row 0. The differential state vector $\dot{y}$ for link i resides in $\text{cpu}_{0,i}$, and consists of four variables: $\ddot{\theta}_i$, $\dot{\theta}_i$, and the input and output differential pressures $\dot{p}_{in(i)}$ and $\dot{p}_{out(i)}$ of the rotary hydraulic actuator. At the end of task 6 in Table 5.1, the $\ddot{\theta}_i$ are available in $\text{cpu}_{i,0}$. The fourth order fixed step-size Runge Kutta algorithm presented by Press [Press 88] is then used to determine the new $y = [\dot{\theta}_i\ \theta_i\ p_{in(i)}\ p_{out(i)}]$. This is done by evaluating tasks 1 to 7 four times, each time calculating a new $\dot{y}$ from an updated state y for a partial step across the interval $h = 0.01$. At the end of the fourth evaluation, a weighted average of the four estimates of $\dot{y}$ is used to obtain the true y over the interval h. The Runge Kutta algorithm for each link is exactly the same as for the single cpu algorithm, except that it only applies to the states related to link i. Given an initial state y, the algorithm is:

$derivative(y, \dot{y}_x)$;
for($i = 0; i < 4; i++$)
    $y_t[i] = y[i] + \frac{h}{2}\dot{y}_x[i]$;

$derivative(y_t, \dot{y}_t)$;
for($i = 0; i < 4; i++$)
    $y_t[i] = y[i] + \frac{h}{2}\dot{y}_t[i]$;

$derivative(y_t, \dot{y}_m)$;
for($i = 0; i < 4; i++$)
{
    $y_t[i] = y[i] + h\dot{y}_m[i]$;
    $\dot{y}_m[i] = \dot{y}_t[i] + \dot{y}_m[i]$;
}

$derivative(y_t, \dot{y}_t)$;

$$\text{for}(i = 0; i < 4; i++)$$
$$y[i] = y[i] + \frac{h}{6}(\dot{y}_x[i] + \dot{y}_t[i] + 2\dot{y}_m[i]); \tag{5.1}$$

The *derivative*$(y, \dot{y})$ function calculates $\dot{y}$ from y. Task 8 is an output operation occurring once every step h if the state vectors y and $\dot{y}$ are recorded in a file, or once every $\frac{1}{20}$th of a second ($5h$ in this simulation) if the θ_i is sent to the display computer. To send this data out, the θ_i is passed out from row 0 cpus to $\text{cpu}_{0,0}$, and then either sent out to the IRIS for display or to the host for storage. In task 9 x_i is sent to $\text{cpu}_{0,0}$ from the control transputer.

5.4.2 The actuators

The actuators simulated in this thesis are simple rotary hydraulic ones, actuated independently through a valve controlled, hydraulic pressure system.

The voltages v calculated by the control cpu from the joystick commands c are converted to spool openings x by adding a centre deadband of $d = \pm$ 0.005 cm and limiting the range to $\pm$ 0.05 cm (Fig. 5.3). The x values are sent to $\text{cpu}_{0,0}$, and each cpu in row 0 (operation 1 of Table 5.1) reads x_i. A negative sign for x_i indicates a reversal of the flow, which would cause the link to rotate in the reverse direction. Then, the previous time step's pressure p_{in} and p_{out} are limited between P_e (the drain tank pressure) and P_s (the tank pressure). The flows are then calculated as follows:

$$\begin{aligned}
\text{if}(\ x > 0)\ Q_{in} &= k_f x \cdot \sqrt{Ps - p_{in}}; \\
\text{else}\ Q_{in} &= k_f x \cdot \sqrt{p_{in} - P_e}; \\
\text{if}(\ x > 0)\ Q_{out} &= k_f x \cdot \sqrt{p_{out} - P_e}; \\
\text{else}\ Q_{out} &= k_f x \cdot \sqrt{Ps - p_{out}};
\end{aligned} \tag{5.2}$$

where k_f is the valve coefficient. The actuator torque τ_a for the link is then

$$\tau_a = (p_{in} - p_{out})\frac{D_m}{8}G - \beta_m\dot{\theta} \tag{5.3}$$

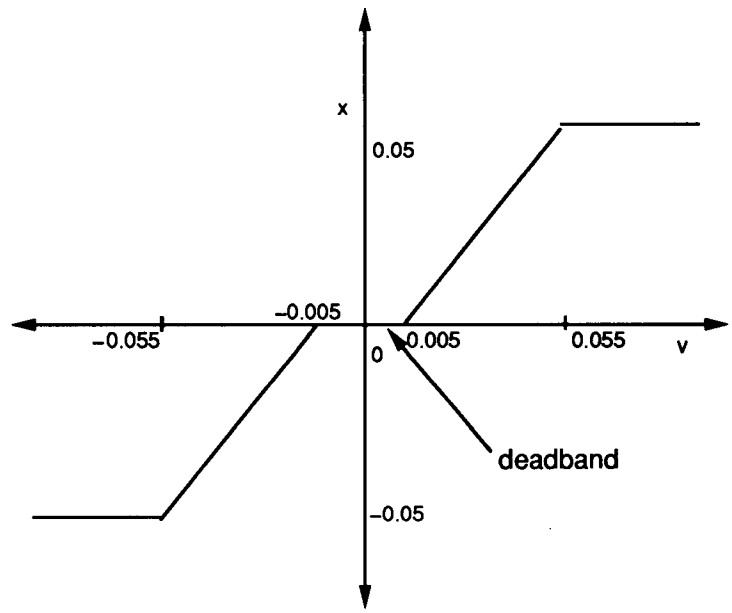

Figure 5.3: Voltage (v) to spool (x) relationship with deadband

where G is the gear ratio of the actuator, D_m is a motor constant, the value 8 is the imperial-to-metric unit conversion factor, and β_m is a velocity damping constant inherent in the actuator. τ_a is added to f_v to get the force vector b.

The differential pressures are proportional to the valve flows Q, the link velocity, and the hydraulic compliance $\frac{1}{\beta_v}$, and may be described as follows:

$$\begin{aligned} \dot{p}_{in} &= (Q_{in} - \dot{\theta} D_m G)\beta_v; \\ \dot{p}_{out} &= (\dot{\theta} D_m G - Q_{out})\beta_v; \end{aligned} \tag{5.4}$$

Thus, given the pressures and the values of the spool displacements, it is possible to calculate the flows, the actuator torques, and the differential pressures. Table 5.2 gives the parameter data for the simulation.

	range/value			
P_e (psi)	0.0			
x (cm)	$\pm$ 0.05			
d (cm)	$\pm$ 0.005			
k_f	26			
D_m	0.7			
β_v	1000			
	cab	boom	stick	bucket
P_s (psi)	4000	7000	4000	4000
G	307	545	220	250
β_m	100000	100000	30000	100

Table 5.2: Parameters used in excavator simulation

5.4.3 The display and joystick interface

The joysticks and the display are interfaced to the dynamics computations by the control transputer and $\text{cpu}_{0,0}$. These cpus control the updating of the display and the actuation of the simulator. Fig. 5.4 and the following sections describe the procedures involved in the interface routines between the IRIS, the control transputer, and $\text{cpu}_{0,0}$.

the IRIS

The IRIS displays the view from the cab of the excavator every $\frac{1}{20}$th of a second using θ_i data received from the interfacing cpus ($\text{cpu}_{0,0}$ and the control transputer). It also samples the joysticks and then sends this data to the control transputer.

The IRIS first reads the joysticks, obtaining four values. In JOINT mode, when each degree of freedom in the joystick represents a link rotation, the values represent angular velocity commands $\dot{\theta}_i$ for the *cab*, *boom*, *stick*, and *bucket* links. In RESOLVED mode, the values represent the linear velocity commands along the radius axis (in a radial direction from the cab), the arc axis (tangential to the arc motion made by the end effector), in the z axis (in a Cartesian sense, the up-down motion of the end effector), and the angular velocity command of the bucket relative to the inertial frame (the ground plane).

The velocity commands c (linear or rotational) are sent from the IRIS to the control transputer through the RS232C line at 9600 baud. The control transputer runs an interrupt driven

buffering routine for receiving the data.

The IRIS then waits for the new values of θ_i determined by the transputer array to be sent back from the control transputer. The θ_i are sent from $\text{cpu}_{0,0}$ to the control transputer which then sends these values on to the IRIS. Since the IRIS receives the angles using an interrupt driven buffer routine, the reception of the angles and the transmission of the joystick values can be accomplished simultaneously.

the control transputer

Once all four velocity command values have been received, the control transputer calculates the voltages (v). If the simulator is in JOINT mode, the commands are $\dot{\theta}_d$, and so the voltage for each link is calculated using a PD controller according to the following equation:

$$v = k_v(\dot{\theta}_d - \dot{\theta}) + k_p(\theta_d - \theta). \tag{5.5}$$

The proportional gains k_p and differential gains k_v for each link of the excavator were chosen by trial and error to give a response similar to that obtained when the same PD controller was used on the excavator. The values chosen are given in the Table 5.3.

If the simulator is in RESOLVED mode, the commands are linear, and so the inverse kinematics are calculated first to get the desired $\dot{\theta}_d$. Once the $\dot{\theta}_d$ are obtained the same PD controller used in JOINT mode is used. The simulator $\dot{\theta}$ values are estimated by a simple differentiation of the θ data sent from $\text{cpu}_{0,0}$ (this data can easily be read from the transputer array if desired, but the communication costs would increase). The θ_d values are estimated by a simple one step integration using the previously estimated value of θ_d and $\dot{\theta}_d$. When v_i has been calculated, it is transformed to the spool value x_i according to Fig. 5.3 and sent on to $\text{cpu}_{0,0}$.

When the angles for the next time instant have been calculated by the array, the data is sent from $\text{cpu}_{0,0}$ to the control transputer.

	cabin	boom	stick	bucket
k_p	2.0	3.0	2.0	2.0
k_v	1.0	2.0	1.0	1.0

Table 5.3: k_p and k_v for PD actuator spool control

the interface transputer $cpu_{0,0}$

The interface transputer $cpu_{0,0}$, acts as the gateway between the display, the joysticks, the spool transputer, and the dynamics simulator. Initially it receives θ_i data from the cpus in row 0 through its *east* link in a pipelined manner. After all four values of θ_i have been collected they are sent through the *south* link to the control transputer and on to the IRIS, where they are used to display the excavator.

5.5 Excavator Simulations

5.5.1 Timing measurements

Single cpu implementation

A single cpu simulation of the excavator was initially performed as a reference from which to compare the parallel simulation. The graphics and joystick subsystems were disabled so that the dynamics computations alone were measured. One cycle, which is comprised of inertia matrix formation (equations 3.17 and 3.23), force vector formation (equations 3.27 to 3.31 and 3.38), simple actuator calculation (equations 5.2 to 5.4), and matrix solution (sec. 3.4), was taken as the best indicator of real-time performance. Results were not stored in a file and the stack pointer of the T800 was initialised to lie in the internal RAM so that all stack operations were faster than if the stack were in external memory.

