1994

Computational dynamics for robotic systems on land and under water

McMillan, Scott

UMI

Order Number 9505256

Computational dynamics for robotic systems on land and under water

McMillan, Scott, Ph.D.

The Ohio State University, 1994

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

Computational Dynamics for Robotic Systems on Land and Under Water

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

by

Scott McMillan, M.S.E.E., B.S.

* * * * *

The Ohio State University

1994

Dissertation Committee:

David E. Orin

Robert B. McGhee

P. Sadayappan

Approved by

[redacted]

Adviser

Department of Electrical Engineering

To my family

ACKNOWLEDGMENTS

I would first like to thank my advisor, Dr. David E. Orin, for the guidance and support he has provided during my research. I am grateful for the abundant time and patience he has afforded me. I would also like to thank the members of my committee, Dr. P. Sadayappan and Dr. Robert B. McGhee, for their valuable comments and discussions along the way, and their careful reading of this dissertation.

Dr. Gregory R. Baker of the Mathematics Department at OSU has my gratitude for his outstanding courses in Scientific Computing and discussions regarding topics contained herein. Thanks also go to Professor Anthony Healey at the Naval Postgraduate School for informative discussions on hydrodynamics, and to James B. Newman of Monterey Bay Aquarium Research Institute for access to and involvement in the *Tiburon* project.

My special gratitude goes to my family for their constant support and encouragement throughout my educational career.

Finally, I would like to recognize the fellowship support I have received from The Ohio State University, the E. I. Dupont de Nemours Corporation, and AT&T Bell Laboratories. This research was also supported by the National Science Foundation from NSF Grant No. BCS-9109989 to the Naval Postgraduate School and No. BCS-9311269 to The Ohio State University.

VITA

March 8, 1966	Born—Wilmington, DE, U.S.A.
1984 - 1988	R. F. Poole Scholar, Clemson University, Clemson, SC, U.S.A.
May 1988	B.S. Computer Engineering Clemson University
Summer 1988	Digital Design Engineer Experimental Physics Laboratory E.I. DuPont de Nemours & Co., Inc. Wilmington, DE, U.S.A.
1988 - 1992	DuPont Fellow in Electrical Engineering The Ohio State University Columbus, Ohio, U.S.A.
1988 - Present	University Fellow The Ohio State University
June–September 1989	Graduate Research Associate The Ohio State University
January –June 1990	Graduate Teaching Associate The Ohio State University
June 1990	M.S. Electrical Engineering The Ohio State University
1990 - 1994	AT&T Ph.D. Scholar The Ohio State University
Summer 1991, 1992	Member of Technical Staff Computing Architectures Res. Dept. AT&T Bell Laboratories Holmdel, NJ, U.S.A.
Summer 1993	Visiting Engineer Naval Postgraduate School/MBARI Monterey, CA, U.S.A.

PUBLICATIONS

"Real-Time Simulation of a Robotic Manipulator Using a Parallel Integration Method," Masters Thesis, The Ohio State University, Columbus, Ohio, June 1990.

"Real-Time Robot Dynamic Simulation on a Vector/Parallel Supercomputer," *IEEE Int. Conf. on Robotics and Automation*, Sacramento, pp. 1836–1841, April 1991 (with D. E. Orin and P. Sadayappan).

"Parallel Real-Time Dynamic Simulation of Robots," *ASCE Engineering Mechanics Specialty Conference: Mechanics Computing in 1990's and Beyond*, Columbus, Ohio, May 1991 (with D. E. Orin and P. Sadayappan).

"Towards Super-Real-Time Simulation of Robotic Mechanisms Using a Parallel Integration Method," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 384–391, March/April 1992 (with D. E. Orin and P. Sadayappan).

"Efficient Dynamic Simulation of Multiple-Manipulator Systems with Singularities," *IEEE Int. Conf. on Robotics and Automation*, Nice, France, pp. 299–304, May 1992 (with D. E. Orin and P. Sadayappan).

"Parallel Methods for Dynamic Simulation of Multiple Manipulator Systems," *Proc. 5th Annual Workshop on Aerospace Computational Control*, Santa Barbara, CA, pp. 267–281, August 1992 (with D. E. Orin and P. Sadayappan).

"Efficient Dynamic Simulation of Multiple-Manipulator Systems with Singular Configurations," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 24, no. 2, pp. 306–312, February 1994 (with P. Sadayappan and D. E. Orin).

"Efficient Dynamic Simulation of an Unmanned Underwater Vehicle with a Manipulator," *IEEE Int. Conf. on Robotics and Automation*, San Diego, CA, pp. 1133–1140, May 1994 (with D. E. Orin and R. B. McGhee).

"Parallel Dynamic Simulation of Multiple Manipulator Systems: Temporal vs. Spatial Methods," To appear in *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 24, no. 7, July 1994 (with P. Sadayappan and D. E. Orin).

Fields of Study

Major Field: Electrical Engineering

Studies in Robotics:	Professors D. E. Orin, C. A. Klein, K. J. Waldron
Studies in Computer Engineering:	Professors S. C. Ahalt, F. Özgüner, D. E. Orin
Studies in Control Engineering:	Professors S. Yurkovich, Ü. Özgüner, K. Passino
Studies in Mathematics:	Professors G. R. Baker, A. Bloch, G. Majda

Table of Contents

List of Tables

LIST OF FIGURES

CHAPTER I

Introduction

1.1 Motivation

For decades the single chain robotic manipulator has been the primary tool of manufacturers and researchers alike. Its capabilities are limited both by the simplicity of its mechanical design, and in many cases, its conservative control algorithms. Currently, robotics is being applied to ever more complex problem domains which require more complex systems and control algorithms that push the limits of the systems' mechanical capabilities. These new systems can have multiple chains, mobile bases, and experience environments that are vastly different from the designer's workbench.

Multiple manipulator systems for factory settings are being developed that must cooperate to lift larger loads or perform more complex manipulation tasks for assembly. These systems are characterized by fixed bases, and closed kinematic loops. Telerobotic systems for applications such as hazardous waste cleanup are also under development. These systems have mobile bases with one or more manipulators. Development of multilegged vehicles is also being pursued for traversal over rugged terrain.

Use of these complex robotic systems is becoming increasingly desirable in the underwater environment for a range of applications such as biological and geophysical research, construction, salvage, rescue and recovery, and maintenance and repair of man-made underwater structures. Such applications may require manipulators mounted on mobile platforms with sophisticated control systems. Multilegged vehicles such as the one shown in Figure 1 are also under development for surveying and construction applications [1]. The importance of these *unmanned* underwater robotic vehicle (URV) systems for marine research and subsea development continues to grow because their manned counterparts are much more expensive to develop and maintain [2]. This increase in use has brought about a concomitant need for accurate simulations of these systems [3].

With the desire for increased capabilities and performance of these robotic systems comes an increase in the cost of their design, development, and operation. Dynamic simulation has proven to be a cost effective tool for single chain manipulators where ten years of research has led to the development of a number of very efficient algorithms and commercial software packages. Researchers continue to search for more efficient dynamics algorithms for complex land-based systems [4, 5, 6, 7], but the application of these ideas has not been applied to URV systems. The work presented here contributes to this research effort.

In both of these domains, the use of dynamic simulation can save significant time and money during the mechanical and control design, test and evaluation phases for these systems. Early in the design process, the use of simulation to perform a

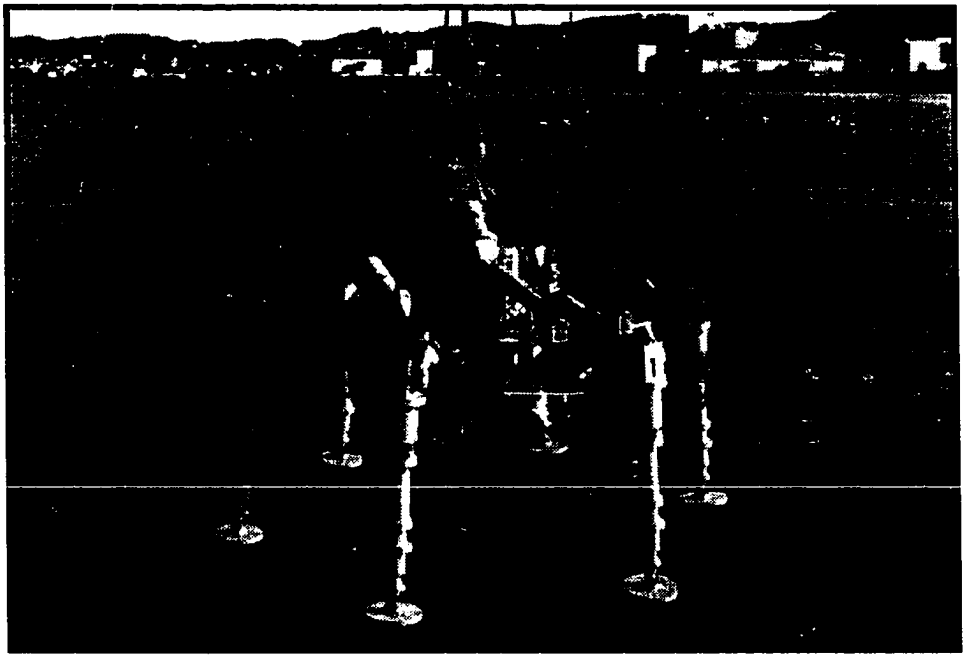


Figure 1: The *Aquarobot* hexapod [1].

thorough dynamic analysis can reduce the need for costly prototypes by eliminating many candidate designs. This not only saves on cost but also the time needed to construct successive prototype generations. In addition, simulators can aid in the design of control algorithms. By using the simulated robotic system to test such algorithms, potentially damaging instabilities due to algorithm errors are eliminated, and risks encountered when the control system is finally implemented in hardware are reduced.

Until only very recently, dynamic simulation of robotic systems has been considered an off-line problem, and this is adequate for the design applications mentioned above. With an increase in computing capabilities and the development of more

efficient algorithms, real-time dynamic simulation of these systems is also within reach. As a result, interest in on-line simulation and its applications is also increasing, such as hardware- and human-in-the-loop operation. With hardware-in-the-loop simulation, control system hardware can also be tested off-line by interfacing it to a computer performing a real-time dynamic simulation of the system [8]. This further reduces the risk to the robotic systems, and is especially important in URV development because the underwater environment is unstructured and unpredictable.

Human-in-the-loop applications can also be implemented when a real-time simulation is coupled with a realistic 3-D graphical display of the system. One such application is a simulator to train operators in the use of the robotic systems before allowing hands-on training. This is analogous to the use of flight simulators to train aviators. Another application can be found in the teleoperation of space-based and untethered underwater vehicles. Because significant delays (speed-of-light, acoustic propagation) occur in communications with such vehicles, human control is significantly degraded. By providing the teleoperator with a simulated display of the system, on-line with no delay, enhanced performance of human-machine interaction can be realized [9, 10].

Beyond real-time computational rates, the application of *super*-real-time simulation has also been considered. In this scheme, seconds of motion trajectory need to be simulated in milliseconds [11]. It is desired in some applications such as in advanced control schemes where trajectory planning is used. This has been proposed for multilegged vehicles [12] where prediction of the action of the present control is

used to ensure safety and stability along a desired trajectory.

1.2 Objectives

For dynamic simulation, the primary problem to be solved is called forward or direct dynamics. This problem requires the determination of motion of the system (accelerations of all of the bodies) given the set of forces acting on the system (external forces and internal joint actuator torques/forces). Once the accelerations are known, numerical integration is used to determine the updated state (positions and velocities) of the system. For multibody robotic systems, the dynamic equations of motion are complex and their computation dominates in the simulation. Therefore, they are the primary obstacle to achieving the required computational rates for the on-line applications. This problem is compounded by increases in the structural and task complexity found in the systems considered in this dissertation, which have multiple chains and redundant numbers of degrees of freedom, operating at higher speeds, with closed kinematic loops, and may operate in the underwater environment.

This dissertation focuses on two classes of multiple chain robotic systems. They are comprised of one reference member which interacts with a number (m) of supporting serial chains. These are referred to as *simple closed chain mechanisms* where the removal of the reference member breaks all closed loops in the system [13]. For Type 1 systems, the ends of each chain are attached to the reference member through an actuated joint. The chains are then closed through contact of the opposite end of the chain with the ground. The typical example of a Type 1 system is the legged vehicle. In Type 2 systems, the chains are attached to a common base (e.g., the

ground) through an actuated joint, and they are closed through an unpowered contact or grasp of the reference member (e.g., a load). Examples of this are multiple cooperating manipulators or dextrous hands.

In an effort to achieve real-time rates, the development of computationally efficient dynamic simulation algorithms for these systems is the primary objective of this research. A number of efficient algorithms have been previously developed to simulate the dynamics of robotic mechanisms. The most efficient approaches to this problem employ recursive equations. Most of these methods are based on either the Composite Rigid Body (CRB) method developed by Walker and Orin (Method 3, [14]) or the Articulated Body (AB) method developed by Featherstone [15]. Since their inception, improvements have been made to the computational efficiency of these algorithms. Balafoutis and Patel [16] have developed a most efficient implementation of the CRB algorithm, and Brandl, Johanni, and Otter [17] have developed an efficient implementation of the AB algorithm.

The AB approach has an advantage because its computational requirements grow linearly with the number of degrees of freedom, whereas the CRB method has cubic complexity. For typical serial chain manipulators, the AB algorithms are accepted as the most efficient. However, if the number of links is small enough or multiple, closed chain systems are being considered, algorithms based on the CRB method can be more efficient. Therefore, the main objective of this dissertation, is to develop new efficient algorithms based on these approaches and determine which is the most efficient for the systems of interest. In the process, both approaches will be expressed

using uniform notation and equivalent methods of modeling these mechanisms so that comparisons are fair and the correct conclusions can be reached.

Parallelization of these algorithms is another technique which can be used to increase the computational rates of the simulation. Fine-grain parallel algorithms for single serial chains have been developed in most previous work [18, 19, 20, 21, 22, 23]. These approaches, while designed to perform well on special-purpose parallel architectures, are much less efficient when implemented on the general-purpose parallel systems widely available today. These systems typically have a limited number of loosely coupled processors that are not designed to handle the large amounts of interprocessor communication and synchronization required by the fine-grain tasks. One form of parallelism in the simulation explored in this work is related to the structural parallelism in the multi-chain mechanisms and is called *spatial parallelism.* With this, the forward dynamics computation is parallelized. The second, is *temporal* parallelism which is implemented with parallel numerical integration. They can also be combined to further increase the computational rates.

The extension of these algorithms to space-based robots with rigid links is straightforward – requiring only the removal of gravity. The use of robotic systems in an underwater environment is relatively new, and efficient multibody dynamics algorithms have not been developed for these systems. More importantly, the application of existing algorithms for land-based systems to URVs is not straightforward because of complex hydrodynamic effects. Therefore, an additional objective for development of a simulator for these systems is the study of the hydrodynamic forces

that are exerted on submerged rigid bodies. Special effort must be made to express these effects using a notation consistent with existing robotics algorithms, and a detailed explanation of how to incorporate these effects is given.

This work culminates in the development of a software package in C++, called **DynaMechs***, that is capable of performing the dynamic simulation for a large class of tree-structured robotic systems having star topologies. These systems range from a single chain with a fixed base to multiple chains with a mobile base. Object oriented design (OOD) techniques are explored to aid in the implementation of an efficient algorithm that is capable of simulating fixed and mobile bases, single and multiple chains with any number of links, and various link types. Techniques such as object hierarchies, encapsulation, inheritance, and polymorphism are employed, and issues such as code reuse and maintenance are discussed. The result is a flexible, efficient, and maintainable package.

1.3 Spatial Notation

Since the dynamics algorithms discussed in this dissertation are complex, the purpose of this section is to introduce some of the notation employed. To begin, the convention for labeling the rigid bodies in simple closed chain mechanisms is presented. First, the reference member is indicated by an r. The links in a serial chain are numbered in order from 1, nearest the base, to N, which is farthest from the base. Note that for Type 1 systems, link 1 is attached through an actuated joint to the reference member, while link N is in contact with the reference member for

*Pronounced dynamics, and stands for dynamics of mechanisms.

Type 2 systems. With multiple chains these links are labeled with ordered pairs where the first is the chain number, from 1 to m, and the second is the link number. These labels are used with spatial notation to specify the dynamic and kinematic quantities associated with each rigid body in the system.

Traditionally, robot dynamics and kinematics algorithms have used three-dimensional angular and linear components of velocity, acceleration and force quantities separately, which leads to overly complex expressions for these algorithms. The purpose for the development of spatial notation was to alleviate some of this complexity and make the equations easier to comprehend. The notation used in this dissertation is based on that used by Lilly and Orin [24]. With this approach, the corresponding angular and linear components of velocity, acceleration, and force are combined into six-element vectors [15, 25]. For example, the spatial velocity of link i is given as follows:

$$\mathbf{v}_i = \begin{bmatrix} \boldsymbol{\omega}_i \\ \boldsymbol{v}_i \end{bmatrix}, \tag{1.1}$$

where $\boldsymbol{\omega}_i$ is the angular velocity of link i expressed in the ith coordinate system, and $\boldsymbol{v}_i$ is its linear velocity. Note that the convention in this dissertation is to use an italic bold variable, such as $\boldsymbol{v}$, to refer to a Cartesian (three-dimensional) vector representing either a rotational or translational quantity, and a block bold variable, such as $\mathbf{v}$, to refer to a spatial (six-dimensional) vector. Similarly, its spatial acceleration is given as follows:

$$\mathbf{a}_i = \begin{bmatrix} \dot{\boldsymbol{\omega}}_i \\ \boldsymbol{a}_i \end{bmatrix}. \tag{1.2}$$

where $\dot{\omega}_i$ is its angular acceleration and a_i is its linear acceleration. A spatial force exerted onto link i is given as:

$$\mathrm{f}_i = \begin{bmatrix} n_i \\ f_i \end{bmatrix}, \tag{1.3}$$

where n_i is a moment and f_i is the three-dimensional force vector.