The results, given in Table 5.4, show that the theoretical and actual implementation percentages for each phase of the computation are very close, indicating that the complexity estimates are relatively correct. Note that the solver was more expensive in practice because an LU solver was actually chosen rather than the Cholesky solver since it was a proven and readily available

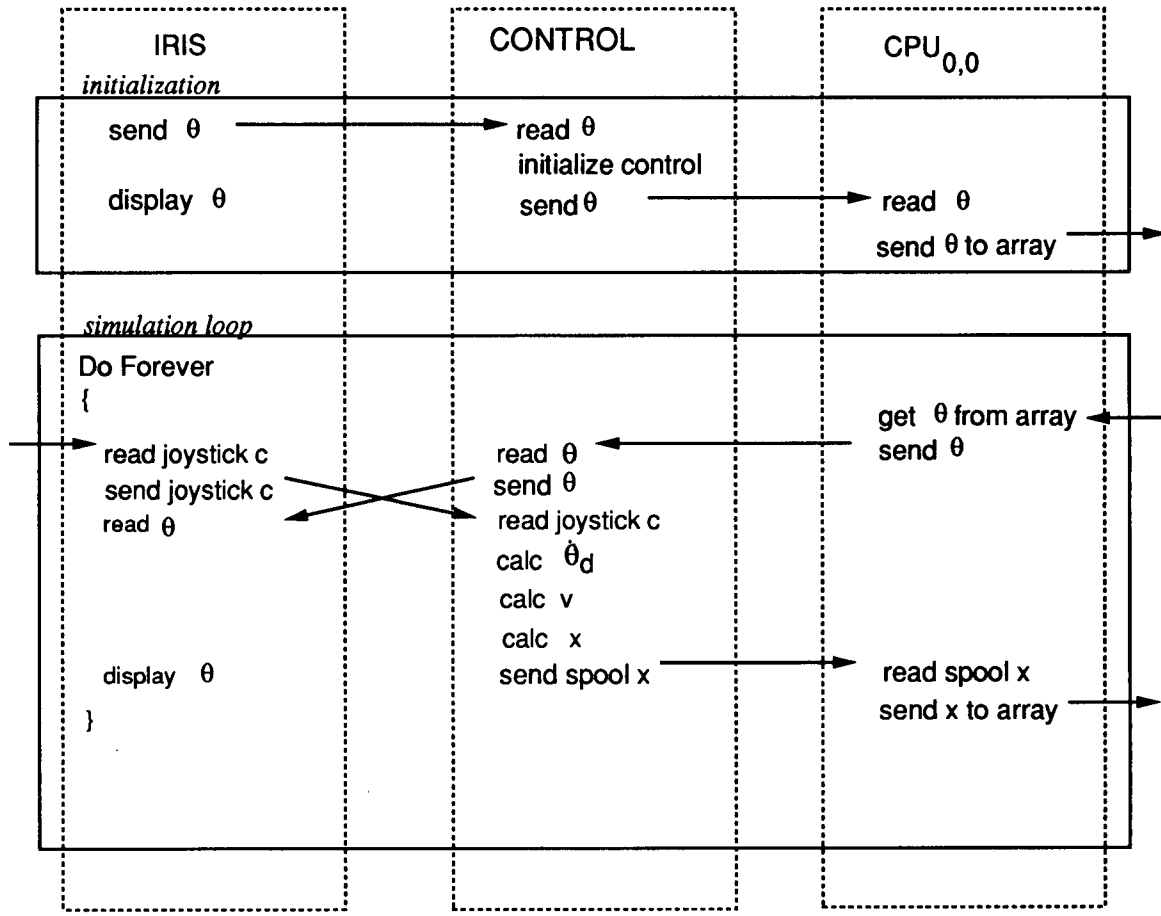

Figure 5.4: The algorithm organisation between $cpu_{0,0}$, the control transputer and the IRIS

subroutine. The implementation is a factor of 7.83 times slower than theoretical expectations (the theoretical estimate was made based on the multiply and add times of the transputer). It is suspected that the extra costs are due to array indexing and subroutine calls, as described in Chapter 3, section 3.5.1.

Overhead costs in computations

Since the actual computational costs exceed the theoretical costs by a factor of 6, it is worthwhile to investigate the source of these delays. The single cpu excavator simulation was used

to time various routines. The algorithm in equation (3.17) for calculating H:

$$
\begin{aligned}
&\text{for } j = 1 \text{ to N } \{ \\
&\quad \text{for } i = j \text{ to N } \{ \\
&\qquad c^i_{i,i} = z^i_{i,i} \quad ; \\
&\qquad \rho^i_{i,i} = k^i_{i,i} \quad ; \\
&\qquad c^i_{i,j} = A^i_{i-1} c^{i-1}_{i-1,j} \quad ; \; i > j \\
&\qquad \rho^i_{i,j} = A^i_{i-1}(\rho^{i-1}_{i-1,j} - l^{i-1}_{i-1,i}) + k^i_{i,i}) \quad ; \; i > j \\
&[U^{-1} Q_{bcs} C]^i_{i,j} = \rho^i_{i,j} \times c^i_{i,j} \quad ; \\
&\qquad H^i_{i,j} = \begin{bmatrix} J^i_i c^i_{i,j} \\ m_i(\rho^i_{i,j} \times c^i_{i,j}) \end{bmatrix} \\
&\quad \} \\
&\}
\end{aligned}
$$

was timed. The cost of an inner loop (calculating $H^i_{i,j}$) was found to be 275 μs, in comparison to a theoretical minimum of 46.5 μs (a factor of 5.9 times slower). The operations involved in the loop can be found in the table below. The overall cost of calculating H was 2260 μs (including the cost of calculating A^i_{i-1}), while the theoretical estimate from equations (3.18) to (3.22) is 319.5 μs ($n = 4$). Thus the overall cost is 7.1 times slower for this phase.

The actual cost for a matrix-vector multiply operation was found to be 65 μs, a factor of 5.4 times slower than the theoretical estimate of 12 μs. It was found that if the 9 multiplies and 6 adds required were calculated using normal variable names (i.e. **array00** rather than

Operation	Actual time (ms)	%	Theory (×,+)	Theory (ms)	%
inertia matrix	3.01	41.28	286×,209+	0.391	42.1
force vector	3.38	46.41	312×,253+	0.439	47.26
actuator	0.35	4.50	44×,24+	0.056	6.03
solver	0.56	7.74	32×,22+	0.043	4.6
Total 1 cpu	7.28	100.0	674×,508+	0.929	100.0
Total $\frac{n(n+1)}{2} + n$ cpus	1.353	100.0	143×,115+	200.5μs	100.0

Table 5.4: Theoretical and real timing estimates of one cycle of simulation on one cpu and the multiprocessor ($\frac{n(n+1)}{2} + n$ cpus).

operation	type	×,+	Theory (μs)	Actual (μs)
$t_1 = \rho^{i-1}_{i-1,j} - l^{i-1}_{i-1,j}$	vec subtract	3-	1.5	18
$t_2 = A^i_{i-1} \cdot t1$	mat-vec multiply	9×,6+	12	63
$\rho^i_{i,j} = t2 + k^i_{i,i}$	vec add	3+	1.5	18
$c^i_{i,j} = A^i_{i-1} c^{i-1}_{i-1,j}$	mat-vec multiply	9×,6+	12	65
$J^i_i c^i_{i,j}$	mat-vec multiply	9×,6+	12	63
$t_3 = \rho^i_{i,j} \times c^i_{i,j}$	cross product	6×,3+	7.5	14
$m_i \cdot t_3$	scalar-vec	3×	3	34

Table 5.5: Timing estimates for calculating H

array[0][0]) the cost was 14 μs, very close to the 12 μs estimate. When indexing was added (**array[0][0]**), the cost rose to 21 μs, due to the extra instructions the microprocessor has to perform to calculate the indexes into **array**. When the array value was addressed as a vector (necessary in order to write a general matrix-vector function that can deal with variable size arrays), explicit integer indexing in the subroutine was necessary to obtain the array element addresses. As a result the cost rose to 56 μs. The cost of calling the subroutine (saving the state on the stack, and returning the state when finished) was found to be 9 μs.

From this investigation, it can be seen that cost reduction can be accomplished by making the matrix-vector functions very specific with respect to the size of the arrays used (e.g. 3×3 matrices). This will reduce the cost of this particular function by 66 %. Direct addressing of array elements is very efficient but the resulting code is very specific and not reusable. Similarly, removing the subroutine structure is not a good solution as the software will lose its generality and also become extremely difficult to debug.

Parallel cpu implementation

The parallel implementation was timed for one cycle, but information for each phase was unobtainable due to the difficulty of extracting times from individual cpus without a global clock as a reference. Another problem is the fact that the transputer array was fully connected into a cone mesh, and as a consequence no links were free to connect to the host and provide timing data. From Table 5.4 it can be seen that the overall time taken was 1.353 ms. This time is

within the period required for real-time operation (2.5 ms for a stepsize of $h = 0.01$ s and a fourth order Runge Kutta). The parallel implementation is a factor of 6.75 times slower than theoretical expectations would indicate, again due to arrays, subroutine calls, and in the parallel case, the cost of communications.

The single/parallel speedup factor was in practice 7.28/1.353= 5.38. Theoretically, this factor was (674×,508+)/(143×,115+)= 4.63. Interestingly, the actual speedup is better than the theoretical speedup in spite of the fact that communications costs were accounted for. Note that in this first implementation, it was decided that explicit (and therefore non-overlapped) communications should be used to simplify the programming and debugging. As a result, no communication costs are hidden. One explanation for the greater speedup factor is that there were fewer subroutine calls and smaller arrays than in the single cpu case. It appears that the cost of subroutine calls and array indexing is at least as costly as communications.

Note that the addition of the IRIS and joystick subsystems slowed down the system so that when the cost of moving data back and forth was included, the time per cycle was increased to approximately 2.1 ms. This is due primarily to the slowness in the RS232C communication protocol. Real-time was achieved in spite of this, however, and the simulator is successfully running as a man-in-the-loop system.

5.5.2 JOINT and RESOLVED mode experiments

Two experiments were performed using the parallel simulator as a substitute for the excavator. The concept of resolved mode teleoperated control was compared to joint mode control. In both cases PD control was used to control the spool value x to approach some desired motion. The experiment was designed by R. Frenette, a staff engineer. The following two subsections outline the results.

JOINT mode control

In joint mode control a profile of desired angular velocities $\dot{\theta}_d$ for each link is specified from a file (rather than read from the joysticks) and fed into the control transputer. The control transputer calculates the appropriate x to achieve $\dot{\theta}_d$ and then feeds x into the array. The resulting values of desired and actual θ and $\dot{\theta}$, and v and x are recorded by the control transputer. Figs. 5.5, 5.6, and 5.7 show the appropriate plots for the *cabin*, *boom*, *stick*, and *bucket*.

Fig. 5.5 shows the desired and actual θ. A steady state error is present due to the lack of an integrator in the PD controller and the deadband in the spool. Note that there is a steady state error in the beginning of the simulation for the boom and stick due to the force of gravity. Gravity loading decreases the actual angle (dotted line) because the link reacts to gravity, pulling the link downward, while the pressure in the hydraulic line is still building up from its initial value of zero. This results in an initial decrease in the angle. Once the line pressure has built up, the link has a steady state error. This initial error is not present in the cabin because its rotational axis is orthogonal to gravity. It is not present in the bucket plot because the bucket has a much smaller gravity load due to its smaller mass and a smaller inertia due to its shape, and as a consequence there is a smaller drop in the angle due to the lack of pressure in the lines. All plots in Fig. 5.5 show a steady state lag at the end of the simulation as the line pressures have already built up to their normal values. This particular error can be eliminated by adding integral control.

Fig 5.6 shows the joint velocities for each of the links. The cabin demonstrates no oscillation at the start since it is orthogonal to gravity and is thus immune to the initial lack of pressure. At the end of the simulation, the velocity of the cabin oscillates due to the inertia of the machine. The inertia causes an overshoot followed by a reaction from the spool, resulting in an oscillatory response. The boom and stick oscillates initially due to gravity loading and the initial lack of pressure. At the end of the simulation the boom and stick velocities are smooth because the deadband in the spool causes the boom and stick spools to be closed. The bucket velocity has a small initial oscillation due to gravity.

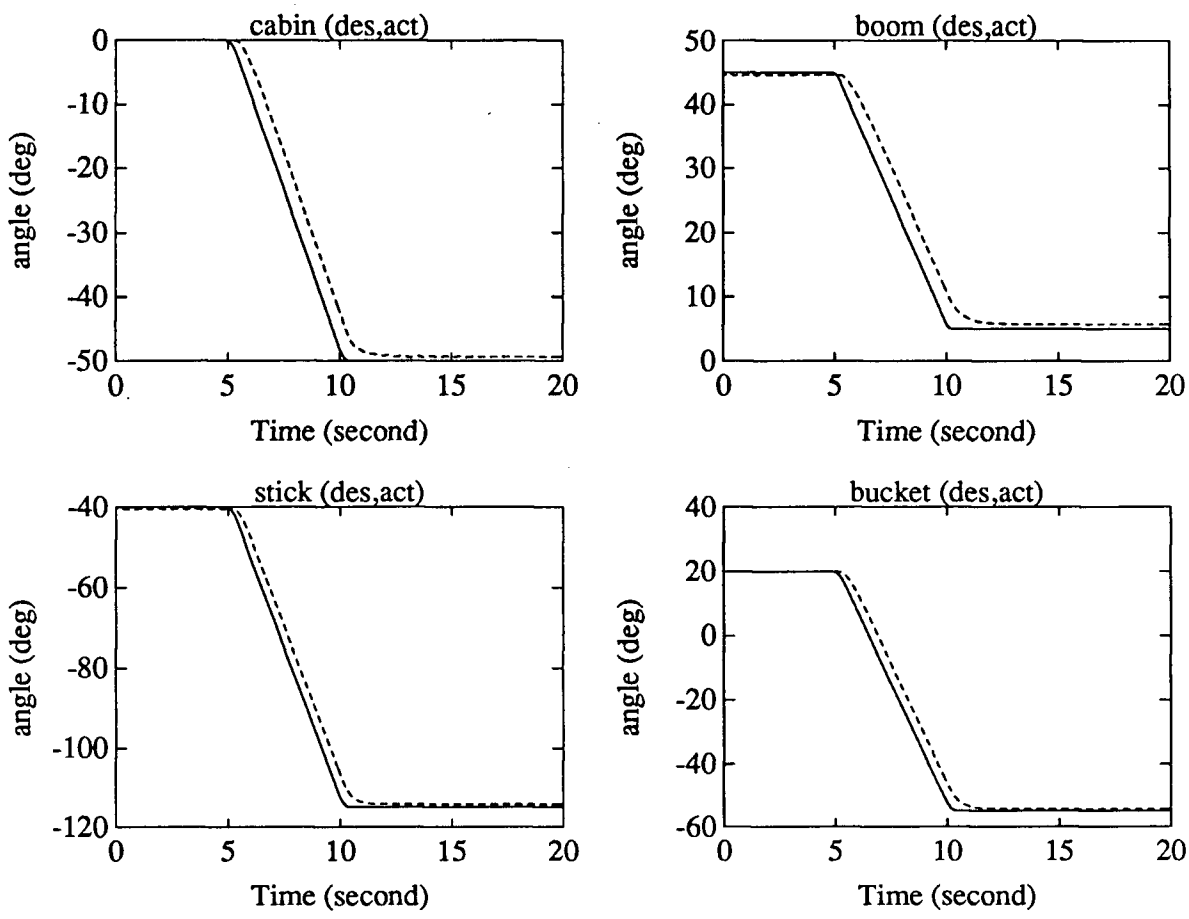

Figure 5.5: JOINT mode θ response in the simulator.