The 6×6 spatial transformation matrix, ${}^{i}\mathbf{X}_{i-1}$, is used to transform spatial vectors between coordinate systems that are fixed to each rigid body. For example, a spatial velocity is transformed to a link's outboard neighbor as follows:

$${}^{i}\mathrm{v}_{i-1} = {}^{i}\mathbf{X}_{i-1}\,{}^{i-1}\mathrm{v}_{i-1}, \tag{1.4}$$

where ${}^{i-1}\mathrm{v}_{i-1}$ is the velocity of link $i-1$ expressed in its own coordinate system, and ${}^{i}\mathrm{v}_{i-1}$ is the same vector expressed with respect to link i's coordinate system. For brevity, the leading superscript is often omitted from ${}^{i-1}\mathrm{v}_{i-1}$ when it is the same as the subscript. The transpose of the spatial transformation matrix is used to transform spatial forces in the other direction:

$${}^{i-1}\mathrm{f}_i = {}^{i}\mathbf{X}_{i-1}^{T}\,\mathrm{f}_i. \tag{1.5}$$

The spatial transformation between links $i-1$ and i is expressed as:

$${}^{i}\mathbf{X}_{i-1} = \begin{bmatrix} {}^{i}R_{i-1} & 0 \\ {}^{i}R_{i-1}\,{}^{i-1}\tilde{p}_i^{\,T} & {}^{i}R_{i-1} \end{bmatrix}. \tag{1.6}$$

where ${}^{i}R_{i-1}$ is the rotation matrix describing the change in orientation between link $(i-1)$'s and i's coordinate systems, and ${}^{i-1}p_i$ is the position of link i's coordinate system with respect to link $(i-1)$'s. The tilde above the position vector signifies that its components should be combined in a skew symmetric matrix such that

$\tilde{\boldsymbol{b}}\boldsymbol{a} = \boldsymbol{b} \times \boldsymbol{a}$. For single degree-of-freedom (DOF) revolute and prismatic joints, modified Denavit-Hartenberg (MDH) parameters as defined in [26] can be used to specify these quantities as shown in Appendix A.

Mass and inertia (first and second moments) properties of a rigid body may also be combined in a 6×6 matrix. For link i, this spatial inertia matrix is defined as follows:

$$\mathbf{I}_i = \begin{bmatrix} \bar{\boldsymbol{I}}_i & \tilde{\boldsymbol{h}}_i \\ \tilde{\boldsymbol{h}}_i^T & m_i \mathbf{1}_3 \end{bmatrix}, \tag{1.7}$$

where $\bar{\boldsymbol{I}}_i$ is the 3×3 moment of inertia tensor for link i defined with respect to its own coordinate system. The 3×1 vector, $\boldsymbol{h}_i$, is its first mass moment and is equal to $(m_i\ \boldsymbol{s}_i)$ where m_i is its mass and $\mathbf{s}_i$ is the vector from the origin of the link's coordinate system to its center of mass. Finally, $\mathbf{1}_3$ is the 3×3 identity matrix. Note that $\mathbf{I}_i$ is constant because it is defined with respect to the coordinate system fixed to the link.

The spatial transformation matrix is also used to transform the spatial inertia matrix to an adjacent coordinate system. This is accomplished with the following congruence transformation [27]:

$${}^{i-1}\mathbf{I}_i = {}^{i}\mathbf{X}_{i-1}^T\, {}^{i}\mathbf{I}_i\, {}^{i}\mathbf{X}_{i-1}, \tag{1.8}$$

where ${}^{i}\mathbf{I}_i$ is the spatial inertia of link i expressed with respect to its own coordinate system, and ${}^{i-1}\mathbf{I}_i$ is the inertia of the link expressed with respect to coordinate system $i-1$.

Finally, a spatial quantity is related to the joints that connect the links of a serial chain. Called the joint motion subspace or just the joint motion axis, $\boldsymbol{\phi}_i$ is

a $6 \times n_i$ matrix where n_i is the number of degrees of freedom in the joint. It is defined in a coordinate system fixed to the link and is therefore constant. For single DOF joints used extensively in this dissertation in which the z-axis of the link's coordinate system lies along the joint axis, ϕ_i is given by $[0\ 0\ 1\ 0\ 0\ 0]^T$ for revolute joints and $[0\ 0\ 0\ 0\ 0\ 1]^T$ for prismatic joints. As a result, the joint positions, q_i, velocities, $\dot{q}_i$, accelerations, $\ddot{q}_i$, and its inputs, τ_i, are given by scalars associated with these vectors.

1.4 Organization

In Part I of this dissertation, the simulation of land-based systems with multiple chains is studied. Chapters II–IV present the development of simulation algorithms for multiple manipulator systems, and Chapter V deals with multilegged vehicles.

In Chapter II, a dynamics algorithm for multiple manipulator systems lifting a common object is developed. Lilly and Orin's [24] operational space formulation of the dynamics is presented where arbitrarily constrained (rigid) grasps are assumed and the resulting system contains closed, kinematic loops. An efficient implementation of these equations, based on the CRB method and having $O(mN^3)$ complexity, is developed and the resulting computational requirements are determined.

In Chapter III, parallelization of this algorithm is investigated. The manipulator system's parallel topology leads to spatial parallelism where the computation of the dynamics for each serial chain is performed simultaneously. Temporal parallelism can be achieved with parallel numerical integration algorithms where the dynamics of the entire system for different points in time is computed simultaneously. Combining

both forms leads to a two-dimensional parallelism and still greater increases in the computation rates. The resulting algorithm is implemented on a Cray Y-MP8 and speedup results are reported for a four PUMA system.

Chapter IV addresses the problems introduced by manipulator singularities on the dynamics algorithm. Mathematical characterization of the dynamic equations of motion when one or more manipulators becomes singular is made. Based on this study, more robust algorithms are developed, and the effect on the computational requirements is determined. This also leads to the development of a new, more efficient dynamics algorithm for dual arm systems with and without singularities.

In Chapter V, a different approach for the simulation of multilegged vehicles is taken. With soft constraints at the feet, the decoupled tree-structure (DTS) approach [7] can be taken where compliance is modeled at the ground contacts and the closed kinematic loops are broken. A new CRB-based algorithm with $O(mN^3)$ complexity is developed for these multiple open-chain systems having star topologies. Efficient kinematic modeling for these systems is also presented, and the computational requirements are determined. Because N is small for these systems, the resulting algorithm is more efficient than the competing AB methods which have a computational complexity of $O(mN)$.

In Part II, the development of a real-time dynamic and hydrodynamic simulation system for underwater robotic vehicle (URV) systems is presented. Chapter VI examines the hydrodynamic forces that are exerted on submerged rigid bodies such as added mass, drag, fluid acceleration and buoyancy. A number of assumptions

must be made to simplify the computation of these forces, and the computational requirements for each is presented.

In Chapter VII, the efficient AB algorithm, for the simulation of land-based robotic systems, forms the basis of a new algorithm developed for URV systems. To accomplish this, a very efficient method for incorporating the hydrodynamic forces is developed and the equations for the dynamics of multiple chains and a mobile base are examined. The computational requirements of the resulting O(mN) algorithm are listed.

In Chapter VIII, OOD techniques are presented for the implementation of a simulation system, called **DynaMechs**, that is based on this algorithm. It is capable of simulating a large class of tree-structured robotic systems, having star topologies, which include fixed and mobile base systems with any number of serial chains containing a wide variety of joint types such as revolute, prismatic, and ball-in-socket joints. To achieve this functionality, techniques such as encapsulation, inheritance, and polymorphism are applied, and a very efficient simulator results. Performance results and examples of graphical interfaces on a Silicon Graphics workstation are also presented.

Finally, in Chapter IX, the contributions of this work are summarized and future areas of research are suggested. Two appendices are also included that examine the computational requirements of many common operations used in the simulation of these robotic systems. In Appendix A, kinematic modeling of serial chains using a modified Denavit-Hartenberg (MDH) notation is presented. Using the MDH param-

eters, new methods for achieving more efficient transformation operations of vectors and matrices are presented. In Appendix B, single chain robotic algorithms for inverse dynamics, computation of joint-space inertia matrices, and the AB method fcr forward dynamics are presented.

Part I

Multiple Chain Dynamic Simulation on Land

CHAPTER II

Dynamics of Multiple Manipulator Systems: An Operational Space Approach

2.1 Introduction

In this and the following two chapters, the problem of simulating multiple manipulators that cooperate to lift large loads is addressed. Each manipulator rigidly grasps a common object, resulting in closed kinematic chains which requires special consideration in the development of the dynamic equations for this system. Since real-time and even super-real-time simulation is desirable, computationally efficient dynamics algorithms are a high priority. This chapter addresses this facet of the problem specifically.

While the problem of efficient dynamics algorithms for single, open-chain mechanisms have been extensively studied [14, 15, 16, 17, 28, 29], much less has been published on simulation algorithms for systems with closed chains. In [4], Oh and Orin extended an $O(N^3)$ approach in [14] to develop a dynamic simulation algorithm based on equation augmentation using the constraint forces applied to the reference member resulting from the closed kinematic loops. However, this results in a large system of equations to be solved, with an $O(m^3N^3)$ complexity where m is the number of chains in the system, and N is the number of degrees of freedom

in each chain.

The $O(N)$ approach to dynamic simulation of single chains developed by Featherstone [15] has also been extended to the simulation of multiple chain mechanisms. Rodriguez, Jain, and Kreutz-Delgado [30] develop mathematical tools called linear operators to derive simulation algorithms for general robotic systems. This was used in [31, 32] to develop the dynamic equations for multiple manipulator systems grasping a common object on fixed and mobile bases respectively. The resulting algorithms have a computational complexity of $O(mN)$ when the manipulators are in nonsingular configurations.

None of this work, however, has addressed the issue of detailed computational requirements which is important in obtaining efficient algorithms. To this end, Lilly and Orin have studied this issue for single and multiple closed-chain mechanisms [24, 33]. The efficient algorithms developed in their work have computational complexities $O(mN^3)$ and $O(mN)$, respectively, but the former requires fewer operations for $N < 21$. The algorithm developed in this chapter builds on these results and presents a new and more efficient algorithm for the simulation of these systems. This also provides a background for key results in later chapters of this Part. Parallelization of this algorithm is examined in Chapter 3 in order to achieve higher, possibly real-time, computational rates. Problems arising from singular manipulators are studied in Chapter 4. Finally in Chapter 5, efficient multiple chain algorithms are also developed specifically for multilegged vehicles that also have a computational complexity of $O(mN^3)$.

In the following section, the dynamic equations for a general multiple manipulator system are developed with arbitrarily constrained grasps on the reference member. The operational space [34] formulation developed by Lilly and Orin in [24] is used which simplifies the conceptual framework for the development of the algorithm. In Section 2.3, it is refined assuming rigid grasps on the reference member. The result is a very efficient algorithm for the simulation of multiple manipulator systems under consideration. Finally, the efficient implementation and resulting computational requirements for this algorithm are presented in Section 2.4. The system used for this analysis has four PUMA 560 manipulators each with six revolute joints.

2.2 Multiple Closed-Chain Dynamics

In this section, the equations for efficient dynamic simulation of a multiple, closed-chain manipulator system are developed. Figure 2 illustrates the system that is used as the model in this chapter. It contains m manipulators, each with N degrees of freedom, that are grasping a common load, called the reference member. The number of degrees of freedom in each chain is assumed to be the same for all chains for simplicity and without loss of generality. In this development, grasps with arbitrary motion constraints are assumed and can be modeled by an augmented chain that has an additional unpowered joint at the point of contact, called the General Contact Joint, and an extra link fixed to the reference member. This joint can have more than one degree of freedom, and can be simulated efficiently using the general joint model developed in [24, 35]. The extra link at the end of the augmented chain is

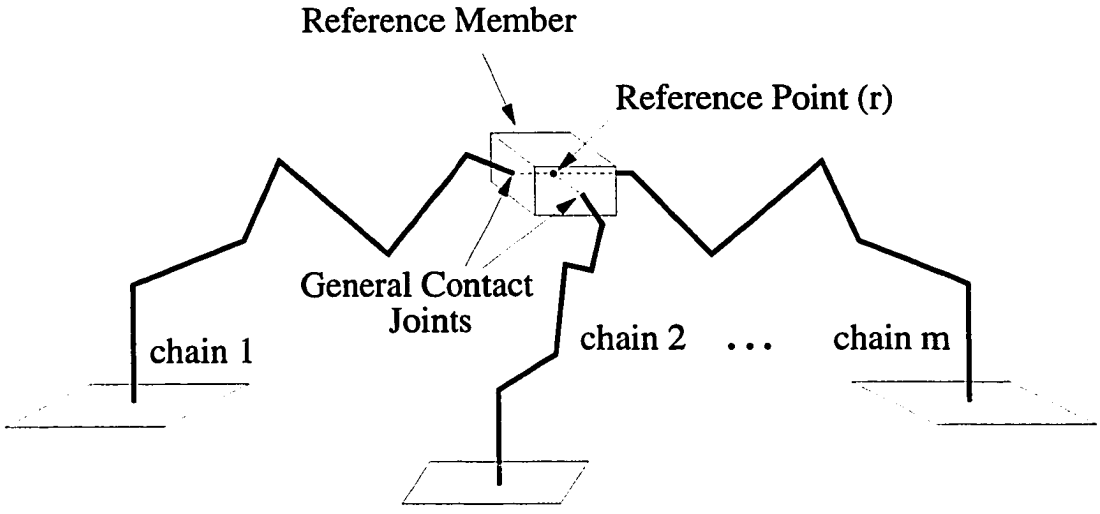


Figure 2: The multiple manipulator system grasping a common object.

fictitious as shown by the dotted lines in the figure.

To begin, each manipulator is regarded as a simple, closed chain where the motion of its base is completely known. Numbering the manipulators in the system 1 through m, each is governed by the dynamic equations of motion which can be written as follows:

$$\boldsymbol{\tau}_k = \mathbf{H}_k(\mathbf{q}_k)\ddot{\mathbf{q}}_k + \mathbf{C}_k(\mathbf{q}_k, \dot{\mathbf{q}}_k) + \mathbf{G}_k(\mathbf{q}_k) + \mathbf{J}_k^T(\mathbf{q}_k)\mathbf{f}_k, \tag{2.1}$$

where the k index refers to the manipulator number,

$\boldsymbol{\tau}_k$ $N \times 1$ vector of torques/forces of the joint actuators,

$\mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k$ $N \times 1$ vectors of joint positions, rates, and accelerations,

$\mathbf{H}_k$ $N \times N$ positive definite joint space inertia matrix,

$\mathbf{C}_k$ $N \times 1$ vector specifying centrifugal and coriolis effects,

$\mathbf{G}_k$ $N \times 1$ vector specifying the effects due to gravity,

$\mathbf{J}_k$ $6 \times N$ Jacobian matrix, and

$\mathbf{f}_k$ 6×1 force exerted by tip of chain k on the reference member.

Since we are interested in the solution to the Forward Dynamics or simulation problem, the state of the system, consisting of joint positions and rates, and the input joint torques/forces are given. The joint accelerations and the tip forces for each chain must then be computed.

By grouping some terms from Eq. (2.1) and solving for the joint accelerations, we obtain the following equation for each chain [24]:

$$\ddot{\mathbf{q}}_k = \ddot{\mathbf{q}}_{k_{open}} - \mathbf{H}_k^{-1}\mathbf{J}_k^T\mathbf{f}_k, \tag{2.2}$$

where $\ddot{\mathbf{q}}_{k_{open}}$ is the vector of joint accelerations for chain k if it were not contacting the reference member resulting in a zero tip force. This is called its open-chain solution which was implemented in [14, 36]. Note that this quantity, as well as $\mathbf{H}_k$ and $\mathbf{J}_k$, depend only on the system state and input joint torques/forces and may be computed from known quantities.

Because the last link of each augmented chain is rigidly attached to the reference member, a point, called the reference point in Figure 2, can be chosen on the reference member that is also fixed with respect to each end-effector. It then becomes convenient to use an operational space approach [34] in the development of the dynamic equations. Hence, this point will serve as the location at which all of the chains' spatial quantities will be resolved. Analogous to Eq. (2.2), the operational space formulation for the dynamics of a closed-chain manipulator can be found by introducing the following well known relationship, using the manipulator

Jacobian, between joint velocities and spatial (Cartesian) velocity of the end-effector of a manipulator at the reference point, $\dot{\mathbf{x}}_k$:

$$\dot{\mathbf{x}}_k = \mathbf{J}_k \dot{\mathbf{q}}_k. \tag{2.3}$$

Taking the time derivative of both sides yields the equation for the closed-chain acceleration of the tip:

$$\ddot{\mathbf{x}}_k = \dot{\mathbf{J}}_k \dot{\mathbf{q}}_k + \mathbf{J}_k \ddot{\mathbf{q}}_k, \tag{2.4}$$

and substituting $\ddot{\mathbf{q}}_k$ from Eq. (2.2) yields the following for $k = 1, \ldots, m$:

$$\ddot{\mathbf{x}}_k = \left(\dot{\mathbf{J}}_k \dot{\mathbf{q}}_k + \mathbf{J}_k \ddot{\mathbf{q}}_{k_{open}}\right) - \left(\mathbf{J}_k \mathbf{H}_k^{-1} \mathbf{J}_k^T\right) \mathbf{f}_k, \tag{2.5}$$

$$= \ddot{\mathbf{x}}_{k_{open}} - \mathbf{\Lambda}_k^{-1} \mathbf{f}_k, \tag{2.6}$$

where $\ddot{\mathbf{x}}_{k_{open}}$ is the acceleration of the tip of chain k (at the reference point on the end-effector) if it were in an open, unconstrained configuration, and the matrix, $\mathbf{\Lambda}_k^{-1}$, is the inverse of its operational space inertia matrix [34].

The physical interpretation of $\mathbf{\Lambda}_k$ is an inertial quantity which combines the dynamic properties of the links of the chain and projects them to its tip. It is the physical resistance that is "felt" when a force is exerted on the tip, and it defines the relationship between the force that is applied *onto* the tip, $-\mathbf{f}_k$, and the resulting acceleration of the tip of the chain, $\ddot{\mathbf{x}}_k$, as illustrated in Figure 3. The advantage to using these operational space quantities is that the matrix sizes are constant regardless of the number of degrees of freedom in the chain. Given the current state of the system and the input, both $\ddot{\mathbf{x}}_{k_{open}}$ and $\mathbf{\Lambda}_k$ can be computed with efficient methods described in [33, 37].

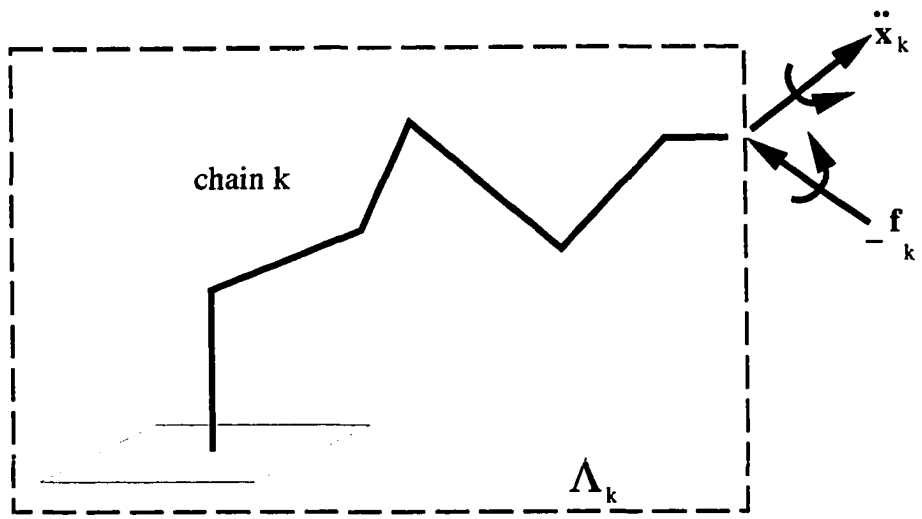


Figure 3: Operational space model for a single chain.

To complete the development, the dynamic equations for the reference member must be defined. Since the end-effector acceleration of each chain is computed at the reference point which is also fixed on the reference member, it is convenient to define the acceleration of the reference member, $\mathbf{a}_r$, at this point as well. As a result, the relationship, $\ddot{\mathbf{x}}_k = \mathbf{a}_r$, holds for all chains and Eq. (2.6) can be written as follows:

$$\mathbf{a}_r = \ddot{\mathbf{x}}_{k_{open}} - \mathbf{\Lambda}_k^{-1}\mathbf{f}_k. \tag{2.7}$$

Using a spatial force balance equation, the sum of the spatial forces exerted by the end-effector of each chain and any other external forces (including gravity) is related to the acceleration of the reference member through its spatial inertia, $\mathbf{I}_r$. This can

be written in equation form for an arbitrary reference point as follows:

$$\sum_{k=1}^{m} \mathbf{f}_k + \mathbf{f}_r^g = \mathbf{I}_r \mathbf{a}_r + \bar{\mathbf{f}}_r. \tag{2.8}$$

The spatial force due to gravity exerted on the reference member is specified by $\mathbf{f}_r^g$, and $\bar{\mathbf{f}}_r$, is a spatial vector specifying the velocity-dependent bias forces of the reference member:

$$\bar{\mathbf{f}}_r = \begin{bmatrix} \omega_r \times \bar{I}_r \omega_r \\ \omega_r \times (\omega_r \times h_r) \end{bmatrix}, \tag{2.9}$$

where ω_r is the angular velocity, $\bar{I}_r$ is the moment of inertia, and h_r is the first moment of mass of the reference member as defined in Chapter 1.

For the purposes of this and the next chapter, we assume that $N \geq 6$ and the manipulators are not at singular configurations. Consequently, all of the Λ_k^{-1} matrices are nonsingular (see Chapter 4 for modifications to this procedure that account for singular configurations). In this case, Eq. (2.7) may be solved for the contact force, $\mathbf{f}_k$, of each chain and substituted into Eq. (2.8). After collecting terms, the following equation results:

$$\left[\mathbf{I}_r + \sum_{k=1}^{m} \Lambda_k\right] \mathbf{a}_r = \left[\mathbf{f}_r^g - \bar{\mathbf{f}}_r + \sum_{k=1}^{m} \Lambda_k \ddot{\mathbf{x}}_{k_{open}}\right]. \tag{2.10}$$

This equation states that the sum of all the inertias of the system multiplied by the reference member acceleration is equal to the sum of all the forces that are exerted on the reference member. This acceleration may now be determined from this linear system of equations. Finally $\mathbf{a}_r$ is substituted back into Eq. (2.7) to determine the tip forces, $\mathbf{f}_k$:

$$\mathbf{f}_k = \Lambda_k \left(\ddot{\mathbf{x}}_{k_{open}} - \mathbf{a}_r\right). \tag{2.11}$$

The tip forces may then be used in Eq. (2.2) to determine the joint accelerations.

2.3 An Efficient Algorithm for Rigid Grasps

In the application of interest, it is assumed that four PUMA manipulators are rigidly grasping the reference member. Using this assumption, the algorithm can be modified to increase its efficiency. Instead of using a single reference point, r, with respect to which all chains' operational space quantities ($\ddot{\mathbf{x}}_{k_{open}}$, $\mathbf{f}_k$, and $\mathbf{\Lambda}_k$) are computed, each chain computes its quantities in its last link's coordinate system. This will be referred to as the kth coordinate system, subsequently. One advantage of this approach is that most efficient single chain robotics algorithms have already been developed by others to compute these quantities and can then be directly applied to this problem. In addition, these coordinate systems are fixed with respect to the reference point, and the cost of transforming between them is much less than simulating the augmented chains with the extra link each.

Because the operational space quantities of each chain and the reference member are now computed with respect to different coordinate system, some of the equations in the previous derivation require modification. First, Eq. (2.7) must be rewritten to express the reference member acceleration at the kth coordinate system:

$$^k\mathbf{a}_r = \ddot{\mathbf{x}}_{k_{open}} - \mathbf{\Lambda}_k^{-1}\mathbf{f}_k \qquad \text{for } k = 1, \ldots, m. \tag{2.12}$$

where the standard spatial transformation for accelerations at different points on a rigid body is used to compute this acceleration:

$$^k\mathbf{a}_r = {}^k\mathbf{X}_r\mathbf{a}_r + \boldsymbol{\zeta}_k, \tag{2.13}$$

and ζ_k contains the velocity-dependent acceleration term as follows:

$$\zeta_k = \begin{bmatrix} \mathbf{0} \\ {}^k\boldsymbol{R}_r \left[\boldsymbol{\omega}_r \times (\boldsymbol{\omega}_r \times {}^r\boldsymbol{p}_k)\right] \end{bmatrix}, \tag{2.14}$$

where $\boldsymbol{\omega}_r$ is the angular velocity of the reference member, and ${}^k\boldsymbol{R}_r$ and ${}^r\boldsymbol{p}_k$ define the orientation and position between the two coordinate systems. Note that a variable without a leading superscript implies that components are expressed in the coordinate system indicated by the trailing subscript. Therefore $\mathbf{a}_r$ is expressed in the reference member coordinate system while ${}^k\mathbf{a}_r$ is expressed in the kth coordinate system.

Second, it is also desirable to include a gravitational acceleration term on both sides of the resulting equation:

$${}^k\mathbf{X}_r \left(\mathbf{a}_r - {}^r\mathbf{a}_g\right) + \zeta_k = \ddot{\mathbf{x}}_{k_{open}} - {}^k\mathbf{a}_g - \boldsymbol{\Lambda}_k^{-1}\mathbf{f}_k, \tag{2.15}$$

where ${}^r\mathbf{a}_g$ and ${}^k\mathbf{a}_g$ are the same acceleration expressed at the reference and kth coordinate systems, respectively. Finally, Eq. (2.15) is solved for the tip forces as follows:

$$\mathbf{f}_k = \boldsymbol{\Lambda}_k \left(\ddot{\mathbf{x}}'_{k_{open}} - {}^k\mathbf{X}_r \mathbf{a}'_r - \zeta_k\right), \tag{2.16}$$

where

$$\ddot{\mathbf{x}}'_{k_{open}} = \ddot{\mathbf{x}}_{k_{open}} - {}^k\mathbf{a}_g, \quad \text{and} \tag{2.17}$$

$$\mathbf{a}'_r = \mathbf{a}_r - {}^r\mathbf{a}_g. \tag{2.18}$$

The reason for including these gravitational acceleration terms is a consequence of the method by which the open-chain accelerations will be computed, and is explained in the next section.

Finally, the force balance on the reference member in Eq. (2.8) is modified. The tip forces must be transformed to the reference point, and the gravitational force vector, $\mathbf{f}_r^g$, is replaced with the equivalent expression in terms of the gravitational acceleration. The resulting equation is written as follows:

$$\sum_{k=1}^{m} {}^k\mathbf{X}_r^T \mathbf{f}_k = \mathbf{I}_r \left(\mathbf{a}_r - {}^r\mathbf{a}_g\right) + \bar{\mathbf{f}}_r, \tag{2.19}$$

$$= \mathbf{I}_r \mathbf{a}'_r + \bar{\mathbf{f}}_r. \tag{2.20}$$

After substituting from Eq. (2.16) and collecting terms, the following system of equations results:

$$\left[\mathbf{I}_r + \sum_{k=1}^{m} \left({}^k\mathbf{X}_r^T \mathbf{\Lambda}_k {}^k\mathbf{X}_r\right)\right] \mathbf{a}'_r = \left[-\bar{\mathbf{f}}_r + \sum_{k=1}^{m} {}^k\mathbf{X}_r^T \mathbf{\Lambda}_k \left(\ddot{\mathbf{x}}'_{k_{open}} - \boldsymbol{\zeta}_k\right)\right], \tag{2.21}$$

which can be solved to obtain $\mathbf{a}'_r$ and from that the reference member acceleration. Using this derivation, the dynamics algorithm is summarized in Table 1.

This approach is an extension of the efficient way that gravitational forces are incorporated into the inverse dynamics computation. By biasing the acceleration with negative gravitational acceleration, the corresponding force is computed along with the bias forces when the biased acceleration is multiplied by the inertia. Physically, this is equivalent to the statement that the force exerted on a stationary body in a gravitational field is equivalent to the one felt if the body is accelerating with the negative of gravitational acceleration in zero-g. This is carried over into the calculation of the reference member acceleration, in Eq. (2.21), where the biased reference member acceleration times its inertia provides the missing gravitational force from the force balance equation. Due to more complex equations, however, unwanted

Table 1: Multiple closed-chain dynamics algorithm.

Step 1: Open Chain Dynamics.

for all $k = 1, \ldots, m$:

Compute $\mathbf{H}_k^{-1}\mathbf{J}_k^T$, $\mathbf{\Lambda}_k$, $\ddot{\mathbf{q}}_{k_{open}}$, $\ddot{\mathbf{x}}'_{k_{open}}$, ${}^k\mathbf{X}_r^T\mathbf{\Lambda}_k{}^k\mathbf{X}_r$, and ${}^k\mathbf{X}_r^T\mathbf{\Lambda}_k\left(\ddot{\mathbf{x}}'_{k_{open}} - \boldsymbol{\zeta}_k\right)$.

Step 2: Reference Member Acceleration.

Solve the following for $\mathbf{a}'_r$:

$$\left[\mathbf{I}_r + \sum_{k=1}^{m}\left({}^k\mathbf{X}_r^T\mathbf{\Lambda}_k{}^k\mathbf{X}_r\right)\right]\mathbf{a}'_r = \left[-\bar{\mathbf{f}}_r + \sum_{k=1}^{m}{}^k\mathbf{X}_r^T\mathbf{\Lambda}_k\left(\ddot{\mathbf{x}}'_{k_{open}} - \boldsymbol{\zeta}_k\right)\right],$$

$$\mathbf{a}_r = \mathbf{a}'_r - {}^r\mathbf{a}_g.$$

Step 3: Closed Chain Dynamics.

for all $k = 1, \ldots, m$:

$$\mathbf{f}_k = \mathbf{\Lambda}_k\left[\left(\ddot{\mathbf{x}}'_{k_{open}} - \boldsymbol{\zeta}_k\right) - {}^k\mathbf{X}_r\mathbf{a}'_r\right],$$

$$\ddot{\mathbf{q}}_k = \ddot{\mathbf{q}}_{k_{open}} - \left(\mathbf{H}_k^{-1}\mathbf{J}_k^T\right)\mathbf{f}_k.$$

---

terms are generated by the multiplication with the chains operational space inertias. These are negated by corresponding terms generated by using a biased open chain tip acceleration on the right hand side.

2.4 Computational Requirements

In this section, the computational requirements for an efficient implementation of each step of the algorithm in Table 1 is presented. To simplify the presentation, multiplication and division operations are combined and labeled as multiplies ($\times$),

and subtractions are combined with additions (+). The computational requirements of the resulting implementation are summarized in Table 2 at the end of this section.

The Open Chain Dynamics (OCD) step is the most computationally expensive step. Details of its implementation have thus far been ignored in this chapter since efficient single, open-chain algorithms to complete this step have been developed by others. Lilly and Orin developed the Modified Composite-Rigid-Body Method (Method III) in [25] which computes $\mathbf{H}_k$ and $\mathbf{J}_k$ simultaneously with a computational cost of $[(12\frac{1}{2}N^2 + 24\frac{1}{2}N - 37)\text{M}, (8N^2 + 40N - 48)\text{A}]^*$ per chain. This algorithm has been improved by replacing its procedure for transforming composite rigid body inertias with the one in Appendix A. For a chain with revolute joints only the computational cost drops to $[(12\frac{1}{2}N^2 + 5\frac{1}{2}N - 18)\text{M}, (8N^2 + 30N - 38)\text{A}]$. Then, $\mathbf{H}_k^{-1}\mathbf{J}_k^T$ (also called $\mathbf{\Omega}_k$ in [25]), $\mathbf{\Lambda}_k^{-1}$, and $\mathbf{\Lambda}_k$ can be determined using the procedure described in Lilly and Orin's Unit Force Method (Method II) of [37].

The inverse dynamics algorithm from Appendix B.2 is used in the computation of $\ddot{\mathbf{q}}_{k_{open}}$ and $\ddot{\mathbf{x}}_{k_{open}}$. By setting the joint accelerations and tip forces to zero, the output of the inverse dynamics computation, also called the bias computation [14] under these conditions, consists of the following terms from Eq. (2.1):

$$\mathbf{b}_k = \mathbf{C}_k(\mathbf{q}_k, \dot{\mathbf{q}}_k) + \mathbf{G}_k(\mathbf{q}_k). \tag{2.22}$$

The computational cost is $[(94N - 114)\text{M}, (81N - 105)\text{A}]$ per chain. With the input joint torques known, the open-chain joint accelerations are computed as follows:

$$\ddot{\mathbf{q}}_{k_{open}} = \mathbf{H}_k^{-1}(\boldsymbol{\tau}_k - \mathbf{b}_k). \tag{2.23}$$

*$[(x)\text{M}, (y)\text{A}] = x$ multiplications, y additions.

Since the Cholesky decomposition of $\mathbf{H}_k$ has already been accomplished in the previous step, only a vector subtraction and back-substitution is required to complete this step at a cost of [(N^2)M, (N^2)A] per chain.

A by-product of the inverse dynamics computation is the computation of the acceleration of each link. With the joint accelerations set to zero and the fixed base's acceleration set to negative gravitational acceleration, $-\mathbf{a}_g$, it can be shown that the acceleration computed for the last link of chain k is:

$$\mathbf{a}_{k,N} = \dot{\mathbf{J}}_k \dot{\mathbf{q}}_k - {}^k\mathbf{a}_g. \tag{2.24}$$

The desired open-chain quantity, $\ddot{\mathbf{x}}'_{k_{open}}$, can then be computed [from Eqs. (2.4) and (2.17)] as follows:

$$\ddot{\mathbf{x}}'_{k_{open}} = \mathbf{J}_k \ddot{\mathbf{q}}_{k_{open}} + \mathbf{a}_{k,N}, \tag{2.25}$$

at a cost of [$(6N)$M, $(6N)$A] per chain.

The final part of the OCD step involves the spatial transformation of operational space quantities to the reference point. With careful kinematic modeling, the transformation between the last link of each chain and the reference point can be made especially efficient. This is accomplished with the coordinate system assignment shown Figure 4. By placing the x-axis of the last link along the common normal of joint N and the reference member's z-axis, the transformation between these coordinate systems is efficiently specified by another set of modified Denavit-Hartenberg (MDH) parameters. Using these four parameters, the spatial transformation, ${}^r\mathbf{X}_k$, is defined as in Appendix A. Another method of defining this transformation is by

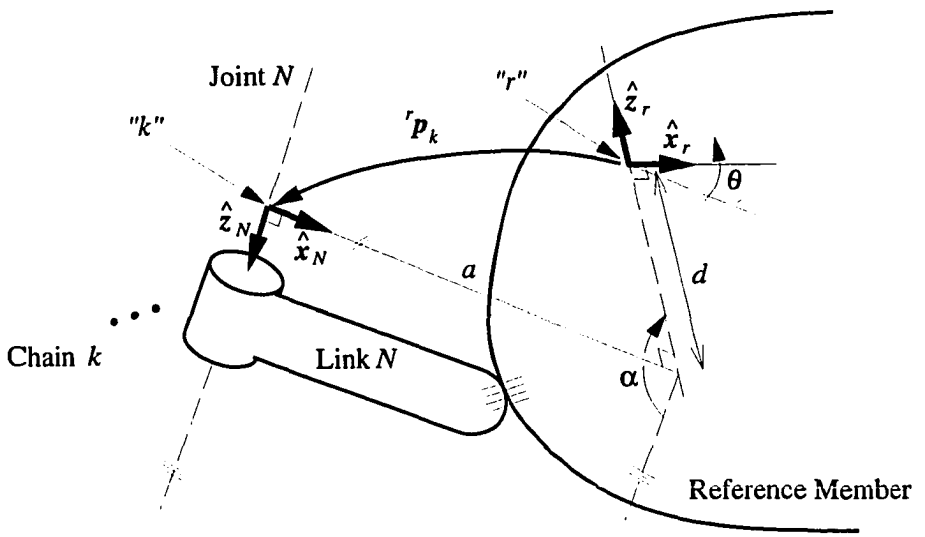


Figure 4: Efficient coordinate system assignment for each chain.

two successive planar screws about the x and z axes [27]:

$$^r\mathbf{X}_k \;=\; \mathbf{X}_z(d,\theta)\,\mathbf{X}_x(a,\alpha). \tag{2.26}$$

However, the transformation that is actually needed in the previous development is $^k\mathbf{X}_r$ which is the inverse of $^r\mathbf{X}_k$. This matrix can be still written as two planar screws as follows:

$$^k\mathbf{X}_r \;=\; \mathbf{X}_x(-a,-\alpha)\,\mathbf{X}_z(-d,-\theta). \tag{2.27}$$

Therefore the efficiency of the transformations using MDH parameters is also achieved with this. With results from Appendix A, the congruence transformation of $\mathbf{\Lambda}_k$, a symmetric 6×6 matrix, requires [84M, 86A]. The computation of the spatial vector, $\mathbf{\Lambda}_k\left(\ddot{\mathbf{x}}'_{k_{open}} - \boldsymbol{\zeta}_k\right)$ and its spatial transformation requires [76M, 55A].