Fig. 5.7 shows the spool (x) and voltage (v, dotted) plots. The v curves are the desired drive voltages, and the x curves include the deadband in the response. The boom v attempts to give a positive drive voltage, but only a zero value of x is produced, because of the deadband. At the end of the simulation, a negative v is desired to close the steady state error, but the deadband causes x to be zero, which results in a zero $\dot{\theta}$.

These plots have not yet been compared to those of the excavator, but initial subjective responses from users of the complete simulator (including the joystick and graphics) indicate that the simulator reponse is felt to be realistic.

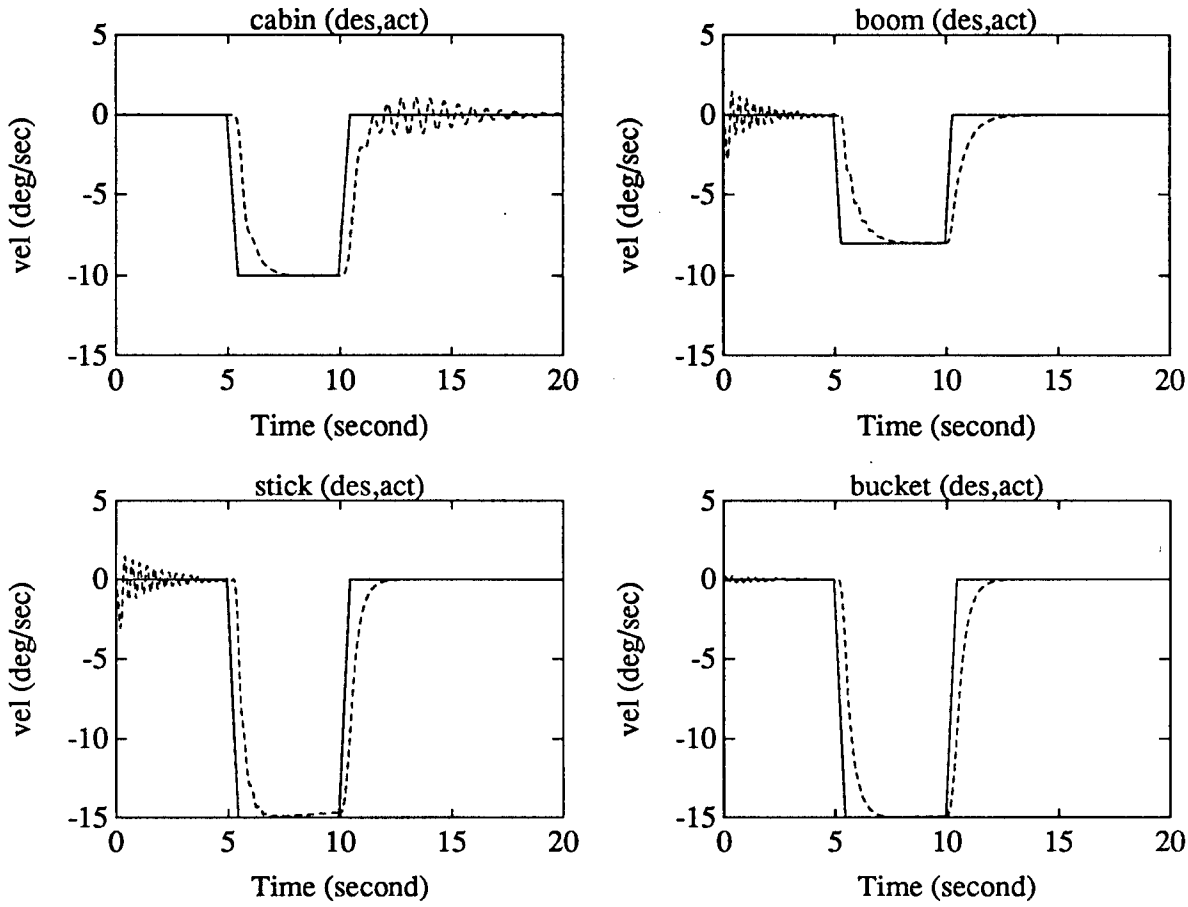

Figure 5.6: JOINT mode $\dot{\theta}$ response in the simulator.

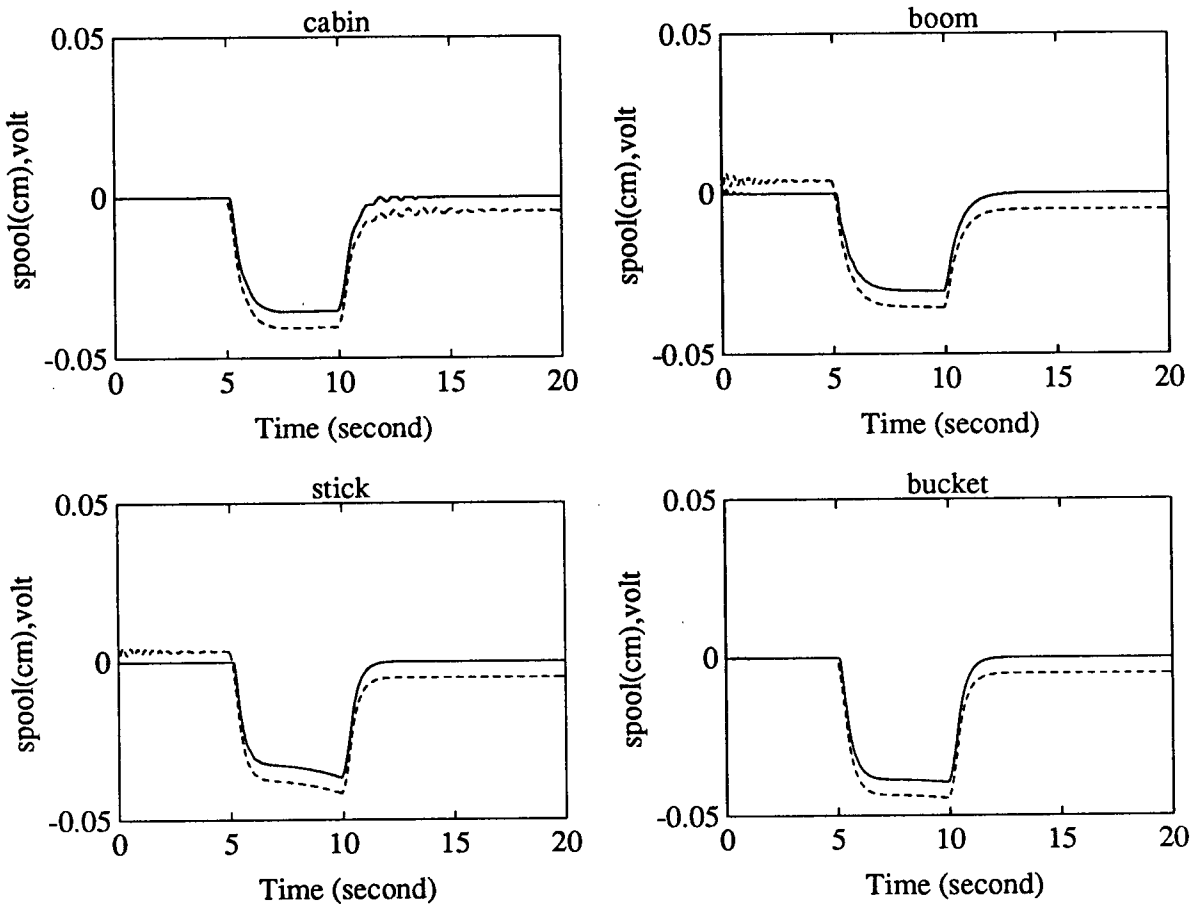

Figure 5.7: JOINT mode v and x response in the simulator.

RESOLVED mode control

In RESOLVED mode, the desired profile represents the linear velocity commands along the radius axis, the arc axis, the z axis, and the angular velocity of the bucket relative to the ground. The setup was similar to the JOINT mode experiment except that inverse kinematics calculations were necessary to derive the $\dot{\theta}_d$ from the linear velocity values. The experiment required an initial 5 seconds of no movement followed by 5 seconds in which the cabin rotated 15 degrees and the end effector moved from a radius of 5m to 6.5m at constant radial velocity while maintaining its original height above the ground. The bucket was required to maintain its angle. The last 5s required all links to maintain their positions. Figs. 5.8, 5.9, and 5.10 show the link reponses.

Fig. 5.8 shows the joint angle reponse. The desired cabin angle (continuous line) is linear. The cabin response shows a significant lag. To maintain a constant end effector height, the boom and stick show a nonlinear joint angle response. As with the JOINT mode experiment, the boom and stick have an initial gravity loading response which when coupled with the lack of initial pressure causes the boom and stick to sag. At the end of the simulation, however, there is a lag due to the deadband in the spool.

Fig. 5.9 shows the desired and actual integrals of the linear velocity curves. The radius, the cabin rotation, and the bucket behave in a similar manner to the results of the JOINT mode experiment, with their initial gravity loads and line pressure buildup responses, followed by final steady state lags. The Z curve indicating the height of the end effector shows an initial sag when the boom and stick sag, followed by a positive error due to the boom and stick lag response. The bucket sags initially due to the initial pressure, but eventually lags the desired response.