In the Reference Member Acceleration (RMA) step of the computation in Table 1, the computation of $\mathbf{a}'_r$ requires the computation of $\bar{\mathbf{f}}_r$, the summation of seven spatial vectors, the summation of m symmetric 6×6 matrices and a spatial inertia matrix, and the solution of a 6×6 system of equations. By placing the reference member coordinate axes along its principal axes, $\boldsymbol{h}_r$ is zero and $\bar{\boldsymbol{I}}_r$ is diagonal. Therefore, $\bar{\mathbf{f}}_r$ contains three zeros and only requires [9M, 3A] to be computed. The summations then require a total of $27m - 9$ additions. Since the resulting matrix is symmetric and positive definite, root-free Cholesky decomposition and back-substitution is the most efficient procedure to compute $\mathbf{a}'_r$ at a cost of [86M, 65A] [14, 24].

To complete the RMA step, $\mathbf{a}_r$, must be computed. To accomplish this, the gravitational acceleration must be transformed to the reference coordinate system and subtracted from $\mathbf{a}'_r$. If the z-axis of the inertial coordinate system is placed along the direction of gravitational acceleration, and Euler angles are used to specify the orientation of the reference member with respect to this frame, the transformation requires [4M, 0A] and the subtraction requires [0M, 3A] since the angular acceleration portion of the gravitational acceleration spatial vector is always zero.

The computation requirements for the Closed Chain Dynamics (CCD) step is relatively straightforward. To compute the constraint forces at the tip requires a spatial transformation [20M, 12A], spatial vector addition [0M, 6A], and spatial matrix-vector multiply [36M, 30A]. The joint accelerations of each chain is obtained from a $N \times 6$ matrix-vector multiply [$(6N)$M, $(5N)$A] and an N-vector addition. The totals at the bottom of Table 2, show that this algorithm has a computational

Table 2: Computational requirements for the closed-chain dynamic simulation algorithm.

Computation	$\times$	$+$
$\mathbf{H}_k,\ \mathbf{J}_k$	$m\left(12\frac{1}{2}N^2+5\frac{1}{2}N-18\right)$	$m\left(8N^2+30N-38\right)$
$\mathbf{H}_k^{-1}\mathbf{J}_k^T$	$m\left(\frac{1}{6}N^3+6\frac{1}{2}N^2-\frac{2}{3}N\right)$	$m\left(\frac{1}{6}N^3+6N^2-6\frac{1}{6}N\right)$
$\mathbf{\Lambda}_k^{-1}$	$21mN$	$m(21N-21)$
$\mathbf{\Lambda}_k$	$141m$	$90m$
$\ddot{\mathbf{q}}_{k_{open}},\ \ddot{\mathbf{x}}'_{k_{open}}$	$m(N^2+100N-114)$	$m(N^2+87N-105)$
${}^k\mathbf{X}_r^T\mathbf{\Lambda}_k{}^k\mathbf{X}_r$	$84m$	$86m$
${}^k\mathbf{X}_r^T\mathbf{\Lambda}_k\left(\ddot{\mathbf{x}}'_{k_{open}}-\boldsymbol{\zeta}_k\right)$	$76m$	$55m$
$\mathbf{a}'_r,\ \mathbf{a}_r$	99	$27m+62$
$\mathbf{f}_k$	$56m$	$48m$
$\ddot{\mathbf{q}}_k$	$6mN$	$6mN$
Total:	$m(\frac{1}{6}N^3+20N^2+131\frac{5}{6}N+225)+99$	$m(\frac{1}{6}N^3+15N^2+137\frac{5}{6}N+142)+62$
$m=4, N=6$	7187	6242

complexity of $O(mN^3)$, and for the four PUMA system [7187M, 6242A] are required to compute the desired accelerations.

2.5 Summary and Conclusions

In this chapter, the dynamics of a multiple manipulator system grasping a common object has been presented. For each chain, at least six degrees of freedom, non-singular configurations, and rigid grasps with the reference member were assumed, and an efficient $O(mN^3)$ algorithm for the computation of the joint accelerations in

each chain and the acceleration of the reference member was developed. The most important characteristic of this development is its dependence on the more common serial chain algorithms wherever possible. One reason is that a significant amount of work has been completed by others in this area. Based on many of their results, the best algorithms have been developed in Appendix B and are heavily relied on in the development in this Chapter. Another reason is that it allows a higher level approach to the development of the algorithm.

For a system consisting of four PUMA 560 manipulators ($m = 4$, $N = 6$), this algorithm requires [7187M, 6242A] to compute the desired accelerations. By comparison, Lilly and Orin's O(mN) algorithm in [24] has a computational cost of [$(656mN - 171m + 128)$M, $(546mN - 208m + 101)$A]. For the four-PUMA system under consideration this method requires [15188M, 12589A]. The new algorithm represents a savings of over 50%, and is more efficient for $N \leq 21$. Although Lilly and Orin's algorithm considers the case of arbitrary grasps on the reference member, most of the savings for the algorithm presented in this chapter is achieved in the computation of the open-chain quantities which are unaffected by the grasp type.

This chapter has also provided the background needed in the next two chapters. While the algorithm developed in this chapter is very efficient, Chapter 3 examines methods of parallelizing this algorithm to further increase the computation rates so that real-time simulation of this system can be achieved. This algorithm also does not address the case when a manipulator becomes singular. This problem will be specifically addressed in Chapter 4.

CHAPTER III

Parallel Simulation: Spatial and Temporal Methods

3.1 Introduction

In the previous chapter, an efficient serial algorithm was developed to compute the dynamics of multiple manipulators that are cooperating to manipulate large loads. This algorithm, which determines accelerations, must be combined with a numerical integration routine in order to simulate the system. Because numerical integration is a discrete approximation of a continuous process, the dynamics computation must be performed hundreds or even thousands of times each second to achieve accurate results. On all but the fastest supercomputers, this computation, today, cannot be performed in real-time. One approach that has been taken to improve computational rates is to parallelize the dynamics computations. With significant speedups over serial implementation, real-time performance can be achieved and hardware-in-the-loop and human-in-the-loop applications can be implemented.

To achieve these rates, fine-grain parallel algorithms have been developed in most previous work. Some have required the use of special-purpose architectures to

Results from this chapter are to be published in "Parallel Dynamic Simulation of Multiple Manipulator Systems: Temporal vs. Spatial Methods," To appear in *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 24, no. 8, August 1994 (with P. Sadayappan and D. E. Orin).

implement the fine-grain parallelism of the computations required for a single-chain system [18, 19, 20, 21, 38, 39]. Others decompose the computation into slightly larger blocks of concurrent tasks and schedule them on a number of tightly coupled parallel processors to produce speedup [22, 23]. These approaches, while designed to perform well on specific parallel architectures, are much less efficient when implemented on the more widely available general-purpose parallel and desktop multiprocessor systems. These systems typically have a limited number of loosely coupled processors that are not designed to efficiently handle the large amount of interprocessor communication and synchronization required by the fine-grain tasks.

In contrast, the work in this chapter focuses on the development of parallel algorithms which run efficiently on existing general-purpose parallel computers, such as hypercubes, or multiprocessor workstations that are now available from SGI, Sun, DEC, and HP. This approach has distinct advantages over the fine-grained approaches because less time is required to develop the algorithms and no hardware development must be performed. In addition, the general-purpose architectures allow for greater flexibility in the approaches to parallelism to be investigated.

In the first stage of this research, the dynamic simulation algorithm for a single, open-chain robotic manipulator was effectively parallelized and the results were reported in [36]. The approach used a parallel block predictor-corrector (BPC) integration method to achieve *temporal* parallelism in which the dynamics of the system was computed for multiple points in time simultaneously. The purpose of this chapter is to extend this work to the simulation of multiple closed-chain systems and,

in particular, multiple manipulator systems that are cooperating to manipulate a large load called the reference member. In this case, a second form of parallelism can be exploited which allows the dynamics of the individual chains to be computed in parallel. This form of parallelism corresponds to the system's structural parallelism and is called *spatial* parallelism. Both forms of parallelism can be combined by implementing this spatial parallelism *within* each temporal parallel task. In this case, the number of parallel tasks possible in the dynamic simulation is the product of the two.

In the next section, the computational structure of the dynamics algorithm for the multiple, closed-chain algorithm from Chapter 2 is briefly reviewed, and its spatial parallelism is discussed. In Section 3.3, numerical integration which is required to perform simulations is introduced. An improved form of the BPC integration method from [36], called the Sliding BPC (SBPC) method, is developed and its increased flexibility will allow varying amounts of temporal parallelism in the simulation algorithm while maximizing accuracy. Section 3.4 discusses how both forms of parallelism are combined and implemented on a general-purpose parallel computer and provides an analysis of the interprocessor communication and serial processing overhead required in the resulting algorithm. This algorithm is then implemented on a CRAY Y-MP8/864, and the speedup results are given in Section 3.5 for various parallel configurations of the simulation algorithm including temporal parallelism only, spatial parallelism only, and various combinations thereof.

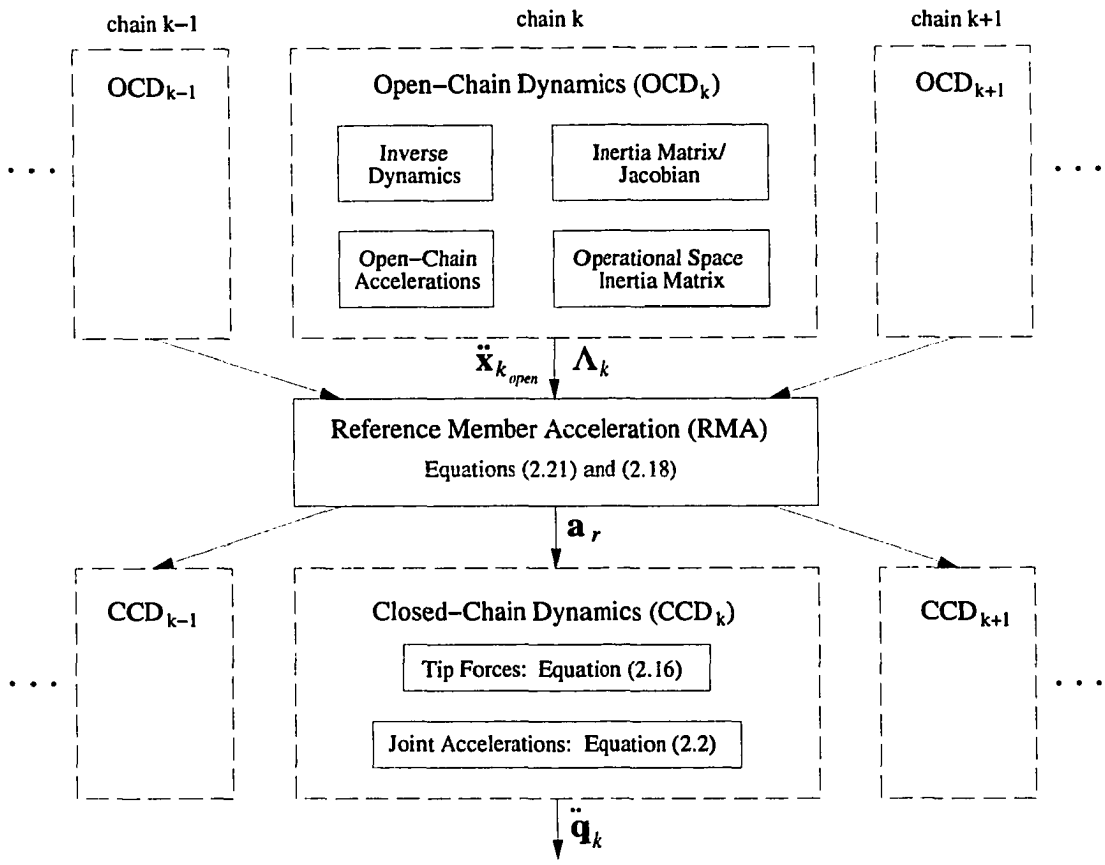


Figure 5: Multiple manipulator dynamics algorithm.

3.2 Spatial Parallelism

The dynamics algorithm to compute the accelerations, the derivatives of state variables, in Table 1 is outlined as a flowchart in Figure 5. Emphasis has been placed on the fact that the computations within the Open-Chain Dynamics (OCD) and Closed-Chain Dynamics (CCD) steps for each chain in the system does not depend on the computations for any other chain in the system. Therefore, these computations can

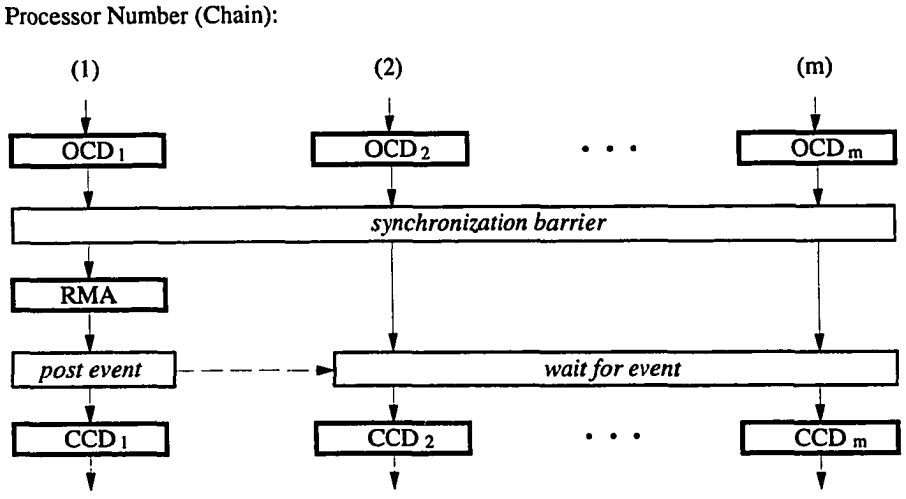


Figure 6: Spatial parallelism in multiple-chain dynamic equations (serial RMA computation).

be performed in simultaneously which leads directly to the *spatial* parallelism in the computation. In the middle of this spatial parallelism, the Reference Member Acceleration (RMA) step is performed once for a given derivative evaluation, and represents serial computation that must be performed.

Figure 6 shows how this algorithm could be implemented on a general-purpose parallel computer for an m-chain system. Processor synchronization is required after the parallel OCD computation in order to collect the chains' operational space inertia matrices and open-chain accelerations before the serial RMA computation can be performed. In Figure 6 this is indicated by the *barrier* whose purpose is to hold the parallel tasks which have completed the OCD section until all of them have finished. After the serial computation is completed by one processor, it signals the other processors, via an *event*, to continue with the parallel CCD computations.

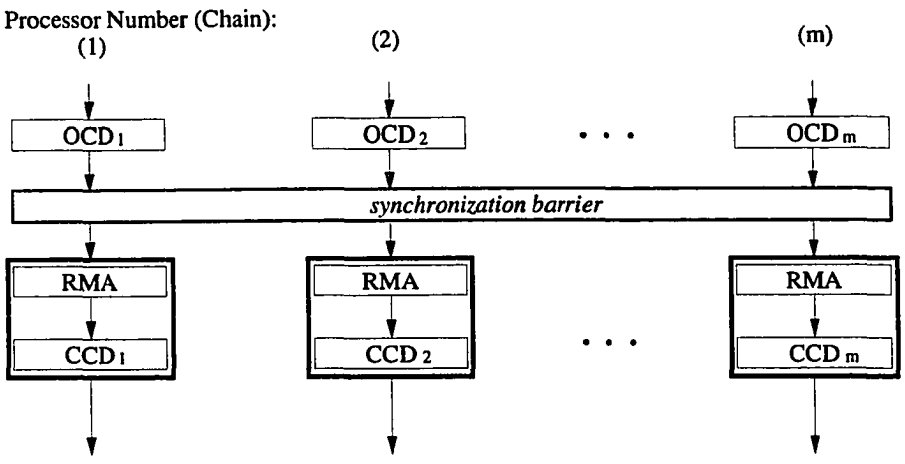


Figure 7: Spatial parallelism in multiple-chain dynamic equations (redundant parallel RMA computation).

This computation is modified in Figure 7 to remove the event synchronization by allowing each task to maintain its own "copy" of the reference member information and perform its own RMA computation. While this introduces redundant computation, the execution (wallclock) time will not increase since it is performed during the time that the processors were waiting on one to complete the RMA computation. It can even reduce the execution time because a costly interprocessor synchronization point has been removed.

3.3 Numerical Integration and Temporal Parallelism

Up to this point, the dynamics algorithm, which computes the accelerations in a robotic system given the state of the system and the joint inputs, has been examined. This is only part of a simulation algorithm which couples a numerical

integration routine with this dynamics computation to update the state information for the next point in time. In this context, the dynamics constitutes the "derivative computation" which uses the state information, positions and velocities, provided by a numerical integration routine and the input torques and forces. In turn, the dynamics provides the numerical integration routine with a new state derivative, velocities and accelerations, which is integrated to obtain the updated state.

Many different methods are available to perform this numerical integration, and the most popular are predictor-corrector and Runge-Kutta methods. In this section, temporal parallelism is investigated which is more readily achieved with the class of predictor-corrector methods called PECE. To describe this form of parallelism, a common serial version of the PECE method is reviewed. Then, a new approach for predicting and correcting the states in parallel, called the Sliding Block Method, is presented.

3.3.1 Serial PECE Method

The PECE methods can be divided into two steps regardless of the order of the method (unlike Runge-Kutta methods). In the first step, the state at the next desired point in time is *P*redicted based on past values and the derivative is *E*valuated based on this prediction. Then, this state is *C*orrected followed by another derivative *E*valuation. In this chapter, fifth-order methods are used to provide an adequate tradeoff between accuracy, which tends to increase with order, and numerical stability, which tends to decrease with order. Figure 8 shows the quantities needed and computed in both steps of the serial method.

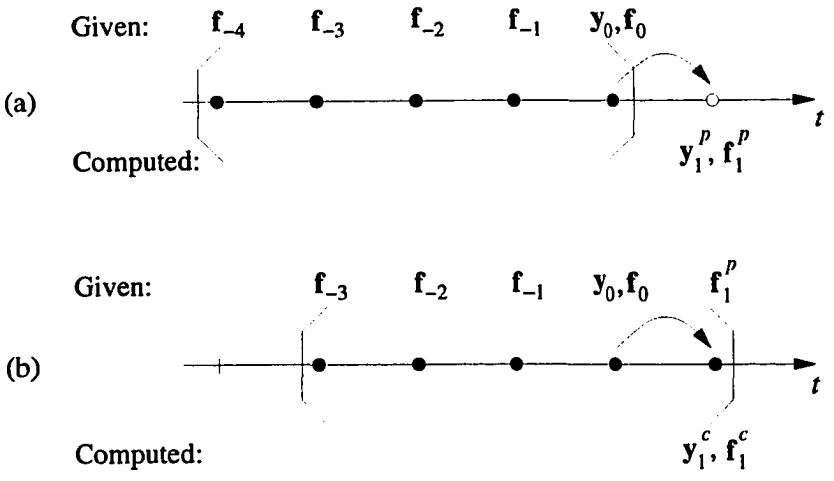


Figure 8: Serial B1PC5 integration: (a) predictor step, (b) corrector step.

The predictor step is shown in Figure 8(a). It computes a linear combination of five (for fifth-order) past derivative vectors, $\mathbf{f}_{-4}, \ldots, \mathbf{f}_0$, and the state at the most recent point in time, $\mathbf{y}_0$ to predict the state of the system at the next point in time, $\mathbf{y}_1^p$. This is used by the dynamics algorithm to compute the derivative vector (joint and reference member accelerations), $\mathbf{f}_1^p$, at this point in time. Note that in this discussion regarding numerical integration, the derivative of state vector, $\mathbf{f}$, should not be confused with a spatial force vector which is used in the discussions of the dynamics algorithms.

To correct the state in the second step of the method, the quantities shown in Figure 8(b) are used. The "oldest" derivative vector is dropped and the linear combination used to compute the corrected state, $\mathbf{y}_1^c$, now includes the predicted derivative vector. The same state vector at time step zero, $\mathbf{y}_0$, is used rather than the predicted one at time step one, $\mathbf{y}_1^p$, for reasons of stability and accuracy. However,

the corrected derivative vector is then computed using $\mathbf{y}_1^c$.

In the next iteration of the method, the values are shifted and four old derivative vectors, $\mathbf{f}_{-3}, \ldots, \mathbf{f}_0$ and the new corrected derivative, $\mathbf{f}_1^c$, become the five derivative vectors used in the next predictor step along with the newest corrected state vector, $\mathbf{y}_1^c$. The block of points to be used has "slid" forward by one step in time while maintaining the same number of points required to achieve the desired order of the method. This method is usually called PECE5, but will be referred to as B1PC5 in this chapter to indicate a one-point Sliding-Block Method which utilizes fifth-order predictor-corrector formulas to contrast it from the multi-point block methods presented next.

3.3.2 Parallel Sliding-Block Methods

Temporal parallelism can be achieved by performing the computations described above for more than one new point in time. This concept is illustrated in Figure 9 for a two-point method called B2PC5 which can be implemented on two processors in parallel. Each processor has access to the past state and derivative vector information, and with an appropriate linear combination (as determined by the method of undetermined coefficients [40]), computes the predicted state for one of the two points in time. Each processor can then perform the dynamics computations for the system independently to determine the derivative vectors, $\mathbf{f}_1^p$ and $\mathbf{f}_2^p$.

Before the corrector step can take place, however, the processors need to exchange the new derivative information because a linear combination including both of these values is needed to compute the corrected state as shown in Figure 9(b).

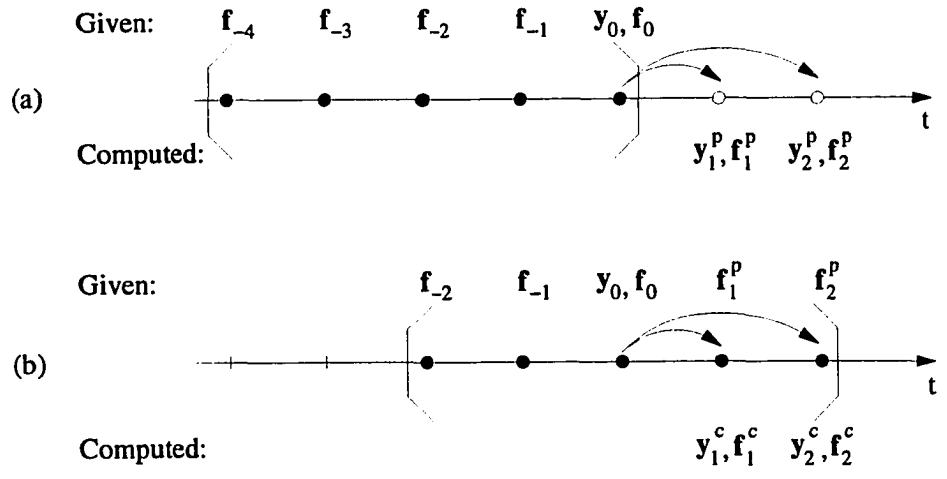


Figure 9: Two-point parallel B2PC5 integration: (a) predictor step, (b) corrector step.

Note that in this step, *two* old values have now been discarded in favor of the new predicted information to maintain fifth-order accuracy, and it is the corrected information at these five points that will be used in the next iteration of the method.

Any amount of temporal parallelism can be achieved by increasing the number of new points to be computed. For reasons of accuracy and stability which decrease with increasing points, the limit for the number of temporal parallel tasks is chosen as one less than the order of the method. For the fifth-order method considered in this work, this limit corresponds to the four-point B4PC5 method shown in Figure 10. This method is also equivalent to the the parallel block predictor-corrector (BPC4) method formulated by Birta and Abou-Rabia [41] and used in [36]. In their approach, however, the number of parallel tasks was always one less than the order of the method to be used so no freedom was given between choosing the desired order

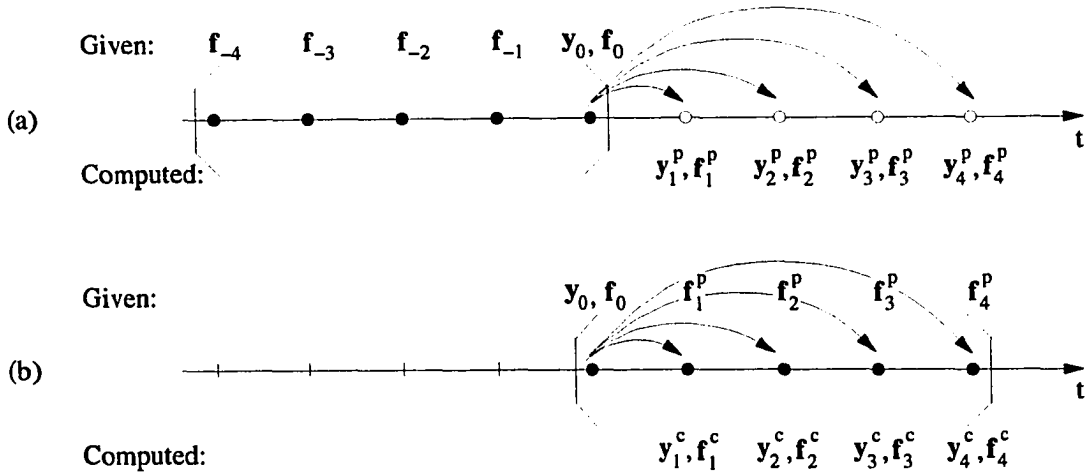


Figure 10: Four-point parallel B4PC5 integration: (a) predictor step, (b) corrector step.

of the method *and* the amount of parallelism. The sliding block method presented in this section overcomes this drawback.

The structure of the temporal parallelism for a single iteration of the methods presented in this section is shown in Figure 11. The PE (compute *P*redicted state and *E*valuate the derivative) and CE (compute *C*orrected state and *E*valuate the derivative) blocks correspond to the predictor and corrector steps in these methods along with the derivative evaluations described in the previous section. There are p parallel tasks which correspond to the number of points computed by the method during a single iteration, and a single column of blocks represents the computation of the solution of the entire m-chain system for a specific point in time on one of p processors. Barriers are used to synchronize the processors between the predictor and corrector steps so that the state and derivative information may be exchanged.

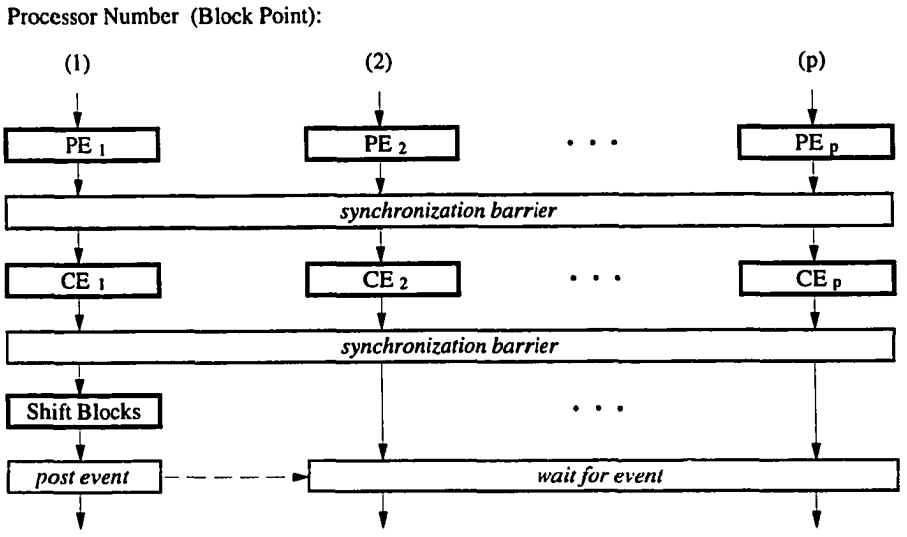


Figure 11: Temporal parallelism in dynamic simulation.

An event is also used so that a small amount of serial processing may be performed to shift the derivative and state information in preparation for the next iteration.

3.4 Combined Parallelism and Task Partitioning

Both forms of parallelism, spatial and temporal, can be combined by implementing the spatial parallelization of the derivative evaluations in Figure 7 within the individual PE/CE computational blocks of Figure 11. Note that the predictor and corrector equations can also be spatially parallelized so that a step to compute the state (either predicted or corrected) of the appropriate chain in the system can be added to each OCD block of Figure 7. A straightforward implementation yields the structure shown in Figure 12.

A drawback to this implementation is the large number of tasks that the event

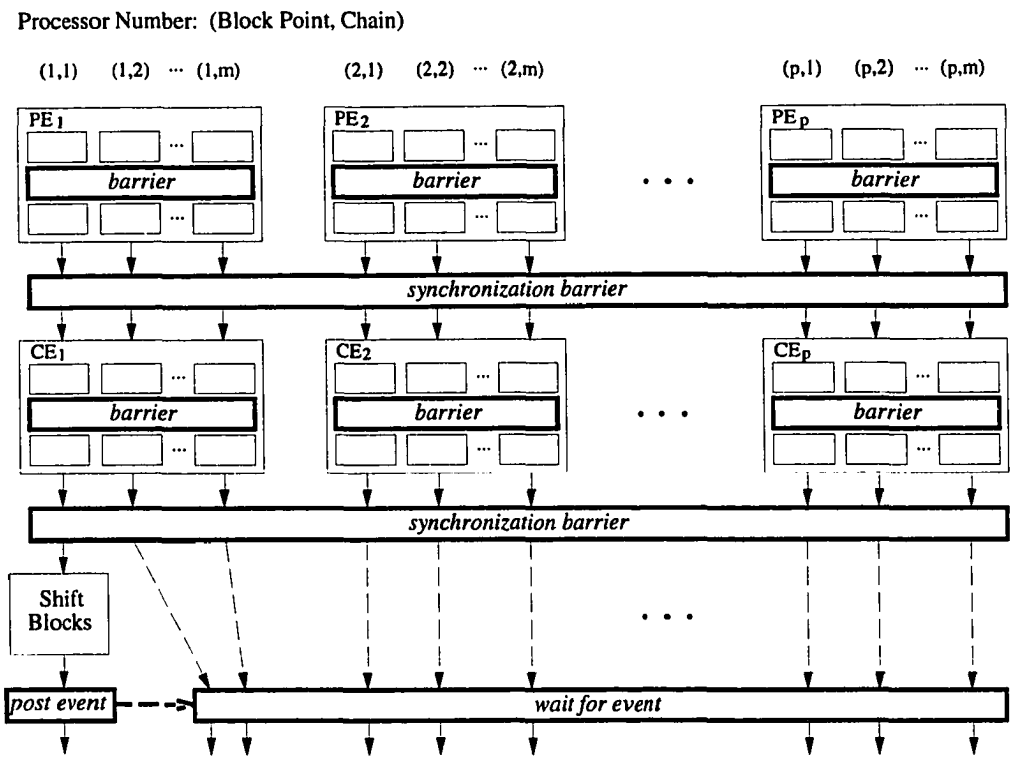


Figure 12: Structure of combined spatial and temporal parallelism.

and barriers are required to synchronize after each step of the integration which can lead to a large amount of overhead in some parallel systems. A modification can be made at these points to relax the synchronization requirements and improve performance. Because the integration of the state of each chain depends only on its own values at all of the block points, these barriers (referred to as temporal barriers) and events are broken apart so that each one only synchronizes with the processors assigned to the same chain. As a result, these temporal synchronization points and the Shift Blocks task have been spatially parallelized. Figure 13 shows the resulting

structure which consists of $m \times p$ parallel tasks.

This figure also shows the synchronization commands that are used in the implementation of this algorithm on the CRAY Y-MP8/864. The simulation code is written in FORTRAN (Version 5.0 running under UNICOS Release 6.0) using macrotasking commands to implement the parallelism. The BARSYNC commands correspond to the synchronization barriers, and the argument given with this command is the number of parallel tasks that it must synchronize. Finally, the EVPOST-EVWAIT commands provide the mechanism by which some processors are suspended while others perform Shift tasks.

Since the CRAY has only eight processors, a robotic system with four chains and the use of the four-way temporal parallelism would result in more parallel tasks than processors. To overcome this problem a *reduced partitioning* mechanism was included in the code that allows the user to specify the number of desired parallel tasks along both the spatial and temporal dimensions of the simulation independently of the number of chains and block points. In order to maintain load balance, however, the specified partition sizes in both dimensions would have to be integer divisors of the total number of chains and block points, respectively. These numbers are denoted by $\tilde{m}$ and $\tilde{p}$, and are used in place of m and p in Figure 13 when referring to the number of actual parallel tasks. In the general case, this mechanism would allow efficient parallel simulation of a robotic system where the product, mp, exceeds the number of physical processors on the target parallel computer.

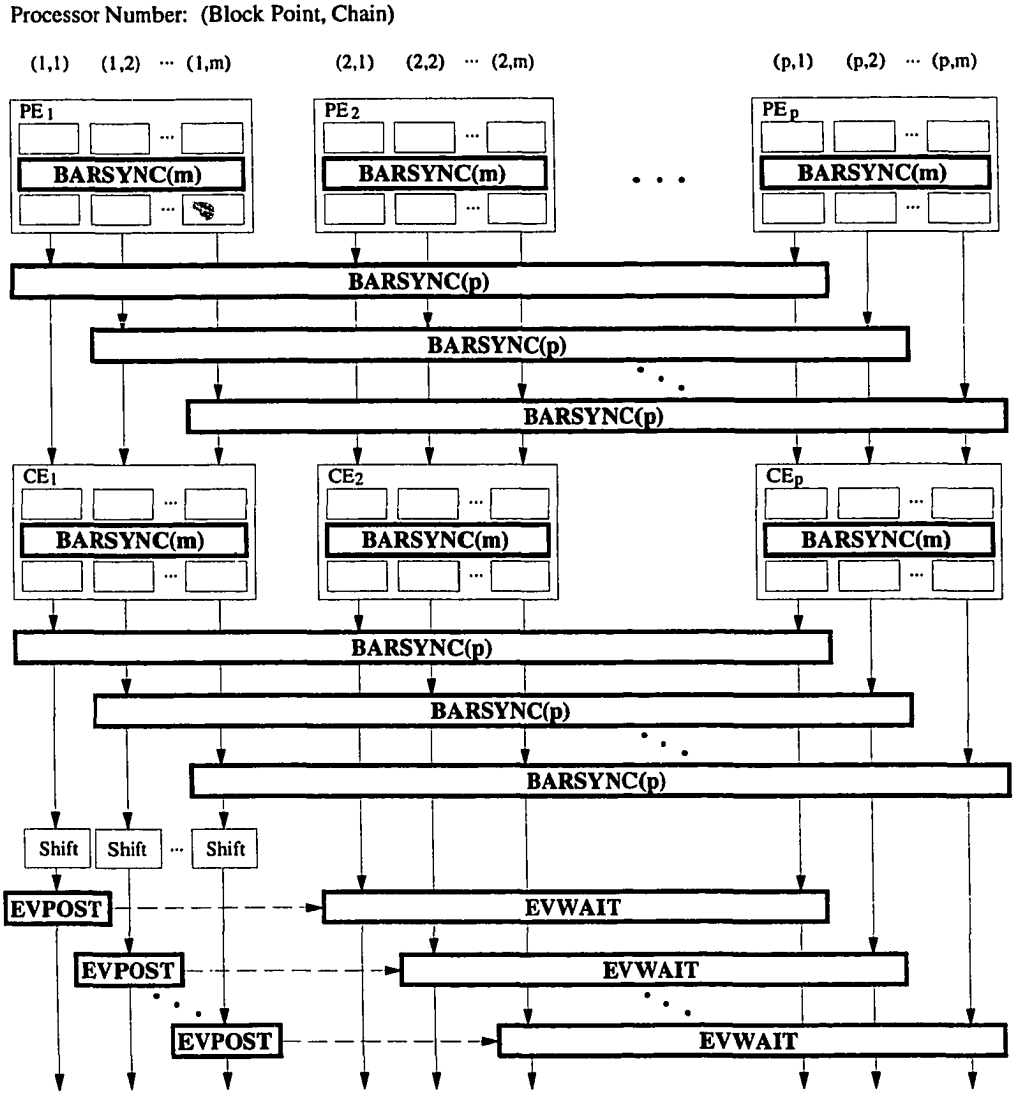


Figure 13: Structure of combined parallelism with reduced synchronization.