Fig. 5.10 shows the desired v (dotted) and actual x (continuous) curves. As in the JOINT mode, the deadband in the x response has a large effect because it causes the spool to remain closed when the link has a steady state error and prevents a reduction in this error by the control algorithm. As the boom and stick extend, the spool opens wider to drive the link faster. This

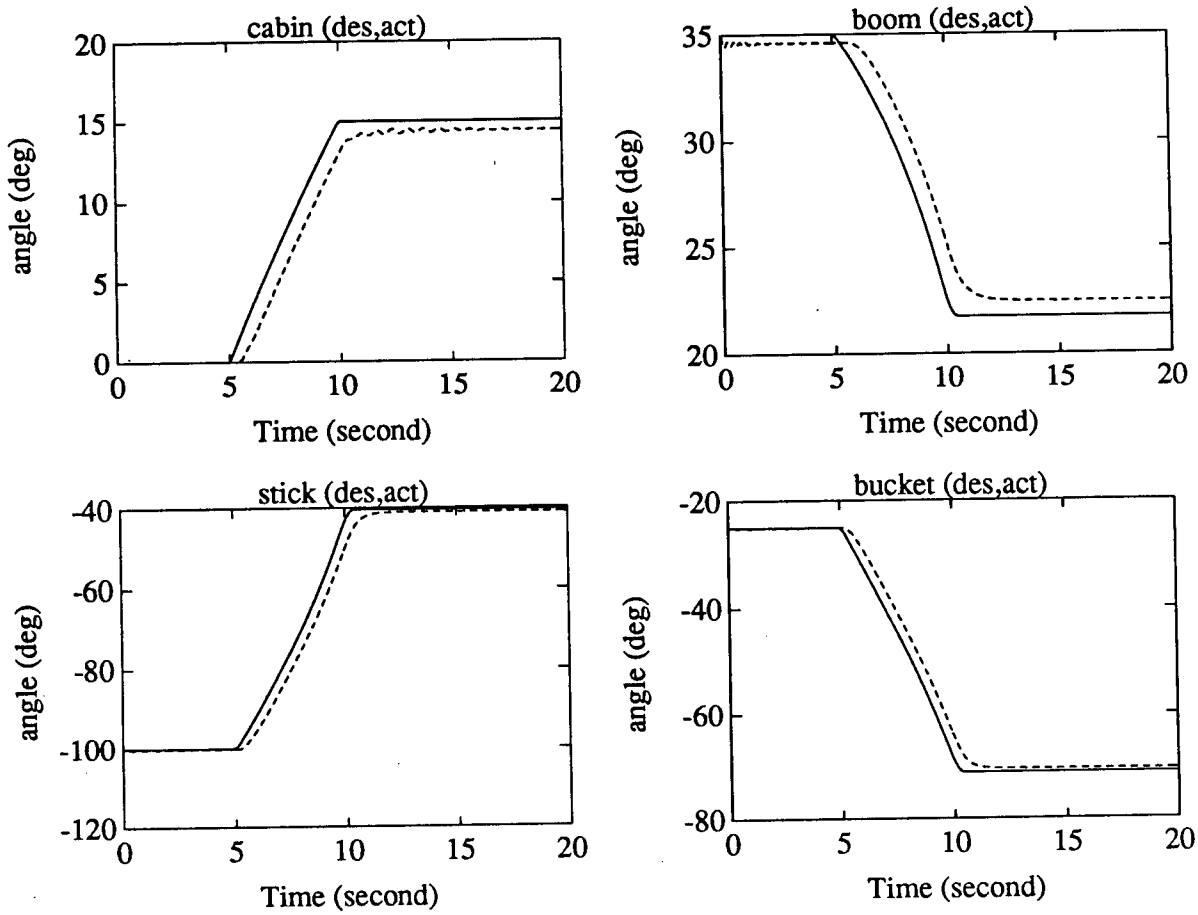

Figure 5.8: RESOLVED mode θ response in the simulator.

is required to make the link travel at a constant radial velocity.

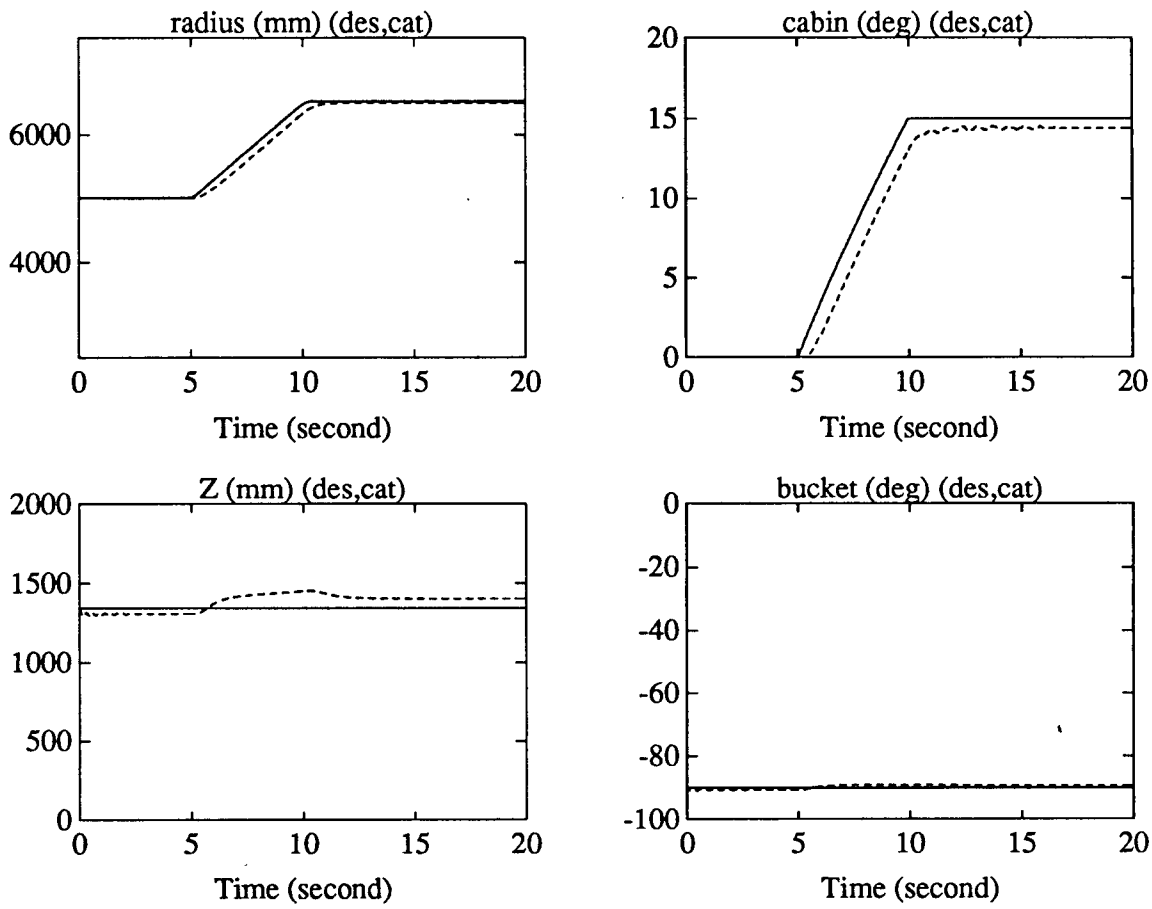

Figure 5.9: RESOLVED mode radius, cabin rotation, height, and bucket rotation responses in the simulator.

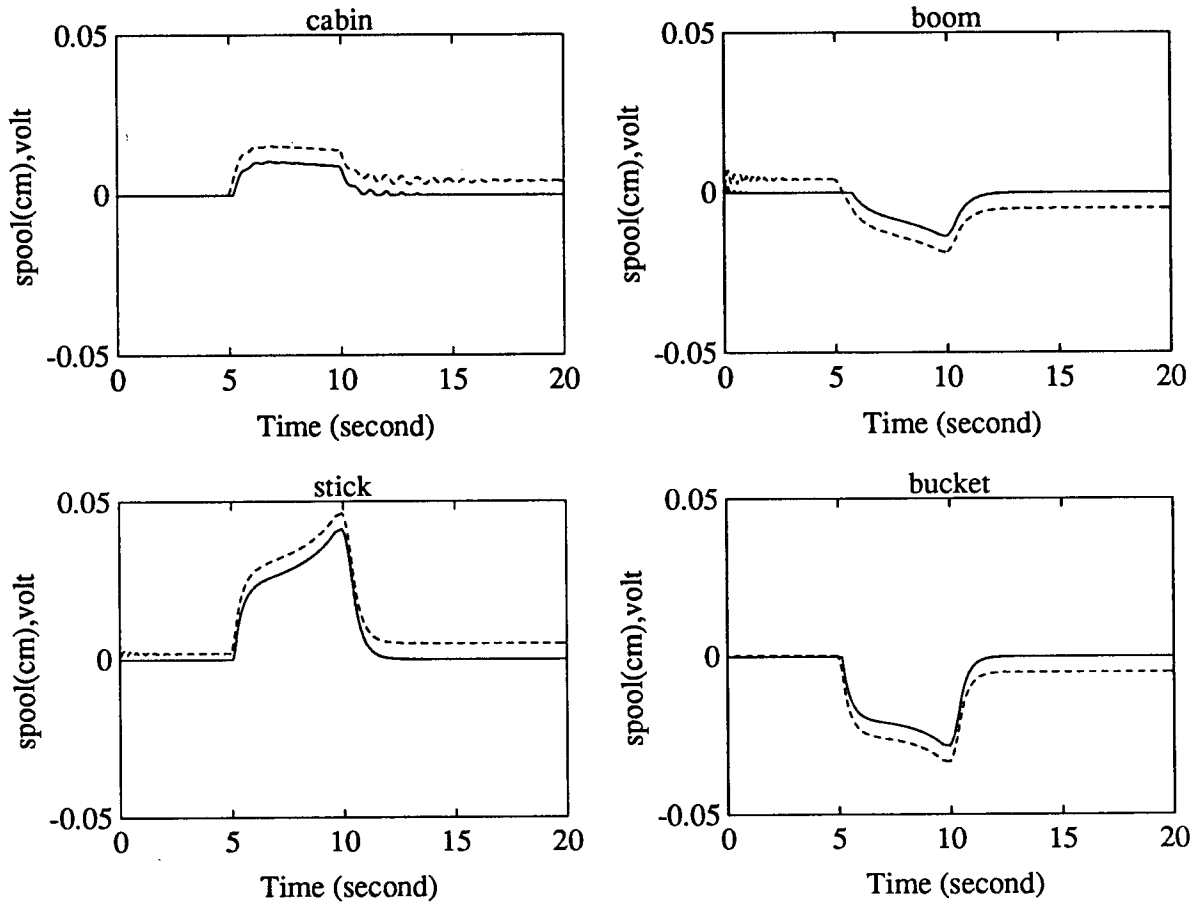

Figure 5.10: RESOLVED mode v and x response in the simulator.

5.6 Summary

In this chapter we have described an implementation of the excavator simulator. The complete computer architecture with joysticks, graphics, and simple actuators has been outlined. The formulation and computer architecture suggested in Chapters 3 and 4 were used to produce a system that met the real-time objective.

Theoretical parallel time cost versus the actual parallel implementation cost showed the implementation was 6.75 times slower than expected. It is speculated that this inadequacy is due to array indexing and subroutine calls.

A theoretical speedup of 4.63 was estimated for the parallel simulation over the single cpu simulation, while in practice a factor of 5.38 was achieved. There were less subroutine calls and array index calculations in the parallel implementation than in the single cpu implementation, and it is assumed that this produced the better than expected parallel/single cpu speedup. This was in spite of the extra costs incurred by the parallel implementation due to link communications.

Finally, JOINT mode and RESOLVED mode simulations were carried out to demonstrate the usefulness of the simulator and to help design the joystick controller. The results have been helpful in tuning the controls, and have been implemented on the excavator. Subjective observations have indicated that the behaviour of the simulator is close to the excavator, even though the true hydraulic system was not simulated. Future experiments, however, will require a more accurate model of the hydraulics.

Chapter 6

Conclusions and Further Work

6.1 Summary of the Thesis

A multibody dynamics formulation has been developed for the purposes of real-time simulation of large scale robotic mechanisms such as excavators. The formulation models the rigid body dynamics of any arbitrary tree structured mechanism (excluding the actuator dynamics, which have been examined in other theses [Sepehri 90]). It is limited at present to single degree of freedom rotational joints, although later in this chapter, the extension to multi-degree of freedom rotational and prismatic joints is described. Closed chains are also not included in the present formulation, but are discussed later in this chapter.

The Newton Euler State Space method developed by Hemami [1982] is used as the basis for the formulation. This method is an example of the orthogonal complement approach, which describes the dynamics by projecting an initial description of the primitive equations of motion (the derivatives of translational and angular momentum, and the kinematic equations) from angular and translational Cartesian coordinates $\dot{w}$ and $\ddot{x}$ to coordinates such as relative Euler angle accelerations $\ddot{\theta}$, using a Jacobian velocity transform. The primitive equations are described using Newton's and Euler's equations for momentum derivatives. The internal constraint forces and torques in the hinges are also eliminated by this projection. Other examples of the orthogonal complement approach use the Lagrangian and kinetic energy to justify the projection [Angeles 1988], [Kim 1986].

Hemami [Hemami 82] initially developed the method using mixed coordinates. The translational forces were referred to the inertial frame and the rotational equations were referred to the body coordinate frame. By referring all equations to body coordinates, an efficient $O(n^3)$

algorithm was developed in this thesis for inertia matrix formation. Buchner [Buchner 86] also referred his translational force equations to body coordinates, but the equations were not developed to the recursive vector cross product form ($\rho^i_{i,j} \times c^i_{i,j}$). As a consequence, the equations were computationally less efficient. The recursive nature of the kinematic and momentum derivative equations has been used in this thesis to derive a single cpu algorithm which is similar to Angeles' algorithm when defined for a single chain. The algorithm is relatively efficient, although other algorithms [Lilly 91] are better for single cpu implementation. This technique of referring each equation to body coordinates was developed independently from Angeles and Ma [1988], who used the body-referred Jacobian transform equations for the dynamics of a single chain. Their work was developed using the Lagrangian approach and Kane's equations to identify the equivalent projection matrix. The work in this thesis has been developed using Newton-Euler principles, as proposed by Hemami and Buchner, but it continues those developments until the very efficient recursive cross product form is established. In addition, the single cpu algorithm models tree structured mechanisms by adding a connectivity matrix to describe the topology of the mechanism.

A new $O(n^2)$ single cpu algorithm for the force vector has been developed which has its origins in the $O(n)$ algorithm developed by Angeles, Ma and Rojas [1989]. The algorithm makes use of matrix calculations performed by the inertia matrix algorithm. The algorithm defines the force vector calculation as the multiplication of the $\bar{H}^T$ projection matrix with the momentum derivative vector. As a consequence of using the $\bar{H}$ matrix (already calculated in the inertia matrix algorithm), this force vector algorithm is only useful when the forward dynamics are calculated. Simulations of the PUMA 600 manipulator and the human torso (with a head and two arms) using open loop inverse dynamics control appeared in Chapter 3 as examples of the veracity of the formulation.

The need for real-time simulation has led to the development of a parallel formulation (Chapter 4) derived from the single cpu algorithms from Chapter 3. Multiprocessor algorithms were developed for inertia matrix formation, force vector formation, matrix solution, and a fourth order fixed step size Runge Kutta integration routine. These algorithms are of $O(n)$ complexity,

and together they form a parallel formulation which is more efficient (Figs. 4.8 and 4.9) than the parallel formulations with which we have compared (given the implementation assumptions stated in section 4.7) The spatially recursive equations for calculating the vectors and forces required in the dynamics naturally suggest a pipeline approach for the computational architecture. This is most easily seen in the computations of row 0 and row 1 of the computer architecture.

The inertia matrix algorithm uses a 2d $\frac{n(n+1)}{2}$ triangular nearest neighbour architecture, as all data communication in the algorithm is between nearest neighbours. The force vector algorithm uses an n cpu pipeline, and shares data with the inertia matrix cpu array. The matrix solver is a feedforward systolic algorithm [Jainandunsing 89] that calculates the solution in $4n$ cycles, compared to $7n - 5$ cycles using Liu and Young's systolic Cholesky factorisation plus backsubstitution method [Liu 83]. The feedforward solver requires $\frac{n(n+1)}{2} + n$ cpus arranged in a triangular nearest neighbour array. A good fit is thus accomplished between the topologies necessary for equation formation and matrix solution, enabling a smooth integration between the formation and solving phases.

The proposed algorithms have been implemented on a computer architecture consisting of a host cpu which controls a cluster of Inmos T800 transputers interconnected by a crossbar network. Both the formulation and the computer architecture are relatively general in nature, permitting more complex mechanisms to be modeled and other types of robotic tasks to be computed. A Caterpillar 215B excavator was simulated in real-time using fourteen transputers connected in a triangular, 2d nearest neighbour grid network. Another transputer was used to interface between the grid, the host computer, and the control transputer which modelled the spool valves of the actuators. The control transputer connected the interface transputer to the graphical display computer and the joysticks (Fig. 5.2).

Single cpu and multiple cpu implementations were developed for the excavator. Simulations testing joint mode and resolved mode joystick control were successfully performed. Timing measurements of the excavator simulations showed that the parallel implementation was able to perform in real-time and that there was a significant speedup relative to the single cpu

implementation.

6.2 Concluding Remarks

At the beginning of this thesis the objective and scope of the study as well as the measures by which its success could be evaluated were stated. The objective was to develop and implement a real-time simulator for the Caterpillar 215B excavator. The scope extended beyond this objective to include a relatively general formulation which would permit implementations of more complex multibody systems. Initial limitations were to exclude closed chain mechanisms and model only single degree of freedom rotational joints. A parallel computer architecture was deemed the best methodology for achieving real-time speed.

The thesis work can be divided into two tasks – the theoretical distribution of the computations, and the implementation of the theoretical distribution on a computer architecture. Two criteria were chosen to evaluate these tasks – computational complexity, an objective measure that can be used to compare the theoretical computational costs of other formulations; and the quality of the implementation, a subjective analysis of the computational architecture examining its expandability, homogeneity, efficiency, and how closely the computer architecture matches the ideal computational architecture.

The recursive form of the equations for calculating the momentum derivative and kinematics dictates the theoretical distribution of the computations, naturally suggesting a pipeline/mesh architecture. In the case of $O(n + \lceil \log_2 n \rceil)$ formulations, the referral of the coordinates to the inertial frame destroys this data dependence, enabling these formulations to use the recursive doubling algorithm to get an efficient algorithm. We have shown, however, that the overhead in referring the equations to the inertial frame results in high initial costs, making our formulation more efficient for systems with moderate numbers of degrees of freedom.

The ideal architecture has been directly implemented using the transputer array (whereas [Fijany 89] chose a mesh even though a hypercube was ideal). A direct implementation has

enabled us to make conclusions about the theoretical distributed architecture. Since the implementation architecture is a mesh, it can be easily expanded to model robots with varying numbers of links by adding a new column of cpus for each new degree of freedom. As the implementation architecture's structure is homogeneous, the architecture does not need to change as the formulation moves from the equation formation phase to the solving phase, thus obviating the need for using either reconfigurable architectures or two separate architectures.

The implementation using Inmos T800 transputers has produced some useful information about tightly coupled mutiprocessor arrays. Timing measurements show that real-time performance is achievable for the excavator simulator in spite of the large overhead incurred due to subroutine calls and array index calculations (a factor of 6 slower). Inline code and direct array element addressing can solve this, but would do so with a loss of generality in the software. The communication costs can be quite significant and it is desirable to minimise the delays by overlapping the communications with computations (although discussed, in this first implementation the overlap technique was not used).

6.3 Contributions to the Literature

The thesis has contributed to the literature in both of the areas described in the last section. On the topic of equation development and the theoretical distribution of computations, a contribution has been made:

- in the development of an efficient recursive set of equations for generating the cross product form of the projection matrix using body coordinates, from Newton-Euler rather than Lagrangian principles.
- in deriving an efficient single cpu $O(n^3)$ formulation of the inertia matrix for both single chain and tree structured mechanisms.
- in deriving an efficient single cpu $O(n^2)$ formulation of the force vector for both single chain and tree structured mechanisms, which must be used in the context of a forward dynamics algorithm as it uses results from the formation of the inertia matrix.

- in deriving an $O(n)$ mesh structured multiprocessor formulation of the inertia matrix for single chain mechanisms.
- in deriving an $O(n)$ pipeline multiprocessor formulation of the force vector for single chain mechanisms. The inertia matrix, force vector and solver algorithms together are more efficient than other algorithms for up to fifteen degrees of freedom (the relative efficiencies will depend on the details of the various implementations).

The real-time implementation of the simulator has led to the following innovative aspects:

- the development of a flexible computer architecture that is able to compute the dynamics as well as other tasks.
- the development of a computational architecture which succesfully integrates the equation formation and solving phases.
- the development of a complete real-time simulation facility for the UBC Teleoperation Laboratory.
- an implementation of Jainandunsing's feedforward systolic solver.

6.4 Topics Requiring Further Exploration

In this thesis we have developed a new parallel formulation for rigid body dynamics. Although the model is limited to rigid bodies with a single degree of rotational freedom, it is a foundation upon which more complex behaviour can be added. The following sections consider some areas of further research that would be logical extensions: multiple degree of freedom joints, a parallel architecture for branched mechanisms, closed chain models, and actuators and friction.

6.4.1 Multiple degree of freedom joints

Joints with more than one degree of freedom per joint can be included by modifying the Jacobian velocity transformation vector $\tilde{H}_{i,j} = [c^i_{i,j} \;\; \rho^i_{i,j} \times c^i_{i,j}]$, which is a 6×1 vector relating the Cartesian

jth joint type	$\bar{H}_{i,j}$	size
Floating base body	$\begin{bmatrix} 0 & A^i_j \\ A^i_j & \tilde{\rho}^i_{i,j} \end{bmatrix}$	6×6
Revolute joint	$\begin{bmatrix} c^i_{i,j/z} \\ \rho^i_{i,j} \times c^i_{i,j/z} \end{bmatrix}$	6×1
Translational joint	$\begin{bmatrix} 0 \\ c^i_{i,j/z} \end{bmatrix}$	6×1
Cylindrical joint	$\begin{bmatrix} c^i_{i,j/z} & 0 \\ \rho^i_{i,j} \times c^i_{i,j/z} & c^i_{i,j/y} \end{bmatrix}$	6×2
Universal joint	$\begin{bmatrix} c^i_{i,j/z} & c^i_{i,j/y} \\ \rho^i_{i,j} \times c^i_{i,j/z} & \rho^i_{i,j} \times c^i_{i,j/y} \end{bmatrix}$	6×2
Spherical joint	$\begin{bmatrix} c^i_{i,j/z} & c^i_{i,j/y} & c^i_{i,j/x} \\ \rho^i_{i,j} \times c^i_{i,j/z} & \rho^i_{i,j} \times c^i_{i,j/y} & \rho^i_{i,j} \times c^i_{i,j/x} \end{bmatrix}$	6×3

Table 6.1: $\bar{H}_{i,j}$ for various types of joints

velocity vector $[w_i \quad \dot{x}_i]$ to the rotational coordinate $\dot{q}_i$. The $\bar{H}_{i,j}$ used throughout the thesis represents the Jacobian vector for a $\dot{q}_i$ which is a single degree of freedom rotational joint. Kim and Vanderploeg [Kim 86a] have derived the $\bar{H}_{i,j}$ for other types of joints. Table 6.1 contains a modified form of Kim's matrices in which the $\bar{H}_{i,j}$ have been referred to the ith coordinate frame, so that the terms are applicable to the algorithms presented in this thesis. Note that we have defined $c^i_{i,j/z} = A^i_j z^j_j$, $c^i_{i,j/y} = A^i_j y^j_j$ and $c^i_{i,j/x} = A^i_j x^j_j$, where z, y, and x are unit axes in the jth coordinate frame and A^i_j is a function of the appropriate $\theta_{x,y,\ or\ z}$.

In the computation of $\mathcal{M}$ for single degree of freedom joints, the $\bar{H}_{i,j}$ vectors are elements of the $[\rho \times C]$ and $[C]$ matrices. For more complex joints, the development of the C and $\rho \times C$ matrices proceed with the matrices from Table 6.1 as the new $\bar{H}_{i,j}$. This can be shown by developing new kinematic equations for the angular velocity, originally described in Chapter 1. These equations must be modified to include more than one degree of freedom in the joint. As an example, consider a body i in which the ith joint has rotational degrees of freedom in y and z axes. The angular velocity kinematics are:

$$w^i_{i,0} = A^i_{i-1} w^{i-1}_{i-1,0} + z^i_i \theta_{i/z} + y^i_i \theta_{i/y} \tag{6.1}$$

$$\dot{w}^i_{i,0} = A^i_{i-1} \dot{w}^{i-1}_{i-1} + w^i_{i,0} \times z^i_i \theta_{i/z} + w^i_{i,0} \times y^i_i \theta_{i/y} + z^i_i \ddot{\theta}_{i/z} + y^i_i \ddot{\theta}_{i/y} \tag{6.2}$$

The A^i_{i-1} matrices must also be redefined to include the sequence of $\theta_{i/y}$ and $\theta_{i/z}$ rotations. Once the w and $\dot{w}$ vectors have been calculated, the $\dot{L}$, $m\ddot{p}$ and W vectors or matrices, used for calculating the force vector, are calculated in the same manner as in Chapter 3.

Consider a single chain with n links and n joints, some of which have multiple degrees of freedom. The total number of degrees of freedom in the system is m ($n < m$) due to the extra degrees of freedom. Including these types of joints in the parallel formulation requires a re-examination of the distribution of the computations. One issue that must be considered is the timing of the pipelined sequence of operations. It may be necessary to add a number of cpus for each extra degree of freedom in the system if the time delay for any operation in the pipeline is to remain comparable to the other operations occurring simultaneously. A first examination of the problem indicates that for each extra degree of freedom, a row of $n - i$ cpus would be added next to the ith row for the computation of H, and a column of i cpus would be added next to the ith column for the $\bar{H}^T H$ computation. To compute the inertia matrix, it may be necessary to start with an array of $\frac{m(m+1)}{2}$ cpus, some of which will be idle during different phases of the computation.