Table 3: Number of synchronizations and RMA computations per processor for various partitions.

# CPU	Spatial Tasks, $\tilde{m}$	Temporal Tasks, $\tilde{p}$	Block Size	1 iteration: # Sync.	1 iteration: # RMA	4 points: # Sync.	4 points: # RMA
4	1	4	4	3	2	3	2
	2	2	4	5	4	5	4
			2	5	2	10	4
	4	1	4	2	8	2	8
			2	2	4	4	8
			1	2	2	8	8
8	2	4	4	5	2	5	2
	4	2	4	5	4	5	4
			2	5	2	10	4

3.5 Results

The partitioning mechanism enables the examination of the effects of various configuration parameters such as the number of chains, spatial tasks, temporal tasks, and integration block size, along with the number of physical processors on the performance of the parallel algorithm. As before, this analysis was carried out for a four manipulator system on both four and eight processor systems. The various ways of partitioning was found to have an effect on the number of RMA computations and interprocessor synchronizations that are performed by a single processor as shown in Table 3.

In this table, the various numbers of spatial ($\tilde{m}$) and temporal ($\tilde{p}$) tasks that lead to load-balanced partitions on four and eight processor systems are listed (note

that $\tilde{m} \times \tilde{p}$ is equal to the number of processors for a load-balanced partition). In addition, all of the possible integration Block Sizes are included for each number of temporal tasks. In the fifth and sixth columns of this table, the RMA computation and synchronization requirements for one pass through the computation shown in Figure 13 are given. Only these two computational requirements are listed since all others (e.g., chain dynamics, predictor and corrector equations) remain unchanged. Since the different Block Sizes lead to the computation of different numbers of trajectory points (equal to the Block Size), the number of RMA computations and synchronizations required to compute four trajectory points are given in the last two columns. For B2PC5 and B1PC5 (Block Sizes of two and one, respectively), this requires two and four iterations of the computation shown in Figure 13.

3.5.1 Parallel Performance

It is evident from the last two columns of Table 3 that the B4PC5 integration method requires the smallest number of synchronizations in each of the partitions considered. In addition, maximizing the number of temporal parallel tasks while using the B4PC5 method also leads to the least amount of RMA computation for a given number of processors. With this result, simulations were first performed using the B4PC5 method and the partitions from Table 3, on four and eight of the CRAY's processors.

A system consisting of four PUMA 560 manipulators was modeled, and a test trajectory with a duration of one second was generated which consisted of lifting a 4kg object 0.8 meters straight up. Then an appropriate joint torque profile was

Table 4: Speedup results for B4PC5.

# CPU	Spatial Tasks, $\tilde{m}$	Temporal Tasks, $\tilde{p}$	T (sec.)	S_p
1	1	1	0.2410	1.00
4	1	4	0.0637	3.78
	2	2	0.0668	3.61
	4	1	0.0731	3.30
8	2	4	0.0390	6.18
	4	2	0.0417	5.78

computed which, when applied to the joints of the manipulators, would produce the desired motion. Using a fixed integration stepsize, these torques were used as input into the simulator. To get execution times as close to dedicated operation times as possible, a number of runs were performed during *low*-load periods on the CRAY, and the fastest times are reported in Table 4.

Examining these execution times for a given number of processors, it can be seen that they increase as the number of temporal parallel tasks, $\tilde{p}$, decrease. This is a direct result of the increased numbers of RMA computations that must be performed when $\tilde{p}$ is less than four (refer to Table 3). At the same time, the small variations in the amount of interprocessor synchronization seems to have much less effect on the relative performances of the different partitions. Consequently, the best speedups due to this parallelization, S_p, were achieved by using the greatest amount of temporal parallelism which were as high as 3.78 on four processors. Runs were also made while requesting all eight of the Y-MP's processors and a speedup of 6.18

was still achieved despite an inability to gain dedicated use of these processors in its multi-user environment.

3.5.2 Accuracy Performance

To this point, only the amount of computation required to evaluate a certain number of simulation points has been examined. The B4PC5 method offers a way to simulate the system using four processors in parallel. Because the method extrapolates farther from the known values in the predictor step, it is subject to larger truncation errors for a given stepsize. Consequently, the quality of these simulation results must be taken into account and compared with the smaller block size integration methods as well. To accomplish this, the same trajectory for the four PUMA system was used, and in this case, the error in the final position of the reference member was used as a measure of accuracy for the various integration methods. For the sake of brevity, only this value is reported and its accuracy was found to be representative of the accuracy of the other state variables in the system.

First, serial implementations of the one-, two-, and four-point Sliding Block Methods were used, and fixed integration stepsizes were used to carry out the simulation. The stepsize was varied over an appropriate range so that a profile of execution time versus error could be obtained, and the results are plotted in Figure 14. As expected, the larger Block Sizes lead to lower accuracy. This means that successively smaller stepsizes are required for B2PC5 and B4PC5 to achieve the same level of accuracy as B1PC5, and therefore, they must perform more iterations. This results in higher execution times over a wide range of accuracies.

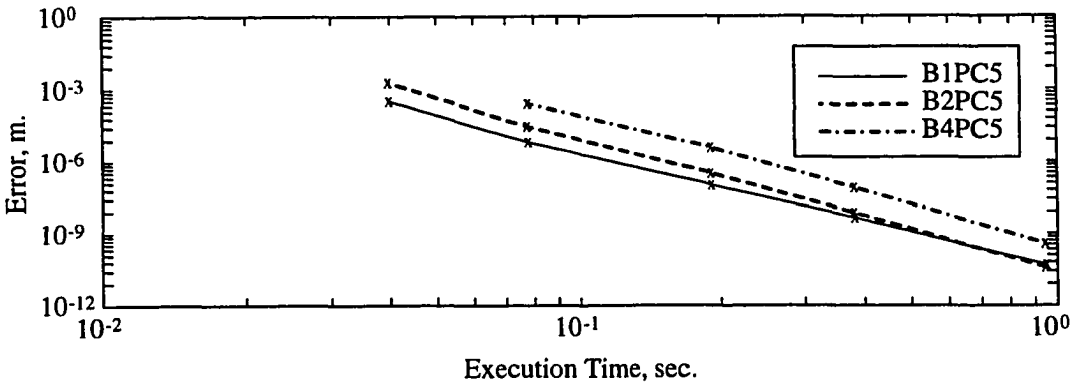


Figure 14: Serial accuracy performance of integration methods.

Table 5: Algorithmic slowdown with larger block sizes.

Block Size	T (sec.)	S_a
1	0.117	1.000
2	0.152	0.770
4	0.241	0.485

To quantify subsequent comparisons of relative performance between the different integration methods, an estimate of execution time, T, required to achieve an error of 10^{-6} meters was chosen as a benchmark. Table 5 lists the serial execution times for this benchmark from the data of Figure 14. This table shows speedups of less than one for B2PC5 and B4PC5 which we call their *algorithmic slowdown*, S_a. Using these results, the parallel speedups achieved with these methods, S_p, must be reduced by about one-half for B4PC5, and by one-fourth for B2PC5. The results are the speedups of these methods over the best serial method (B1PC5) for the given

Table 6: Accuracy performance for various partitions.

# CPU	Spatial Tasks, $\tilde{m}$	Temporal Tasks, $\tilde{p}$	Block Size	T (sec.)	S_p	S_a	S_T
4	1	4	4	0.0637	3.78	0.485	1.83
	2	2	4	0.0668	3.61	0.485	1.75
			2	0.0444	3.42	0.770	2.63
	4	1	4	0.0731	3.30	0.485	1.60
			2	0.0463	3.28	0.770	2.53
			1	0.0377	3.10	1.000	3.10
8	2	4	4	0.0390	6.18	0.485	3.00
	4	2	4	0.0417	5.78	0.485	2.80
			2	0.0295	5.15	0.770	3.97

accuracy (10^{-6} m) and are a measure of their *accuracy performance.*

In Table 6, the accuracy performance data for all of the possible partitions on four and eight processors are presented. The same benchmark is used in which the execution times, T, are for the parallel simulations achieving the error of 10^{-6} meters. S_p is the speedup of the simulation over its own serial performance, S_a is the algorithmic slowdown associated with the Block Size, and S_T is the speedup over the best serial implementation (its accuracy performance).

These results lead to a much different conclusion than when the speedup, S_p, alone was considered. Table 3 showed that synchronization requirements increased for the computation of a given number of simulation points as the Block Size decreased. At the same time, the accuracy of the method increased requiring the computation of fewer overall points. The final result is that the *smaller* block sizes

lead to better effective speedups. Since the system under consideration had four chains, the serial B1PC5 method was the best on four processors and the resulting speedup was 3.1. On eight processors, where B1PC5 could not be used and still have eight parallel tasks, the B2PC5 method showed the greatest speedup at 3.97. From these results, it appears that B4PC5 has little or no advantage over the other integration methods. However, when a smaller number of chains is simulated, this method allows a greater number of parallel tasks to be generated than the other methods and would enhance the performance.

3.6 Summary and Conclusions

In this chapter, two methods for achieving effective parallelization for dynamic simulation of multiple chain systems on a general-purpose parallel computer were presented. One approach that was discussed in [36] for parallel simulation of a single chain system was based on temporal parallelism achieved with the use of a parallel numerical integration method. The topology of multiple chain systems and the resulting form of the dynamics algorithm introduces a second form of parallelism. Called spatial parallelism, this results from the ability to compute the dynamics of individual chains simultaneously.

Both forms of parallelism were combined in an effort to achieve greater speedups. For the cases where the number of resulting parallel tasks was greater than the number of available processors, a partitioning mechanism was implemented to divide the computational task into fewer parallel tasks while minimizing the amount of interprocessor communication and maintaining a load balance. Runtime performance

results were collected for a four PUMA system on four and eight processors of a CRAY Y-MP8 for all of the reasonable task partitions.

The results show that the best effective speedups were obtained using the integration method with the smallest block size while partitioning the computation so that the number of parallel tasks was equal to the number of processors. On four processors of the CRAY, the simulation exhibited the greatest speedup (3.1) with spatial parallelism only, and the use of the serial predictor-corrector integration method. The greatest speedup on eight processors (3.97) was also achieved with the use of full spatial parallelism. In this case, a two-point parallel integration method had to be used to achieve the eight desired parallel tasks. It also appears that the four-point integration method would be beneficial if sixteen processors were available.

An additional benefit to these forms of parallelism is that they do not preclude any of the previous work mentioned in the introduction that dealt with the fine-grain parallel algorithms for computation of the robot dynamics quantities. The special-purpose architectures required could be set up in parallel and used in conjunction with the methods discussed in this chapter. The resulting combination of parallel computation could be thought of as occurring in three dimensions, where the third comes from parallelism in the computation *within* each chain, and shows promise for even greater speedups.

CHAPTER IV

Systems with Singular Configurations

4.1 Introduction

The previous two chapters have developed efficient dynamics algorithms and parallel implementations of the simulation for multiple closed-chain manipulators grasping a common object. Thus far, however, the case where one or more of the manipulators are in singular configurations has not been addressed. In comparison to the body of work related to the previous chapters, much less has been completed in this area. Despite this, singularities are an important issue in robotics. They can not only pose serious problems for control algorithms that are not designed to handle them, but also cause unreasonable and erroneous behavior in numerical simulations. The latter is caused by ill-conditioned matrices that occur at and near singularities and must be inverted in the dynamics algorithms.

The algorithm in the previous chapters was inspired by the work of Lilly and Orin [24]. While they developed a sequential algorithm with a computational complexity of $O(mN)$, an equivalent approach has been developed in this work that is much

Results from this chapter have been previously published in "Efficient Dynamic Simulation of Multiple-Manipulator Systems with Singular Configurations," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 24, no. 2, pp. 306-313, February 1994 (with P. Sadayappan and D. E. Orin).

more efficient for manipulators with six degrees-of-freedom but has a complexity of $O(mN^3)$. Unfortunately, they do not fully consider the case where chains may be in singular configurations. Rodriguez, Jain, and Kreutz-Delgado have also developed multiple manipulator algorithms in [30, 31]. They go on to state that a computation with an $O(m^3)$ complexity must be added to the simulation algorithm in the presence of singularities. The purpose of this chapter is to examine only those equations that are affected when manipulators become singular, and to develop algorithms with lower complexity and more efficient computations.

In particular, the equations that introduce the $O(m^3)$ computation in [30, 31] are examined, and the improved algorithm presented in this chapter introduces an $O(s^3)$ computation where s is the number of chains in singular configurations. This represents a significant improvement over previous results. The new algorithm can also detect when manipulators are *near* singularities, and in these situations, it is also numerically very robust. In addition to this result, efficient $O(m)$ procedures are also presented for the cases where only one or two chains are in singular configurations. These cases are important since it is undesirable, if not uncommon and easily avoidable, to deal with systems where many of the manipulators grasping the common object are singular simultaneously. Finally, by applying the algorithm developed for the case of two singularities to a dual-arm system, the resulting algorithm requires fewer total floating point operations than that of the method presented in [24], with the added benefit of being robust in the presence of singular manipulators.

In the next section, the dynamic equations used for simulating systems of mul-

tiple manipulators are reexamined, and particular attention is given to the computational problems introduced when manipulators are in singular configurations. In Section 4.3, an efficient algorithm is developed giving special attention to the cases where only one or two manipulators are in singular configurations. In the section following, the computational complexity for this algorithm is discussed. In addition, a specific example of special interest in many robotics applications, the dual-arm system, is examined and the especially efficient computational requirements for the affected equations are listed. Finally, Section 4.5 contains a mathematical analysis which characterizes the singular solutions, and proves that the accelerations computed by the dynamics are unique under these conditions.

4.2 Dynamic Equations for Systems With Singular Manipulators

If a manipulator is singular, its Jacobian is rank deficient which implies that $\mathbf{\Lambda}_k^{-1}$ is also singular and cannot be inverted as required by the algorithm presented in Chapter 2. This condition affects the computation of the reference member acceleration in Eq. (2.10) and the tip forces in Eq. (2.11). These equations are repeated here for reference:

$$\left[\mathbf{I}_r + \sum_{k=1}^{m} \mathbf{\Lambda}_k\right] \mathbf{a}_r = \left[\mathbf{f}_r^g - \bar{\mathbf{f}}_r + \sum_{k=1}^{m} \mathbf{\Lambda}_k \ddot{\mathbf{x}}_{k_{open}}\right], \tag{4.1}$$

$$\mathbf{f}_k = \mathbf{\Lambda}_k \left(\ddot{\mathbf{x}}_{k_{open}} - \mathbf{a}_r\right), \qquad \text{for } k = 1, \ldots, m. \tag{4.2}$$

This section presents an algorithm, beginning with Eqs. (2.7) and (2.8), that replaces the equations above when manipulators become singular. For clarity, this approach

does not begin with the efficient procedure presented in Section 2.3 for rigid grasps. However, the same analysis presented in this and subsequent sections can be carried, as desired, out by starting with Eqs. (2.16) and (2.20) in place of Eqs. (2.7) and (2.8).

To begin, Eqs. (2.7) and (2.8) are combined into a single block matrix equation as follows:

$$\begin{bmatrix} \mathbf{\Lambda}_1^{-1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{\Lambda}_2^{-1} & & \mathbf{0} & \mathbf{1} \\ \vdots & & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & & \mathbf{\Lambda}_m^{-1} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \cdots & \mathbf{1} & -\mathbf{I}_r \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_m \\ \mathbf{a}_r \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{x}}_{1_{open}} \\ \ddot{\mathbf{x}}_{2_{open}} \\ \vdots \\ \ddot{\mathbf{x}}_{m_{open}} \\ \bar{\mathbf{f}}_r - \mathbf{f}_r^g \end{bmatrix}, \tag{4.3}$$

where $\mathbf{1}$ is the 6×6 identity matrix and $\mathbf{0}$ is the 6×6 zero matrix. A linear system solution can then be performed on this entire matrix equation to determine the unknown tip forces and reference member acceleration. The computational complexity of this step results in the $O(m^3)$ term discussed in [30, 31]. This order of complexity is directly related to the size of the matrix in the equation to be solved which is $(6m + 6) \times (6m + 6)$, and which grows linearly with the number of chains in the system. This is undesirable because the cubic complexity can lead to a large amount of computation. Also, the numerical accuracy of the method may be problematic since it decreases significantly for very large systems of equations.

Before developing the efficient algorithms in this chapter which deal with systems where some of the chains are in singular configurations, some additional notation is defined. First, the chains are numbered so that chains $1, \ldots, (m - s)$ are the nonsingular manipulators. Then, the following two quantities illustrated in Figure 15

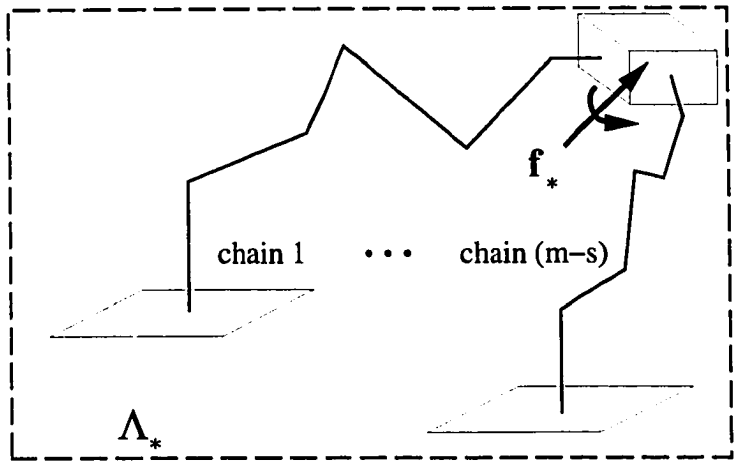


Figure 15: Dynamic variables for the nonsingular subsystem of chains.

are defined:

$$\Lambda_* = \mathbf{I}_r + \sum_{k=1}^{m-s} \Lambda_k, \tag{4.4}$$

$$\mathbf{f}_* = \mathbf{f}_r^g - \bar{\mathbf{f}}_r + \sum_{k=1}^{m-s} \Lambda_k \ddot{\mathbf{x}}_{k_{open}}. \tag{4.5}$$

The first quantity, Λ_*, can be thought of as the operational space inertia matrix of the nonsingular subsystem of chains which consists of the $m - s$ nonsingular manipulators and the reference member, and is computed at the reference point described in Figure 2. Likewise, $\mathbf{f}_*$ is the sum of all of the forces of this subsystem that are exerted on the reference member at the reference point.