6.4.2 A parallel architecture for the branched formulation

To simulate branched topologies with the multiprocessor mesh, it is necessary to examine the single cpu algorithm developed in Chapter 3. From equation 3.14,

$$C = \begin{bmatrix} z & 0 & 0 & 0 & 0 \\ A^2_1 z & z & 0 & 0 & 0 \\ A^3_2 A^2_1 z & A^3_2 z & z & 0 & 0 \\ A^4_1 z & 0 & 0 & z & 0 \\ A^5_4 A^4_1 z & 0 & 0 & A^5_4 z & z \end{bmatrix}, \tag{6.3}$$

where the C matrix (as well as the $\rho \times C$ matrix) shows the same structure as the connectivity matrix U^{-1}. One can imagine each row of the C matrix being produced by one of the row 1 cpus in the inertia matrix array. The pipeline in row 1 is broken up, however, since $\text{cpu}_{1,4}$, calculating $A^4_1 z$, does not make use of any terms produced by the cpu in front of it in the

pipeline (the third row in the C matrix). It receives its input vector from $\text{cpu}_{1,1}$ since body 4 is connected to body 1 in the topological description. The pipeline must be altered to provide an extra link from $\text{cpu}_{1,1}$ to $\text{cpu}_{1,4}$. The pipeline of cpus in the torque calculator (row 0 of the mesh) must also be altered to provide this connection. As a general rule, it is necessary for rows 0 and 1 to conform to the topology of the mechanism.

Another consideration is the requirement for the $\bar{H}^T H$ calculation and the systolic solver to use a full triangular nearest neighbour architecture. Connections must be provided between $\text{cpu}_{i,3}$ and $\text{cpu}_{i,4}$ in order that the terms may be accumulated in the $\bar{H}^T H$ computation, and so that the systolic matric solver can perform its embedded rotations. Thus even the cpus which would contain 0 in the C matrix computation need to be present.

It is possible to use the existing network topology to pass the data in rows 0 and 1, but this would require some cpus to transmit data destined for other cpus which are not relevant to their own operations, and this would cause difficulties in the timing of operations. A better solution would be to provide an extra common bus between rows 0 and 1 so that data that is not destined for an adjacent cpu can be routed via this bus to its destination. This would only be necessary during the $[C \;\; \rho \times C]$ computation phase in row 1 cpus, and the $[\dot{L} \;\; m\ddot{p}]$ computation for row 0 cpus. After this, the nearest neighbour connections would be used for the subsequent phases. At present this connectivity is not provided in the existing computational architecture, but it should not be difficult to implement, as no extra communication links are required from the transputer. The communication links across rows 0 and 1 need to be connected to a common bus such as the VME bus itself, or perhaps the secondary serial S-bus. Fig. 6.1 illustrates the architecture for a four body branched system. Note that the bus provides the path for cpus b_1 and a_{11} to connect to b_3 and a_{13}.

6.4.3 Closed chains

The simulation of closed chain mechanisms can involve both analytical and numerical methods. The primary problem is the selection or identification of the set of independent coordinates

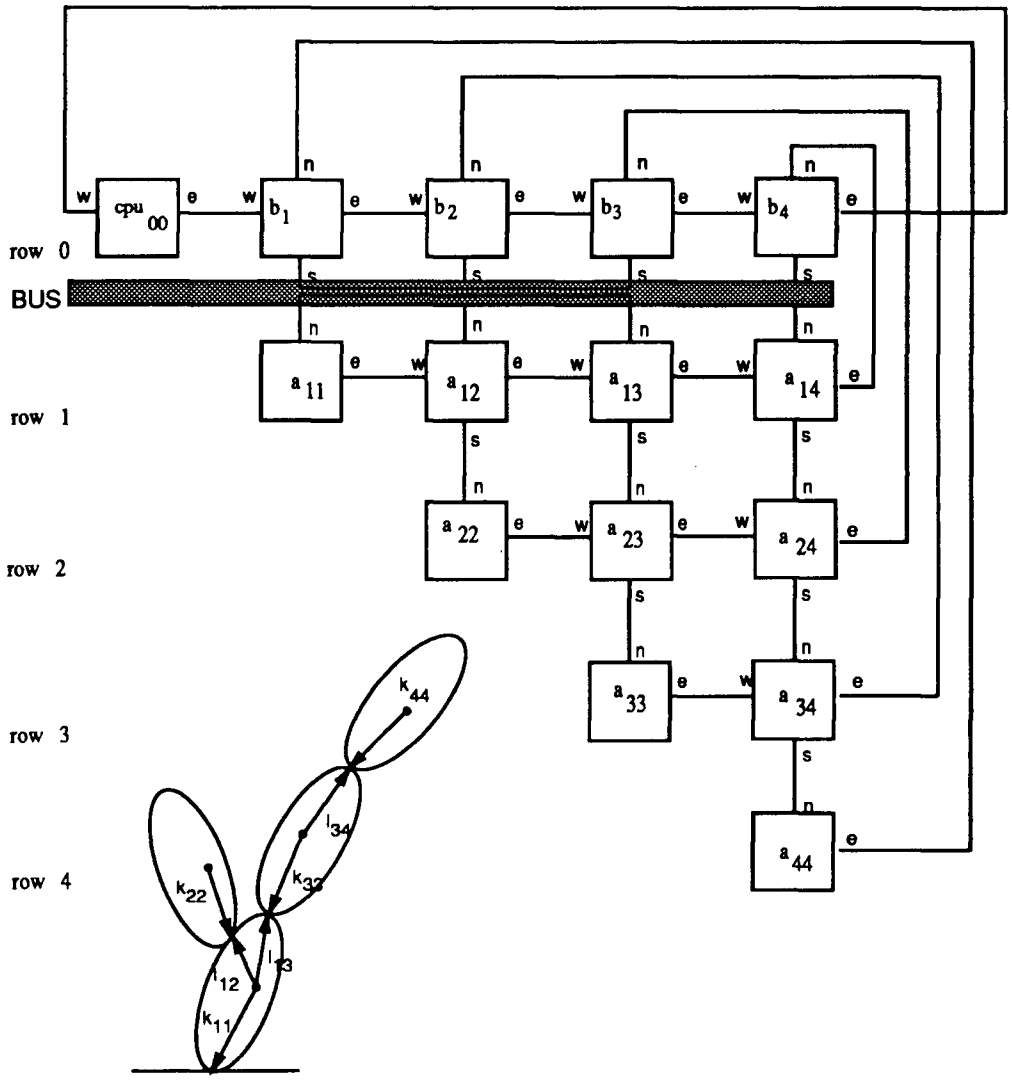

Figure 6.1: Architecture for branched four body system

that are used to describe the system. This problem is exacerbated when joints lock together or separate, causing a time varying topology in which the variables belonging to the set of independent coordinates change membership with time. Initially the loops in the closed chain system are cut to form a tree topology. The independent coordinates are then selected, and the constraints forces which close the loops are added to make the motion of the tree system consistent with the closed chain system.

Murray and Lovell [1989], Ibrahim [Ibrahim 1988] and others have approached the problem by using a tree structure as the initial representation of the closed chain mechanism (loops are cut to form a tree). The inertia matrix and force vector for the tree are developed, using a set of independent coordinates representing the tree. A reduction of this set of coordinates is then made, so that the remaining subset of coordinates can represent the closed loop structure (there are less degrees of freedom in a closed loop system). The reduced set of coordinates may often be chosen by a logical examination of the topology, followed by the formation of the constraint equations used to close the loops in the tree structure (in order to be equivalent to the original closed chain system). In cases when the appropriate coordinates are not obvious, they may be found numerically (e.g. QR decomposition of a matrix representing the system constraints). Once the reduced set of coordinates is identified, a projection matrix is formed which projects from the set representing the tree to a subset representing the closed chain.

The following equations describe the approach Murray and Gilbert used to formulate the closed chain dynamics of an arbitrary mechanism. Following this, we discuss how this method might be incorporated into the algorithms of Chapter 3.

Consider a closed chain mechanism with m rigid bodies (one degree of freedom (dof) per joint) with n true dof (some of the dof are constrained by the closed loops, and are immobile, so $n < m$). The n one dof joints are driven by n actuator torques τ, producing an $n \times 1$ vector of joint displacements q. An open chain equivalent system (ignoring the constraint forces for now) has m dof and $m \times 1$ joint displacements q_r, one for each body. The equations of motion for this open chain system are (relating them to Chapters 2 and 3):

$$\tau_r = \mathcal{M}_r(q_r)\ddot{q}_r + H_r(q_r, \dot{q}_r) \tag{6.4}$$

where $\mathcal{M}_r$ is the tree mechanism system inertia matrix and H_r is the force vector of Coriolis, centrifugal and gravity forces. To equate the open chain model to the closed chain model, the virtual work done must be equal, and so

$$\delta q^T \tau = \delta q_r^T \tau_r \tag{6.5}$$

According to D'Alembert's principle, no net work can be done by the applied and actuating

forces for any virtual displacement consistent with the constraints, and so as the set of q is changed to q_r, τ changes to τ_r in order that the system virtual work remains the same. Murray shows that q_r is a function of q, and more specifically, that the Jacobian from this relationship can be defined as

$$q_r = f(q) \tag{6.6}$$

$$\dot{q}_r = G\dot{q} \tag{6.7}$$

Equation 6.7 represents the transformation from the open loop coordinate set to the closed loop subset. G is the $m \times n$ Jacobian constraint matrix describing the forces necessary to close the tree structured mechanism so that it is equivalent to the closed chain mechanism.

Substituting $G\dot{q}$ for q_r in the equation of virtual work, it is possible to show that

$$\tau = G^T \tau_r \tag{6.8}$$

The equations of motion can be rearranged to yield

$$\tau = G^T[\mathcal{M}_r(q_r)\ddot{q}_r + H_r(q_r, \dot{q}_r)] \tag{6.9}$$

The time derivative of $\dot{q}_r$ is

$$\ddot{q}_r = G\ddot{q} + \dot{G}\dot{q} \tag{6.10}$$

The equation of motion can then be rewritten as

$$\tau = G^T[\mathcal{M}_r(f(q))(G\ddot{q} + \dot{G}\dot{q}) + H_r(f(q), G\dot{q})] \tag{6.11}$$

$$= \mathcal{M}_c\ddot{q} + H_c \tag{6.12}$$

where

$$\mathcal{M}_c = G^T\mathcal{M}_r(f(q))G \tag{6.13}$$

$$H_c = G^T\mathcal{M}_r(f(q))\dot{G}\dot{q} + G^T H_r(f(q), G\dot{q}) \tag{6.14}$$

In the above development, we assume collocated links (all measured joint coordinates are also actuated), whereas Murray and Gilbert also consider non-collocated manipulators. Non-collocated

manipulators would project the system to yet another set of coordinates, say q_m, which would produce a non-symmetric inertia matrix, as equations 6.6 and 6.9 would be functions of q_m and G_m.

If it is possible to measure all the q_r and $\dot{q}_r$ from the tree structured system, then $\mathcal{M}_r(q_r)$ is obtained from the equations of Chapter 3, and so $\mathcal{M}_c$ can be computed as

$$\mathcal{M}_c = G^T \bar{H}^T M \bar{H} G \tag{6.15}$$

where $\bar{H}$ is from Chapter 3 and M is the square matrix of Js and mIs. Once G is computed, $P = \bar{H}G$ replaces $\bar{H}$ as the pre and post multiplier matrix transformation in the inertia matrix algorithm. The computation of H_c is more complicated due to the presence of the $\dot{G}\dot{q}$ term in 6.10. Gilbert proposes to calculate H_c from a closed chain inverse dynamics algorithm by setting $\ddot{q}$ to zero in a similar way to the open loop inverse dynamics algorithm described by Walker and Orin [Walker 1982].

The incorporation of the above procedure into the algorithms developed in Chapters 2, 3, and 4 requires a number of issues to be addressed. These include the computation of H_c, the computation of G (which requires the identification of q), and the parallel computation of P and its effect on the computer architecture. Since the final size of the inertia matrix is $n \times n$, the number of cpus used in the calculation of the open loop $\mathcal{M}_r$ ($\frac{m(m+1)}{2}$) is more than that required by the systolic solver phase ($\frac{n(n+1)}{2}$ cpus). It is thus necessary to find the best way to compute P so that, if possible, only $\frac{n(n+1)}{2}$ cpus are needed for all phases of the computation.

6.4.4 Actuators and friction effects

Currently, the hydraulic actuators are modeled as simple rotary actuators which are not coupled to each other. As a consequence, actuator i's equations are calculated in $\text{cpu}_{i,0}$. Once a distributed version of the true actuation equations is developed, these can be calculated in row 0 cpus, or another row of cpus can be added on top of row 0. Friction terms, may be a function of the velocity $\dot{\theta}_i$ (e.g. viscous friction) or a function of reaction forces f_v and inertial terms. Viscous friction may be independently computed because each term is a function of only local

information (e.g. the $\dot{\theta}_i$ variables are produced by row 0 cpus, and so they will be readily available for the friction computations). The resulting friction vector can simply be added to the b vector. Dry friction is more complex and involves the redistribution of variables produced by other cpus in the pipeline.

Appendix A

Single Chain Dynamics

The matrices describing the equations of motion are (for a five body fixed chain):

$$N_1 = \begin{bmatrix} R^1_{1,1} & -A^1_2 R^2_{2,2} & 0 & 0 & 0 \\ 0 & R^2_{2,2} & -A^2_3 R^3_{3,3} & 0 & 0 \\ 0 & 0 & R^3_{3,3} & -A^3_4 R^4_{4,4} & 0 \\ 0 & 0 & 0 & R^4_{4,4} & -A^4_5 R^5_{5,5} \\ 0 & 0 & 0 & 0 & R^5_{5,5} \end{bmatrix}; J = \begin{bmatrix} J^1_1 & & & & \\ & J^2_2 & & & \\ & & J^3_3 & & \\ & & & J^4_4 & \\ & & & & J^5_5 \end{bmatrix};$$

$$U = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ -I & I & 0 & 0 & 0 \\ 0 & -I & I & 0 & 0 \\ 0 & 0 & -I & I & 0 \\ 0 & 0 & 0 & -I & I \end{bmatrix}; M = \begin{bmatrix} m_1 I & & & & \\ & m_2 I & & & \\ & & m_3 I & & \\ & & & m_4 I & \\ & & & & m_5 I \end{bmatrix};$$

$$\dot{Q}w = \begin{bmatrix} +A^0_1(\tilde{w}^1_{1,0})^2 k^1_{1,1} \\ -A^0_1(\tilde{w}^1_{1,0})^2 l^1_{1,2} + A^0_2(\tilde{w}^2_{2,0})^2 k^2_{2,2} \\ -A^0_2(\tilde{w}^2_{2,0})^2 l^2_{2,3} + A^0_3(\tilde{w}^3_{3,0})^2 k^3_{3,3} \\ -A^0_3(\tilde{w}^3_{3,0})^2 l^3_{3,4} + A^0_4(\tilde{w}^4_{4,0})^2 k^4_{4,4} \\ -A^0_4(\tilde{w}^4_{4,0})^2 l^4_{4,5} + A^0_5(\tilde{w}^5_{5,0})^2 k^5_{5,5} \end{bmatrix}; f = \begin{bmatrix} w^1_{1,0} \times J^1_1 w^1_{1,0} \\ w^2_{2,0} \times J^2_2 w^2_{2,0} \\ w^3_{3,0} \times J^3_3 w^3_{3,0} \\ w^4_{4,0} \times J^4_4 w^4_{4,0} \\ w^5_{5,0} \times J^5_5 w^5_{5,0} \end{bmatrix}; g = \begin{bmatrix} m_1 \mathbf{g} \\ m_2 \mathbf{g} \\ m_3 \mathbf{g} \\ m_4 \mathbf{g} \\ m_5 \mathbf{g} \end{bmatrix};$$

$$Q = \begin{bmatrix} A^0_1 \tilde{k}^1_{1,1} & 0 & 0 & 0 & 0 \\ -A^0_1 \tilde{l}^1_{1,2} & A^0_2 \tilde{k}^2_{2,2} & 0 & 0 & 0 \\ 0 & -A^0_2 \tilde{l}^2_{2,3} & A^0_3 \tilde{k}^3_{3,3} & 0 & 0 \\ 0 & 0 & -A^0_3 \tilde{l}^3_{3,4} & A^0_4 \tilde{k}^4_{4,4} & 0 \\ 0 & 0 & 0 & -A^0_4 \tilde{l}^5_{4,5} & A^0_5 \tilde{k}^5_{5,5} \end{bmatrix};$$

$$U^{-1} = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ I & I & 0 & 0 & 0 \\ I & I & I & 0 & 0 \\ I & I & I & I & 0 \\ I & I & I & I & I \end{bmatrix}; E = \begin{bmatrix} I & -A_2^1 & 0 & 0 & 0 \\ 0 & I & -A_3^2 & 0 & 0 \\ 0 & 0 & I & -A_4^3 & 0 \\ 0 & 0 & 0 & I & -A_5^4 \\ 0 & 0 & 0 & 0 & I \end{bmatrix};$$

$$C = \begin{bmatrix} z & 0 & 0 & 0 & 0 \\ A_1^2 z & z & 0 & 0 & 0 \\ A_1^3 z & A_2^3 z & z & 0 & 0 \\ A_1^4 z & A_2^4 z & A_3^4 z & z & 0 \\ A_1^5 z & A_2^5 z & A_3^5 z & A_4^5 z & z \end{bmatrix}; H_2 = \begin{bmatrix} I & -I & 0 & 0 & 0 \\ 0 & I & -I & 0 & 0 \\ 0 & 0 & I & -I & 0 \\ 0 & 0 & 0 & I & -I \\ 0 & 0 & 0 & 0 & I \end{bmatrix}; \; H_1 = -Q^T.$$

$$U^{-1}Q = \begin{bmatrix} A_1^0 \tilde{k}_{1,1}^1 & 0 & 0 & 0 & 0 \\ A_1^0 \tilde{d}_{2,1}^1 & A_2^0 \tilde{k}_{2,2}^2 & 0 & 0 & 0 \\ A_1^0 \tilde{d}_{2,1}^1 & A_2^0 \tilde{d}_{3,2}^2 & A_3^0 \tilde{k}_{3,3}^3 & 0 & 0 \\ A_1^0 \tilde{d}_{2,1}^1 & A_2^0 \tilde{d}_{3,2}^2 & A_3^0 \tilde{d}_{4,3}^3 & A_4^0 \tilde{k}_{4,4}^4 & 0 \\ A_1^0 \tilde{d}_{2,1}^1 & A_2^0 \tilde{d}_{3,2}^2 & A_3^0 \tilde{d}_{4,3}^3 & A_4^0 \tilde{d}_{5,4}^4 & A_5^0 \tilde{k}_{5,5}^5 \end{bmatrix};$$

An algorithm for calculating $[(U^{-1}\dot{Q}w)_{bcs}]_i$ is

$$[(U^{-1}\dot{Q}w)_{bcs}]_i = [(U^{-1}\dot{Q}w)_{bcs}]_{i-1} + (\tilde{w}_{i-1,0}^{i-1})^2 l_{i-1,i}^{i-1} - (\tilde{w}_{i,0}^{i})^2 k_{i,i}^{i}; \qquad i = 1, \ldots n \quad \text{(A.1)}$$

$$[U^{-1}Q]_{bcs} = \begin{bmatrix} \tilde{k}_{1,1}^1 & 0 & 0 & 0 & 0 \\ A_1^2 \tilde{d}_{2,1}^1 & \tilde{k}_{2,2}^2 & 0 & 0 & 0 \\ A_1^3 \tilde{d}_{2,1}^1 & A_2^3 \tilde{d}_{3,2}^2 & \tilde{k}_{3,3}^3 & 0 & 0 \\ A_1^4 \tilde{d}_{2,1}^1 & A_2^4 \tilde{d}_{3,2}^2 & A_3^4 \tilde{d}_{4,3}^3 & \tilde{k}_{4,4}^4 & 0 \\ A_1^5 \tilde{d}_{2,1}^1 & A_2^5 \tilde{d}_{3,2}^2 & A_3^5 \tilde{d}_{4,3}^3 & A_4^5 \tilde{d}_{5,4}^4 & \tilde{k}_{5,5}^5 \end{bmatrix};$$

Appendix B

Branched Chain Dynamics

The following matrices are the coefficients for the multibody system represented in Fig. 3.1.

$$N_1 = \begin{bmatrix} R^1_{1,1} & -A^1_2 R^2_{2,2} & 0 & -A^1_4 R^4_{4,4} & 0 \\ 0 & R^2_{2,2} & -A^2_3 R^3_{3,3} & 0 & 0 \\ 0 & 0 & R^3_{3,3} & 0 & 0 \\ 0 & 0 & 0 & R^4_{4,4} & -A^4_5 R^5_{5,5} \\ 0 & 0 & 0 & 0 & R^5_{5,5} \end{bmatrix};$$

$$E = \begin{bmatrix} I & -A^1_2 & 0 & -A^1_4 & 0 \\ 0 & I & -A^2_3 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & -A^4_5 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}; f = \begin{bmatrix} w^1_{1,0} \times J^1_1 w^1_{1,0} \\ w^2_{2,0} \times J^2_2 w^2_{2,0} \\ w^3_{3,0} \times J^3_3 w^3_{3,0} \\ w^4_{4,0} \times J^4_4 w^4_{4,0} \\ w^5_{5,0} \times J^5_5 w^5_{5,0} \end{bmatrix}; g = \begin{bmatrix} m_1 \mathbf{g} \\ m_2 \mathbf{g} \\ m_3 \mathbf{g} \\ m_4 \mathbf{g} \\ m_5 \mathbf{g} \end{bmatrix};$$

$$U = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ -I & I & 0 & 0 & 0 \\ 0 & -I & I & 0 & 0 \\ -I & 0 & 0 & I & 0 \\ 0 & 0 & 0 & -I & I \end{bmatrix}; \quad \dot{Q}w = \begin{bmatrix} +A^0_1(\tilde{w}^1_{1,0})^2 k^1_{1,1} \\ -A^0_1(\tilde{w}^1_{1,0})^2 l^1_{1,2} + A^0_2(\tilde{w}^2_{2,0})^2 k^2_{2,2} \\ -A^0_2(\tilde{w}^2_{2,0})^2 l^2_{2,3} + A^0_3(\tilde{w}^3_{3,0})^2 k^3_{3,3} \\ -A^0_1(\tilde{w}^1_{1,0})^2 l^1_{1,4} + A^0_4(\tilde{w}^4_{4,0})^2 k^4_{4,4} \\ -A^0_4(\tilde{w}^4_{4,0})^2 l^4_{4,5} + A^0_5(\tilde{w}^5_{5,0})^2 k^5_{5,5} \end{bmatrix};$$

$$Q = \begin{bmatrix} A^0_1 \tilde{k}^1_{1,1} & 0 & 0 & 0 & 0 \\ -A^0_1 \tilde{l}^1_{1,2} & A^0_2 \tilde{k}^2_{2,2} & 0 & 0 & 0 \\ 0 & -A^0_2 \tilde{l}^2_{2,3} & A^0_3 \tilde{k}^3_{3,3} & 0 & 0 \\ -A^0_1 \tilde{1,4}^1 & 0 & 0 & A^0_4 \tilde{k}^4_{4,4} & 0 \\ 0 & 0 & 0 & -A^0_4 \tilde{l}^5_{4,5} & A^0_5 \tilde{k}^5_{5,5} \end{bmatrix};$$

$$U^{-1} = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ I & I & 0 & 0 & 0 \\ I & I & I & 0 & 0 \\ I & 0 & 0 & I & 0 \\ I & 0 & 0 & I & I \end{bmatrix}; \quad H_2 = U^T, \quad H_1 = -Q^T.$$

Bibliography

[Amin-Javaheri 88] M. Amin-Javaheri and D.E. Orin, *Systolic Architectures for the Manipulator Inertia Matrix.* IEEE Trans. Systems, Man, and Cybernetics, Vol. 18, No. 6, November/December 1988, pp 939-951.

[Angeles 88] J. Angeles and O. Ma, *Dynamic Simulation of n-Axis Serial Robotic Manipulators using a Natural Orthogonal Complement.* Int. J. Robotics Res., Vol. 7, No. 5, October 1988, pp 32-47.

[Angeles 89] J. Angeles, O. Ma, and A. Rojas, *An Algorithm for the Inverse Dynamics of n-Axis General Manipulators Using Kane's Equations.* Computers Math. Applic. Vol. 17, No. 12, 1989, pp 1545-1561.

[Armstrong 79] W.W. Armstrong, *Recursive Solution to the Equations of Motion of an n-Link Manipulator.* Proc. 5th World Congress on the Theory of Machines and Mechanisms, Vol. 2, 1979, pp 1343-1346.

[Bae 87a] D.S. Bae and E.J. Haug, *A Recursive Formulation for Constrained Mechanical System Dynamics Part I: Open Loop Systems.* Mechanics of Structures and Machines, vol. 15, 1987, pp 359-382.