From these two quantities, the acceleration of the reference member without the interaction of the singular chains (called the open, nonsingular subsystem acceleration) would be determined by $\Lambda_*^{-1}\mathbf{f}_*$, which is equivalent to Eq. (4.1) when $s = 0$. This implies that the computation involving the nonsingular chains will be unmod-

ified, and therefore, introduces no additional computation in the algorithm from Chapter 2. What is modified, however, is the computation of the forces at the tips of the singular chains and the reference member acceleration. The following section describes an efficient way to perform these computations.

4.3 Efficient Singularity Algorithm

In this section, algorithms for the computation of the reference member acceleration and the tip forces of each chain are presented for systems with m manipulators where an arbitrary number of them are in singular configurations. These algorithms have a lower complexity than the one described in the previous section which solved Eq. (4.3) directly. In the first two parts, algorithms are presented for the cases when one and two manipulators in a general m-manipulator system are singular. Then, these results are extended to the case where the system has an arbitrary number of singular chains, s.

4.3.1 One Singular Chain

For an m-manipulator system with one singular chain (chain m), Eq. (4.3) is rearranged as follows to isolate the singular chain:

$$\begin{bmatrix} \Lambda_1^{-1} & 0 & \cdots & 0 & 1 & 0 \\ 0 & \Lambda_2^{-1} & & 0 & 1 & 0 \\ \vdots & & \ddots & & \vdots & \\ 0 & 0 & & \Lambda_{m-1}^{-1} & 1 & 0 \\ 1 & 1 & \cdots & 1 & -\mathbf{I}_r & 1 \\ 0 & 0 & & 0 & 1 & \Lambda_m^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{m-1} \\ \mathbf{a}_r \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{x}}_{1_{open}} \\ \ddot{\mathbf{x}}_{2_{open}} \\ \vdots \\ \ddot{\mathbf{x}}_{m-1_{open}} \\ \bar{\mathbf{f}}_r - \mathbf{f}_r^g \\ \ddot{\mathbf{x}}_{m_{open}} \end{bmatrix}. \tag{4.6}$$

An efficient block elimination approach is then used to begin reducing this system. The inverse of chain m's operational space inertia matrix is singular and, therefore,

positive semidefinite. The remainder are strictly positive definite, and can be used to eliminate the first $m-1$ identity matrices in the mth row. This is accomplished by premultiplying row k of Eq. (4.6) by $\mathbf{\Lambda}_k$ and subtracting it from the mth row (for $k = 1, \ldots, m-1$). This results in the following system of equations:

$$\begin{bmatrix} \mathbf{\Lambda}_1^{-1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_2^{-1} & & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \vdots & & \ddots & & \vdots & \\ \mathbf{0} & \mathbf{0} & & \mathbf{\Lambda}_{m-1}^{-1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{\Lambda}_* & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{1} & \mathbf{\Lambda}_m^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{m-1} \\ \mathbf{a}_r \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{x}}_{1_{open}} \\ \ddot{\mathbf{x}}_{2_{open}} \\ \vdots \\ \ddot{\mathbf{x}}_{m-1_{open}} \\ -\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m_{open}} \end{bmatrix}, \tag{4.7}$$

where $\mathbf{\Lambda}_*$ and $\mathbf{f}_*$ are the quantities defined in the previous section with $s = 1$. This elimination step has accumulated dynamic quantities related to the nonsingular subsystem, namely its inertia and the forces acting upon it, and decoupled its dynamic equation as well as the singular chain's from the remainder of the system so that a much smaller system of equations needs to be solved.

This system can be reduced further, since $\mathbf{\Lambda}_*$ is positive definite (it is the sum of positive definite matrices), and therefore, nonsingular. Using its inverse, the equations in the last two rows are reduced to the following:

$$\begin{bmatrix} -\mathbf{\Lambda}_* & \mathbf{1} \\ \mathbf{0} & \mathbf{\Lambda}_m^{-1} + \mathbf{\Lambda}_*^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_r \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} -\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_* \end{bmatrix}. \tag{4.8}$$

Because $\mathbf{\Lambda}_m^{-1}$ is positive semidefinite and $\mathbf{\Lambda}_*^{-1}$ is positive definite, their sum is also positive definite, and therefore, invertible. As a result, a solution to the system may be obtained by performing back-substitution on the block matrix equations defined by Eq. (4.8) and the first $m-1$ rows of Eq. (4.7). This results in the following

algorithm:

$$\mathbf{f}_m = \left(\mathbf{\Lambda}_m^{-1} + \mathbf{\Lambda}_*^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_*\right), \tag{4.9}$$

$$\mathbf{a}_r = \mathbf{\Lambda}_*^{-1}\left(\mathbf{f}_m + \mathbf{f}_*\right), \tag{4.10}$$

$$\mathbf{f}_k = \mathbf{\Lambda}_k\left(\ddot{\mathbf{x}}_{k_{open}} - \mathbf{a}_r\right) \qquad \text{for all } k = 1, \ldots, m-1. \tag{4.11}$$

Note that there is always a unique solution for the motion of an object in a physical system which means that the solution for $\mathbf{a}_r$ will be unique. Since $\mathbf{\Lambda}_m^{-1} + \mathbf{\Lambda}_*^{-1}$ is also invertible, there will always be a unique solution for the tip forces in a system with one singular chain as well.

4.3.2 Two Singular Chains

With two singular chains ($s = 2$), the block matrix equation is again rewritten by moving the singular inverse operational space inertia matrices (for chains $m-1$ and m) to the bottom as follows:

$$\begin{bmatrix} \mathbf{\Lambda}_1^{-1} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Lambda}_2^{-1} & & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \vdots & & \ddots & & & \vdots & \\ \mathbf{0} & \mathbf{0} & & \mathbf{\Lambda}_{m-2}^{-1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & & \mathbf{1} & -\mathbf{I}_r & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{1} & \mathbf{\Lambda}_{m-1}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{\Lambda}_m^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \vdots \\ \mathbf{f}_{m-2} \\ \mathbf{a}_r \\ \mathbf{f}_{m-1} \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{x}}_{1_{open}} \\ \ddot{\mathbf{x}}_{2_{open}} \\ \vdots \\ \ddot{\mathbf{x}}_{m-2_{open}} \\ \bar{\mathbf{f}}_r - \mathbf{f}_r^g \\ \ddot{\mathbf{x}}_{m-1_{open}} \\ \ddot{\mathbf{x}}_{m_{open}} \end{bmatrix}, \tag{4.12}$$

As in the previous section, the first $m-2$ identity matrices in the $(m-1)$st row are eliminated by taking the operational space inertia matrices of the first $m-2$ chains and performing block eliminations. As a result, the last three equations can

be decoupled from the others and written as follows:

$$\begin{bmatrix} -\Lambda_* & 1 & 1 \\ 1 & \Lambda_{m-1}^{-1} & 0 \\ 1 & 0 & \Lambda_m^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_r \\ \mathbf{f}_{m-1} \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} -\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m-1_{open}} \\ \ddot{\mathbf{x}}_{m_{open}} \end{bmatrix}. \tag{4.13}$$

Because Λ_* is the sum of all of the positive definite inertia matrices, it will always be positive definite, and can be used to eliminate the two identity matrices in the first column. The resulting system of equations appears as follows:

$$\begin{bmatrix} -\Lambda_* & 1 & 1 \\ 0 & \Lambda_{m-1}^{-1} + \Lambda_*^{-1} & \Lambda_*^{-1} \\ 0 & \Lambda_*^{-1} & \Lambda_m^{-1} + \Lambda_*^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_r \\ \mathbf{f}_{m-1} \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} -\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m-1_{open}} - \Lambda_*^{-1}\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m_{open}} - \Lambda_*^{-1}\mathbf{f}_* \end{bmatrix}. \tag{4.14}$$

The matrix, $\Lambda_{m-1}^{-1} + \Lambda_*^{-1}$ (or $\Lambda_m^{-1} + \Lambda_*^{-1}$ for that matter), is nonsingular, and can be used to reduce the last 2×2 block of matrices. The resulting equation appears as follows:

$$\begin{bmatrix} -\Lambda_* & 1 & 1 \\ 0 & \Lambda_{m-1}^{-1} + \Lambda_*^{-1} & \Lambda_*^{-1} \\ 0 & 0 & \Lambda_m^{-1} + \Lambda_*^{-1} - \Lambda_*^{-1}\left(\Lambda_{m-1}^{-1} + \Lambda_*^{-1}\right)^{-1}\Lambda_*^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_r \\ \mathbf{f}_{m-1} \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} -\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m-1_{open}} - \Lambda_*^{-1}\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m_{open}} - \Lambda_*^{-1}\mathbf{f}_* - \Lambda_*^{-1}\left(\Lambda_{m-1}^{-1} + \Lambda_*^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m-1_{open}} - \Lambda_*^{-1}\mathbf{f}_*\right) \end{bmatrix}. \tag{4.15}$$

The rank deficiency (dimension of the null space) in the original block matrix equation of Eq. (4.3) will be equal to the rank deficiency in the lower right block of Eq. (4.15). If this block is nonsingular, then a unique solution can be found for $\mathbf{a}_r$ and all of the $\mathbf{f}_k$'s using the following equations:

$$\mathbf{f}_m = \left[\Lambda_m^{-1} + \Lambda_*^{-1} - \Lambda_*^{-1}\left(\Lambda_{m-1}^{-1} + \Lambda_*^{-1}\right)^{-1}\Lambda_*^{-1}\right]^{-1}\left[\ddot{\mathbf{x}}_{m_{open}} - \Lambda_*^{-1}\mathbf{f}_* - \Lambda_*^{-1}\left(\Lambda_{m-1}^{-1} + \Lambda_*^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m-1_{open}} - \Lambda_*^{-1}\mathbf{f}_*\right)\right], \tag{4.16}$$

$$\mathbf{f}_{m-1} = \left(\Lambda_{m-1}^{-1} + \Lambda_*^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m-1_{open}} - \Lambda_*^{-1}\mathbf{f}_* - \Lambda_*^{-1}\mathbf{f}_m\right), \tag{4.17}$$

$$\mathbf{a}_r = \mathbf{\Lambda}_*^{-1}\left(\mathbf{f}_{m-1} + \mathbf{f}_m + \mathbf{f}_*\right), \tag{4.18}$$

$$\mathbf{f}_k = \mathbf{\Lambda}_k\left(\ddot{\mathbf{x}}_{k_{open}} - \mathbf{a}_r\right) \qquad \text{for all } k = 1, \ldots, m-2. \tag{4.19}$$

If, on the other hand, the lower right block in the matrix of Eq. (4.15) is singular, the rigid body dynamics of the system of manipulators implies that the matrix equation will still be consistent so that solutions exist. In this case, however, an infinite number of solutions will exist for the tip forces. At the same time, only one solution will be found for the motion ($\mathbf{a}_r$) in the system. A proof of these statements is given in Section 4.5.

This can also be intuitively understood because one or more components of the force vector, $\mathbf{f}_m$, will be opposed by an infinite inertia that is created by a combination of the nonsingular subsystem and the singular chain $m-1$. These components will also see an infinite inertia from chain m as well, and regardless of the value assigned to them, no motion will result. In this case, the singularities of both chains can be thought of as having "lined-up." Mathematically, the null spaces of $\mathbf{J}_{m-1}^T$ and $\mathbf{J}_m^T$ overlap as shown in Section 4.5. Note, however, that values given to these components of $\mathbf{f}_m$ will be negated by corresponding components in $\mathbf{f}_{m-1}$.

To solve for $\mathbf{f}_m$, in this case, a linear system solution with full pivoting can be employed which will produce one or more zero rows that correspond to the components of this force that can be set to any value. For convenience, they are set to zero without affecting the resulting acceleration of the reference member or any of the chains' joints. The remaining components of $\mathbf{f}_m$ can then be determined, and the solution for the remaining forces and reference member acceleration proceeds as

specified in Eqs. (4.17), (4.18), and (4.19).

4.3.3 More Than Two Singular Chains

By extending this procedure to a system of m chains with an arbitrary number of singular chains ($3 \leq s \leq m$), the following set of equations need to be solved to find the required tip forces exerted by the singular chains onto the reference member:

$$\begin{bmatrix} \Lambda_{m-s+1}^{-1} + \Lambda_*^{-1} & \Lambda_*^{-1} & \cdots & \Lambda_*^{-1} \\ \Lambda_*^{-1} & \Lambda_{m-s+2}^{-1} + \Lambda_*^{-1} & & \Lambda_*^{-1} \\ \vdots & & \ddots & \vdots \\ \Lambda_*^{-1} & \Lambda_*^{-1} & \cdots & \Lambda_m^{-1} + \Lambda_*^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_{m-s+1} \\ \mathbf{f}_{m-s+2} \\ \vdots \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{x}}_{m-s+1_{open}} - \Lambda_*^{-1}\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m-s+2_{open}} - \Lambda_*^{-1}\mathbf{f}_* \\ \vdots \\ \ddot{\mathbf{x}}_{m_{open}} - \Lambda_*^{-1}\mathbf{f}_* \end{bmatrix}. \quad (4.20)$$

Then, the following set of equations are used to compute the remainder of the unknowns:

$$\mathbf{a}_r = \Lambda_*^{-1}\left(\sum_{k=1}^{s} \mathbf{f}_{m-k+1} + \mathbf{f}_*\right), \quad (4.21)$$

$$\mathbf{f}_k = \Lambda_k\left(\ddot{\mathbf{x}}_{k_{open}} - \mathbf{a}_r\right) \qquad \text{for all } k = 1, \ldots, m-s. \quad (4.22)$$

Because $\Lambda_{m-s+1}^{-1} + \Lambda_*^{-1}$ is positive definite (using the same argument as before), it can be inverted and used to reduce the size of the block matrix equation that needs to be solved. The resulting size of the block matrix in this equation is $6(s-1) \times 6(s-1)$. This equation is then used to compute the values for $\mathbf{f}_{m-s+2}, \ldots, \mathbf{f}_m$, and the following equation is used to compute $\mathbf{f}_{m-s+1}$:

$$\mathbf{f}_{m-s+1} = \left(\Lambda_{m-s+1}^{-1} + \Lambda_*^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m-s+1_{open}} - \Lambda_*^{-1}\mathbf{f}_* - \Lambda_*^{-1}\sum_{k=1}^{s-1} \mathbf{f}_{m-k+1}\right). \quad (4.23)$$

By the same arguments that were given in the previous section, the smaller block matrix system of equations for finding $\mathbf{f}_{m-s+2}, \ldots, \mathbf{f}_m$, will always have at least one solution, but could have an infinite number of them. Therefore, a method for the solution of this linear system should again employ full pivoting to identify the free components of the force vectors (if any), and set them to zero.

4.4 Computational Requirements

Chapter 2 presented an efficient algorithm for the evaluation of $\mathbf{\Lambda}_k^{-1}$ and $\ddot{\mathbf{x}}_{k_{open}}$ for each chain in the system. When none of these chains are in singular configurations, the original algorithm may be used which consists of Eqs. (4.1) and (4.2). They require m inverses (of size 6×6) to find the operational space inertia matrices for all of the manipulators, followed by a 6×6 linear system solution to find $\mathbf{a}_r$. The computation required to compute these quantities ($\mathbf{a}_r$ and the $\mathbf{f}_k$'s) has a complexity of $O(m)$ which does not change the overall complexity of the dynamic simulation algorithm.

With the new algorithms presented in this chapter and the same system of manipulators, the inverses for all m operational space inertia matrices must still be attempted in order to identify the singular or nearly singular chains (the ones with inordinately large condition numbers). The number of such matrices indicates which of the procedures from the previous section should be used to compute the tip forces and acceleration of the reference member.

All of these methods require an additional 6×6 inverse of the positive definite $\mathbf{\Lambda}_*$ matrix at a cost of [141M, 90A] [37]. For the case with one singular chain, a linear

system solution is used to compute the force exerted at the tip of the singular chain, and a matrix-vector multiply is used to compute the acceleration of the reference member. Note that this is the opposite of what is done in the original algorithm. Computation of the rest of the forces proceeds as before.

With two singular chains, the inverse of $\mathbf{\Lambda}_{m-1}^{-1} + \mathbf{\Lambda}_{*}^{-1}$ can be performed. However, less computation is required if a root-free Cholesky decomposition is performed. This decomposition requires $[(\frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n)\mathrm{M}, (\frac{1}{6}n^3 - \frac{1}{6}n)\mathrm{A}]$ for an $n \times n$ matrix which is [50M, 35A] when $n = 6$. To perform a matrix-vector multiply with the inverse, one back-substitution is used requiring $[(n^2)\mathrm{M}, (n^2 - n)\mathrm{A}]$, or [36M, 30A] for $n = 6$ [37]. Matrix-matrix multiplies are accomplished with six back substitutions (one for each column of the matrix to be multiplied). Therefore linear systems solutions are required for both, $\mathbf{f}_m$ and $\mathbf{f}_{m-1}$, and a matrix-vector multiply is used to compute the acceleration.

In the unlikely case when more than two chains are singular, the last part of the previous section shows that the size of the linear system to be solved, $(6s - 6) \times (6s - 6)$, grows linearly with s. This solution requires an elimination method with full pivoting which has computational complexity of $\mathrm{O}(s^3)$. This is a significant improvement over the previous $\mathrm{O}(m^3)$ results where the size of the matrix is $(6m + 6) \times (6m + 6)$. As before, a 6×6 linear system solution yields $\mathbf{f}_{m-s+1}$, and a matrix-vector multiply computes the acceleration.

4.4.1 Special Case: The Dual-Arm System

Another interesting and useful result can be obtained from the solution procedure presented in this chapter when a dual-arm system is simulated. This case is of particular interest because many applications for such a system have been cited and a significant amount of research has been performed in this area [42]. A comparison of the results using the method summarized in Section 2 [Eqs. (4.1) and (4.2)] and that presented in the section discussing two singular chains [Eqs. (4.16), (4.17), and (4.18)] is discussed in this section. The two sets of equations are referred to as Method I and Method II, respectively.