[Bae 87b] D.S. Bae and E.J. Haug, *A Recursive Formulation for Constrained Mechanical System Dynamics Part II: Closed Loop Systems.* Mechanics of Structures and Machines, vol. 15, 1987, pp 481-506.

[Bae 88a] D.S. Bae, R.S. Hwang, and E.J. Haug, *A Recursive Formulation for Real-Time Dynamic Simulation.* Advances in Design Automation, DE-Vol. 14, ASME Design and Automation Conference, September 1988.

[Bae 88b] D.S. Bae, J.G. Kuhl, and E.J. Haug, *A Recursive Formulation for Constrained Mechanical System Dynamics Part III: Parallel Processor Implementation.* Mechanisms, Structures and Machines, 16 (2), 1988.

[Baumgarte 72] J. Baumgarte, *Stabilisation of Constraints and Integrals of Motion in Dynamic Systems.* Computer Methods in Applied Mechanics and Engineering, 1, 1972, pp 1-16.

[Bayo 88] E. Bayo, J. Garcia de Jalon, and M.A. Serna, *A Modified Lagrangian Formulation for the Dynamic Analysis of Constrained Mechanical Systems.* Computer Methods in Applied Mechanics and Engineering, 71, 1988, pp 183-195.

[Bodley 78] C. Bodley, A. Devers, A. Park, and H. Frisch, *A Digital Computer Program for Dynamic Interaction Simulation of Controls and Structures (DISCOS).* NASA Technical paper 1219, May 1978.

[Brandl 86] H. Brandl, R. Johanni, and M. Otter, *A Very Efficient Algortihm for the Simulation of Robots and Similar Multibody Systems Without Inversion of the Mass Matrix.* IFAC/IFIP/IMACS International Symposium on the Theory of Robots, Vienna, 1986.

[Buchner 86] H.J. Buchner, *Control of Robot Manipulators on Task Oriented Surfaces by Nonlinear Decoupling Feedback and Compensation of Certain Classes of Disturbances.* PhD Thesis, Department of Electrical Engineering, Ohio State University, 1986.

[Faddeev 59] V.N. Faddeev, *Computational Methods of Linear Algebra.* Dover, New York, 1959.

[Featherstone 83] R. Featherstone, *The Calculation of Robot Dynamics using Articulated-Body Inertias.* Int. J. Robotics Res., Vol. 1, No. 1, pp 13-30, Spring 1983.

[Fijany 89] A. Fijany and A. Bejczy, *A Class of Parallel Algorithms for the Computation of the Manipulator Inertia Matrix.* IEEE Trans. Robotics and Automation, Vol. RA-5, No. 5, October 1989, pp 600-615.

[Fuhrer 89] C. Fuhrer, B. Leimkuhler, *Stabilized Differential-Algebraic Formulation of the Equations of Motion of Constrained Mechanical Systems.* NATO Advanced Research Workshop on Real-Time Integration Methods for Mechanical System Simulation, Snowbird, Utah, August 1989.

[Haug 89] E.J. Haug, *Computer-Aided Kinematics and Dynamics of Mechanical Systems Vol. I: Basic Methods.* Allyn and Bacon, Massachusetts, 1989.

[He 89] X. He, and A.A. Goldenberg, *An Algorithm for Efficient Computation of Dynamics of Robotic Manipulators.* Advanced Robotics: 1989, Proceedings of the 4th International Conference on Advanced Robotics, Columbus, Ohio, June 13-15, 1989.

[Hemami 82] H. Hemami, *A State Space Model for Interconnected Rigid Bodies.* IEEE Transactions on Automatic Control, Vol. AC-27, No. 2, April 1982, pp 376-382.

[Ho 74] J. Ho, *The Direct Path Method for Deriving the Dynamic Equations of a Multibody Flexible Spacecraft with a Topological Tree Configuration.* AIAA Mechanics and Control of Flight Conference, paper no. 74-786, California 1974.

[Hooker 66] W.W. Hooker and G. Margulies, *The Dynamical Attitude Equations for an n-Body Satellite.* J. Astronautical Sciences, 12 (1965), pp 123-128.

[Hooker 70] W.W. Hooker, *A Set of r Dynamical Attitude Equations for an Arbitrary n-Body Satellite having r Rotational Degrees of Freedom.* AIAA Journal, Vol. 8, No. 7, July 1970, pp 1205-1207.

[Hooker 73] W.W. Hooker, *Equations of Attitude Motion of a Topological Tree of Bodies, The Terminal Members of which may be Flexible.* Technical Report LMSC-D354938, November 1973, Lockheed Missiles and Space Company, Palo Alto, California.

[Hughes 79] P.C. Hughes, *Dynamics of a Chain of Flexible Bodies.* J. of the Astronautical Sciences, Vol. 27, 1979, pp 359-380.

[Hughes 86] P.C. Hughes, *Spacecraft Attitude Dynamics.* John Wiley and Sons, New York, 1986.

[Hwang 88] R.S. Hwang, D.S. Bae, and E.J. Haug, *Parallel Processing for Real-Time Dynamic System Simulation.* Advances in Design Automation, DE-Vol. 14, ASME Design and Automation Conference, September 1988.

[Ibrahim 88] A.E. Ibrahim, *Mathematical Modelling of Flexible Multibody Dynamics with Application to Orbiting Systems.* PhD Thesis, Department of Mechanical Engineering, University of British Columbia, April 1988.

[Jainandunsing 89] K. Jainandunsing and E.F. Deprettere, *A New Class of Parallel Algorithms for Solving Systems of Linear Equations.* SIAM J. Scientific and Statistical Computing, Vol. 10, No. 5, September 1989, pp 880-912.

[Jerkovsky 78] W. Jerkovsky, *The Structure of Multibody Dynamics Equations.* J. of Guidance and Control, Vol. 1, No. 3, May-June 1978, pp 173-182.

[Keat 83] J. Keat, *Dynamic Equations of Multibody Systems with Application to Space Structure Deployment.* PhD Thesis, MIT, 1983.

[Khosravi 87] B. Khosravi, S. Yurkovich, and H. Hemami, *Control of a Four-Link Biped in a Back Somersault Maneuver.* IEEE Trans. Systems, Man, and Cybernetics, Vol. SMC-17, No. 2, March/April, 1987.

[Kim 86a] S.S. Kim and M.J. Vanderploeg, *A General and Efficient Method for Dynamic Analysis of Mechanical Systems using Velocity Transformations.* ASME Journal of Mechanisms, Transmissions and Automation in Design, Vol. 108, June 1986, pp 176-182.

[Kim 86b] S.S. Kim and M.J. Vanderploeg, *QR Decomposition for State Space Representation for Constrained Mechanical Dynamic Systems.* ASME J. Mechanisms, Transmissions and Automation in Design, Vol. 108, June 1986, pp 183-188.

[Kim 88] S.S. Kim and E.J. Haug, *A Recursive Formulation for Flexible Multibody Dynamics Part I: Open Loop Systems.* Computer Methods in Applied Mechanics and Engineering, 71, 1988, pp 293-314.

[Lee 88] C.S.G. Lee and P.R. Chang, *Efficient Parallel Algorithms for Robot Forward Dynamics Computation.* IEEE Trans. Systems, Man, and Cybernetics, Vol. SMC-18, No. 2, March/April 1988, pp 238-251.

[Lilly 91] K.W. Lilly and D.E. Orin, *Alternate Formulations of the Manipulator Inertia Matrix.* The Int. J. Robotics Res., Vol. 10, No. 1, February 1991.

[Liu 83] P.S. Liu and T.Y. Young, *VLSI Array Design Under Constraint of Limited I/O Bandwidth.* IEEE Trans. Computers, Vol. C-32, No. 12, December 1983, pp 1160-1170.

[Luh 80] J.Y.S. Luh, M.W. Walker, and R.P. Paul, *On-line Computational Scheme for Mechanical Manipulators.* ASME J. Dynamical Systems, Measurement and Control. Vol. 102, No. 2, June 1980, pp 69-76.

[Mani 84] N.K. Mani and E.J. Haug, *Use of Singular Valued Decomposition for Analysis and Optimization of Mechanical System Dynamics.* Technical Report 84-13, University of Iowa, Iowa City, August 1984.

[Murray 89] J.J. Murray and G.H. Lovell, *Dynamic Modeling of Closed-Chain Robotic Manipulators and Implications for Trajectory Control.* IEEE Trans. Robotics and Automation, Vol. RA-5, No. 4, August 1989.

[Nikravesh 84] P.E. Nikravesh, *Some Methods for the Dynamic Analysis of Constrained Mechanical Systems: A Survey.* Computer Aided Analysis and Optimization of Mechanical System Dynamics, ed. E.J. Haug. Springer Verlag, Heidelberg, 1984.

[Orlandea 77] N. Orlandea, M.A. Chace, and D.A. Calahan, *A Sparsity Oriented Approach to Dynamic Analysis and Design of Mechanical Systems, Part I and II.* ASME J. Engr. Indust., Vol. 99, August 1977, pp 773-784.

[Park 86] T.W. Park and E.J. Haug, *A Hybrid Numerical Integration Method for Machine Dynamic Simulation* ASME J. Mechanisms, Transmissions and Automation in Design, Vol. 108, 1966, pp 211-216.

[Press 88] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C.* Cambridge University Press, Cambridge 1988.

[Roberson 66] R.E. Roberson and J. Wittenberg, *A Dynamic Formalism for an Arbitrary Number of Interconnected Rigid Bodies, with Reference to the Problem of Satellite Attitude Control.* Proc. 3rd Congress IFAC (London 1966), Vol. 1, Book 3, Paper 46D. Butterworth, London.

[Roberson 88] R.E. Roberson and R. Schwertassek, *Dynamics of Multibody Systems.* Springer Verlag, Berlin, 1988.

[Rodriguez 87] G. Rodriguez, *Kalman Filtering, Smoothing, and Recursive Robot Arm Forward and Inverse Dynamics.* IEEE Journal of Robotics and Automation, Vol. RA-3, No. 6, December 1987.

[Rodriguez 88] G. Rodriguez, *Recursive Mass Matrix Factorization and Inversion.* JPL publication 88-11, March 15, 1988.

[Rulka 90] W. Rulka, *SIMPACK, a Computer Program for Simulations of Large-Motion Multibody Systems.* in Handbook of Multibody Systems, W. Schiehlen (ed.) Springer-Verlag, Berlin, 1990.

[Schwertassek 84] R. Schwertassek and R.E. Roberson, *A State Space Dynamical Representation for Multibody Mechanical Systems Part II: Systems with Closed Loops.* Acta Mechanica 51, 1984, pp 15-29.

[Schwertassek 89] R. Schwertassek and W. Rulka, *Aspects of Efficient and Reliable Multibody System Simulation.* Unpublished paper, 1989.

[Sepehri 90] N. Sepehri, *Dynamic Simulation and Control of Teleoperated Heavy-Duty Hydraulic Manipulators.* PhD Thesis, Department of Mechanical Engineering, University of British Columbia, 1990.

[Silver 82] W.M. Silver, *On the Equivalence of Lagrangian and Newton Euler Dynamics for Manipulators.* Int. J. Robotics Res., Vol. 1, 1982, pp 118-128.

[Vereschagin 74] A.F. Vereschagin, *Computer Simulation of the Dynamics of Complicated Mechanisms of Robot-Manipulators.* Engineering Cybernetics, No. 6, 1974, pp 65-70.

[Walker 82] M.W. Walker and D.E. Orin, *Efficient Dynamic Computer Simulation of Robotic Mechanisms.* ASME J. Dynamical Systems, Measurement and Control. Vol. 104, 1982, pp 205-211.

[Wehage 82] R.A. Wehage and E.J. Haug, *Generalised Coordinate Partitioning for Dimension Reduction in the Analysis of Constrained Dynamic Systems.* ASME J. Mechanical Design, Vol. 104, 1982, pp 247-255.

[Whittaker 27] E.T. Whittaker, *A Treatise on the Analytical Dynamics of Particles and Rigid Bodies.* Cambridge University Press, Cambridge, 1927.

[Wittenberg 77] J. Wittenberg, *Dynamics of Systems of Rigid Bodies.* Teubner Stuttgart, Germany, 1977.

[Zheng 84] Y.F. Zheng, *Modeling, Control and Computer Simulation of a Three Dimensional Robotic System with Application to Biped Locomtion.* PhD Thesis, Department of Electrical Engineering, Ohio State University, 1984.

[Zheng 86] Y.F. Zheng and H. Hemami, *Computation of Multibody Dynamics by a Multiprocessor Scheme.* IEEE Trans. Systems, Man, and Cybernetics, Vol. SMC-16, No. 1, Jan. 1986, pp 102-110.