In Table 7, the equations for these methods have been rewritten specifically for the dual-arm system. It has been assumed that efficient algorithms have been employed to compute the open-chain accelerations and the inverse operational space inertia matrices for both manipulators, and only the computations to find the forces at the tips and the acceleration of the reference member are examined here. When Method II is employed for this system, no checking for chains in singular configurations needs to be done before these equations are employed. Instead, the operational space inertia matrices of the two manipulators are treated as if they were singular. As a consequence, the terms, $\mathbf{\Lambda}_*$ and $\mathbf{f}_*$, simplify to the reference member inertia, $\mathbf{I}_r$, and a force term, $\mathbf{f}_r^g - \bar{\mathbf{f}}_r$.

Although the equations for Method I appear to be much simpler, detailed inspection of the methods indicates that Method II can be performed using significantly fewer computations when the reference point is placed at the center of gravity of

Table 7: Methods for computation of the dual-arm system.

Given: $\mathbf{I}_r$ and $\mathbf{I}_r^{-1}$ (constant, diagonal), $\bar{\mathbf{f}}_r - \mathbf{f}_r^g$, and

Λ_k^{-1}, $\ddot{\mathbf{x}}_{k_{open}}$, for $k = 1, 2$.

Method I: (from Chapter 2)

$$\begin{aligned} \mathbf{a}_r &= (\Lambda_1 + \Lambda_2 + \mathbf{I}_r)^{-1} \left[\Lambda_1 \ddot{\mathbf{x}}_{1_{open}} + \Lambda_2 \ddot{\mathbf{x}}_{2_{open}} - \left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)\right], \\ \mathbf{f}_k &= \Lambda_k \left(\ddot{\mathbf{x}}_{k_{open}} - \mathbf{a}_r\right) \qquad k = 1, 2. \end{aligned}$$

Method II: (this chapter)

$$\begin{aligned} \mathbf{f}_2 &= \left[\Lambda_2^{-1} + \mathbf{I}_r^{-1} - \mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\mathbf{I}_r^{-1}\right]^{-1} \cdot \\ &\quad \left\{\ddot{\mathbf{x}}_{2_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right) - \mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\left[\ddot{\mathbf{x}}_{1_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)\right]\right\} \\ \mathbf{f}_1 &= \left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\left[\ddot{\mathbf{x}}_{1_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right) - \mathbf{I}_r^{-1}\mathbf{f}_2\right] \\ \mathbf{a}_r &= \mathbf{I}_r^{-1}\mathbf{f}_1 + \mathbf{I}_r^{-1}\mathbf{f}_2 - \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right) \end{aligned}$$

---

the reference member, and the axes of the coordinate frame correspond to the reference member's principal axes. Under these circumstances, the inertia matrix for the reference member, $\mathbf{I}_r$, will be diagonal as will its inverse. Since this coordinate frame is also fixed to the reference member, this quantity is constant and can be computed off-line.

The computation required for both methods can then be reduced in certain steps by using the knowledge that the inertia matrix, $\mathbf{I}_r$ is diagonal, and the operational space inertia matrices are symmetric. One significant observation is that it becomes much less costly to compute the inverse of $\Lambda_1^{-1} + \mathbf{I}_r^{-1}$ directly, rather than performing

Table 8: Computational requirements for Method 1.

Computation		×	+
$\left(\mathbf{\Lambda}_k^{-1}\right)^{-1}$	$k=1,2$	282	180
$\mathbf{\Lambda}_k\ddot{\mathbf{x}}_{k_{open}}$	$k=1,2$	72	60
$\sum_{k=1}^{2}\mathbf{\Lambda}_k\ddot{\mathbf{x}}_{k_{open}}-\left(\bar{\mathbf{f}}_r-\mathbf{f}_r^g\right)$			12
$\mathbf{\Lambda}_1+\mathbf{\Lambda}_2+\mathbf{I}_r$			27
$\mathbf{a}_r$		86	65
$\ddot{\mathbf{x}}_{k_{open}}-\mathbf{a}_r$	$k=1,2$		12
$\mathbf{f}_k$	$k=1,2$	72	60
Total:		512	416

the Cholesky decomposition and back-substitution, because the latter involves the diagonal matrix, $\mathbf{I}_r^{-1}$, and it is more efficient to perform matrix-matrix multiplies instead. This results in the total number of operations shown in Tables 8 and 9. As a result Method II reduces the amount of computation by approximately 30%. In addition to this computational advantage, Method II is more robust because it will also work correctly and accurately in situations when one or both arms are at or near singularities.

4.5 Singularity Conditions

In this section, the case when the dynamic equations of motion result in a singular system of equations is examined. The conditions under which the system matrix becomes singular and its null space are characterized in the first part. Then, in the second part, it is shown that the reference member acceleration will be unique even

Table 9: Computational requirements for Method 2.

Computation		×	+
$\Lambda_k^{-1} + \mathbf{I}_r^{-1}$	$k = 1,2$		12
$\mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)$		6	
$\ddot{\mathbf{x}}_{k_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)$	$k = 1,2$		12
$\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}$ (6 × 6 inverse)		141	90
$\mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}$		36	
$\mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\mathbf{I}_r^{-1}$		21	
$\mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\left[\ddot{\mathbf{x}}_{1_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)\right]$		36	30
$\left[\ddot{\mathbf{x}}_{2_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)\right] - \mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\left[\ddot{\mathbf{x}}_{1_{open}} + \mathbf{I}_r^{-1}\left(\mathbf{f}_r - \mathbf{f}_r^g\right)\right]$			6
$\left(\Lambda_2^{-1} + \mathbf{I}_r^{-1}\right) - \mathbf{I}_r^{-1}\left(\Lambda_1^{-1} + \mathbf{I}_r^{-1}\right)^{-1}\mathbf{I}_r^{-1}$			21
$\mathbf{f}_2$ (6 × 6 linear system solution)		86	65
$\mathbf{I}_r^{-1}\mathbf{f}_2$		6	
$\left[\ddot{\mathbf{x}}_{1_{open}} + \mathbf{I}_r^{-1}\left(\bar{\mathbf{f}}_r - \mathbf{f}_r^g\right)\right] - \mathbf{I}_r^{-1}\mathbf{f}_2$			6
$\mathbf{f}_1$		36	30
$\mathbf{I}_r^{-1}\mathbf{f}_1$		6	
$\mathbf{a}_r$			12
Total:		374	284

when the manipulator tip forces are not.

4.5.1 Characterization of Singular Solutions

In this section, solutions to the dynamic equations are characterized for the case when two manipulators are in singular configurations. This problem reduces to showing that the linear system of equations, $\mathbf{Ax} = \mathbf{b}$, has a solution when $\mathbf{A}$ and $\mathbf{b}$

are taken from the last line of Eq. (4.15):

$$\mathbf{A} = \mathbf{\Lambda}_m^{-1} + \mathbf{\Lambda}_*^{-1} - \mathbf{\Lambda}_*^{-1}\left(\mathbf{\Lambda}_*^{-1} + \mathbf{\Lambda}_{m-1}^{-1}\right)^{-1}\mathbf{\Lambda}_*^{-1}, \text{ and} \tag{4.24}$$

$$\mathbf{b} = \ddot{\mathbf{x}}_{m_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_* - \mathbf{\Lambda}_*^{-1}\left(\mathbf{\Lambda}_*^{-1} + \mathbf{\Lambda}_{m-1}^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m-1_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_*\right). \tag{4.25}$$

In these equations, $\mathbf{\Lambda}_{m-1}^{-1}$ and $\mathbf{\Lambda}_m^{-1}$, are symmetric, positive semidefinite matrices, and $\mathbf{\Lambda}_*^{-1}$ is symmetric, positive definite. When $\mathbf{A}$ is nonsingular there will always be a unique solution. The next step is to determine the conditions under which $\mathbf{A}$ becomes singular.

Note from Eqs. (2.5) and (2.6) that $\mathbf{\Lambda}_{m-1}^{-1} = \mathbf{J}_{m-1}\mathbf{H}_{m-1}^{-1}\mathbf{J}_{m-1}^T$, where $\mathbf{H}_{m-1}^{-1}$ is also symmetric and positive definite, but $\mathbf{J}_{m-1}$ is rank deficient. Therefore, Eq. (4.24) can be rewritten as follows:

$$\mathbf{A} = \mathbf{\Lambda}_m^{-1} + \mathbf{\Lambda}_*^{-1} - \mathbf{\Lambda}_*^{-1}\left(\mathbf{\Lambda}_*^{-1} + \mathbf{J}_{m-1}\mathbf{H}_{m-1}^{-1}\mathbf{J}_{m-1}^T\right)^{-1}\mathbf{\Lambda}_*^{-1}. \tag{4.26}$$

The term inside the parentheses can now be rewritten by applying the following lemma:

Lemma 1 (Woodbury) [40] *If* $\mathbf{P}$, $\mathbf{Q}$, *and* $\mathbf{Q}^{-1} + \mathbf{R}\mathbf{P}^{-1}\mathbf{S}$ *are nonsingular then the following holds:*

$$(\mathbf{P} + \mathbf{S}\mathbf{Q}\mathbf{R})^{-1} = \mathbf{P}^{-1} - \mathbf{P}^{-1}\mathbf{S}\left(\mathbf{Q}^{-1} + \mathbf{R}\mathbf{P}^{-1}\mathbf{S}\right)^{-1}\mathbf{R}\mathbf{P}^{-1} \tag{4.27}$$

The proof can be completed by multiplying the right hand side by $(\mathbf{P} + \mathbf{S}\mathbf{Q}\mathbf{R})$ *and, with some manipulation, showing that it is equal to the identity matrix.*

The $\mathbf{A}$ matrix can then be written in the following form:

$$\mathbf{A} = \mathbf{\Lambda}_m^{-1} + \mathbf{\Lambda}_*^{-1} - \mathbf{\Lambda}_*^{-1}\left[\mathbf{\Lambda}_* - \mathbf{\Lambda}_*\mathbf{J}_{m-1}\left(\mathbf{H}_{m-1} + \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\mathbf{J}_{m-1}\right)^{-1}\mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\right]\mathbf{\Lambda}_*^{-1}. \tag{4.28}$$

This simplifies to:

$$\mathbf{A} = \mathbf{\Lambda}_m^{-1} + \mathbf{J}_{m-1}\left(\mathbf{H}_{m-1} + \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\mathbf{J}_{m-1}\right)^{-1}\mathbf{J}_{m-1}^T, \tag{4.29}$$

$$= \mathbf{J}_m\mathbf{H}_m^{-1}\mathbf{J}_m^T + \mathbf{J}_{m-1}\left(\mathbf{H}_{m-1} + \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\mathbf{J}_{m-1}\right)^{-1}\mathbf{J}_{m-1}^T. \tag{4.30}$$

Since $\mathbf{H}_m^{-1}$ and the term inside the parentheses are symmetric and positive definite, $\mathbf{A}$ is also symmetric, and *usually* positive definite. The only time that $\mathbf{A}$ is singular is when there exists a vector which lies in the null spaces of both $\mathbf{J}_{m-1}^T$ and $\mathbf{J}_m^T$. When this is the case, then it can be formally stated that $\mathbf{A}^T\mathbf{v} = \mathbf{A}\mathbf{v} = \mathbf{0} \;\; \forall\, \mathbf{v} \in N(\mathbf{J}_{m-1}^T) \cap N(\mathbf{J}_m^T)$.

To also show the existence of solutions even when $\mathbf{A}$ is singular, a set of vectors, $\mathbf{v}$, must satisfy the conditions in the following theorem which is stated without proof:

Theorem 1 (Fredholm Alternative) [43] *The equation,* $\mathbf{Ax} = \mathbf{b}$ *has a solution if and only if* $\mathbf{v}^T\mathbf{b} = \mathbf{0}$ $\forall\mathbf{v}$ *that satisfies* $\mathbf{A}^T\mathbf{v} = \mathbf{0}$.

Therefore, $\mathbf{v}^T\mathbf{b} = \mathbf{0}$ must be true for this set of vectors. To begin, the following equation is used:

$$\mathbf{v}^T\mathbf{b} = \mathbf{v}^T\left[\ddot{\mathbf{x}}_{m_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_* - \mathbf{\Lambda}_*^{-1}\left(\mathbf{\Lambda}_*^{-1} + \mathbf{\Lambda}_{m-1}^{-1}\right)^{-1}\left(\ddot{\mathbf{x}}_{m-1_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_*\right)\right]. \tag{4.31}$$

Using Lemma 1 again, this also can be rewritten as follows:

$$\mathbf{v}^T\mathbf{b} = \mathbf{v}^T\left\{\ddot{\mathbf{x}}_{m_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_* - \mathbf{\Lambda}_*^{-1}\left[\mathbf{\Lambda}_* - \mathbf{\Lambda}_*\mathbf{J}_{m-1}\left(\mathbf{H}_{m-1} + \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\mathbf{J}_{m-1}\right)^{-1} \cdot \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\right]\left(\ddot{\mathbf{x}}_{m-1_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_*\right)\right\} \tag{4.32}$$

$$= \mathbf{v}^T\left[\ddot{\mathbf{x}}_{m_{open}} - \ddot{\mathbf{x}}_{m-1_{open}} + \mathbf{J}_{m-1}\left(\mathbf{H}_{m-1} + \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\mathbf{J}_{m-1}\right)^{-1} \cdot \mathbf{J}_{m-1}^T\mathbf{\Lambda}_*\left(\ddot{\mathbf{x}}_{m-1_{open}} - \mathbf{\Lambda}_*^{-1}\mathbf{f}_*\right)\right]. \tag{4.33}$$

Since, by definition of the null space, $\mathbf{J}_{m-1}^T\mathbf{v} = \mathbf{0}$, this simplifies even further:

$$\mathbf{v}^T\mathbf{b} = \mathbf{v}^T\left(\ddot{\mathbf{x}}_{m_{open}} - \ddot{\mathbf{x}}_{m-1_{open}}\right). \tag{4.34}$$

Using Eq. (2.5) it is rewritten as follows:

$$\mathbf{v}^T\mathbf{b} = \mathbf{v}^T\left(\ddot{\mathbf{x}}_m + \mathbf{J}_m\mathbf{H}_m^{-1}\mathbf{J}_m^T\mathbf{f}_m - \ddot{\mathbf{x}}_{m-1} - \mathbf{J}_{m-1}\mathbf{H}_{m-1}^{-1}\mathbf{J}_{m-1}^T\mathbf{f}_{m-1}\right), \tag{4.35}$$

$$= \mathbf{v}^T\left(\ddot{\mathbf{x}}_m - \ddot{\mathbf{x}}_{m-1}\right). \tag{4.36}$$

The right hand side is indeed zero because the closed-chain accelerations of all of the chains are equal [and equal to the reference member acceleration as stated for Eq. (2.7), $\ddot{\mathbf{x}}_k = \mathbf{a}_r$]. Therefore the existence of solutions is proven.

The same method for characterizing the singular solutions may still be used on the system of equations when the system contains more than two manipulators in singular configurations. Eq. (4.20) is manipulated, in this case, to obtain a system of equations which will now grow linearly with the number of singular manipulators. The resulting matrix will be singular when the null spaces of any pair of singular Jacobians overlap; that is, the singular "directions" of any two manipulators coincide.

4.5.2 Uniqueness of Motion

To show that for the case where two manipulators are in singular configurations, the solution for the reference member acceleration, $\mathbf{a}_r$, is still unique, a previous form of the dynamic equations from Eq. (4.13) is needed:

$$\begin{bmatrix} -\mathbf{\Lambda}_* & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{\Lambda}_{m-1}^{-1} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{\Lambda}_m^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{a}_r \\ \mathbf{f}_{m-1} \\ \mathbf{f}_m \end{bmatrix} = \begin{bmatrix} -\mathbf{f}_* \\ \ddot{\mathbf{x}}_{m-1_{open}} \\ \ddot{\mathbf{x}}_{m_{open}} \end{bmatrix}. \tag{4.37}$$

From this set of equations, the following three equations for $\mathbf{a}_r$ can be written:

$$\mathbf{a}_r = \mathbf{\Lambda}_*^{-1}\left(\mathbf{f}_* + \mathbf{f}_{m-1} + \mathbf{f}_m\right), \tag{4.38}$$

$$= -\mathbf{\Lambda}_{m-1}^{-1}\mathbf{f}_{m-1} + \ddot{\mathbf{x}}_{m-1_{open}}, \text{ and} \tag{4.39}$$

$$= -\mathbf{\Lambda}_m^{-1}\mathbf{f}_m + \ddot{\mathbf{x}}_{m_{open}}. \tag{4.40}$$

The previous result concerning the existence of solutions indicates that if $\mathbf{f}_m$ satisfies these equations, then $\mathbf{f}'_m = \mathbf{f}_m + \mathbf{v}$ also satisfies these equations where $\mathbf{v}$ is any vector in the null space of $\mathbf{A}$ as defined above. By substituting $\mathbf{f}'_m$ into Eq. (4.40), the following equation is obtained:

$$\mathbf{a}'_r = -\mathbf{\Lambda}_m^{-1}\mathbf{f}'_m + \ddot{\mathbf{x}}_{m_{open}}, \tag{4.41}$$

$$= -\mathbf{\Lambda}_m^{-1}\left(\mathbf{f}_m + \mathbf{v}\right) + \ddot{\mathbf{x}}_{m_{open}}. \tag{4.42}$$

Since $\mathbf{\Lambda}_m^{-1}\mathbf{v} = \mathbf{0}$, then

$$\mathbf{a}'_r = -\mathbf{\Lambda}_m^{-1}\mathbf{f}_m + \ddot{\mathbf{x}}_{m_{open}}, \tag{4.43}$$

$$= \mathbf{a}_r. \tag{4.44}$$

Therefore, the reference member acceleration has not changed. To find the associated force at the tip of the other singular chain, $\mathbf{f}'_{m-1}$, this result is substituted into Eq. (4.38) which yields the following equalities:

$$\mathbf{a}_r = \mathbf{\Lambda}_*^{-1}\left(\mathbf{f}_* + \mathbf{f}_{m-1} + \mathbf{f}_m\right) = \mathbf{\Lambda}_*^{-1}\left(\mathbf{f}_* + \mathbf{f}'_{m-1} + \mathbf{f}_m + \mathbf{v}\right), \tag{4.45}$$

which implies that

$$\mathbf{f}'_{m-1} = \mathbf{f}_{m-1} - \mathbf{v}. \tag{4.46}$$

This result shows that any components of force added to the tip of one singular manipulator in the "direction" of the singularity, are compensated for by equal and opposite components of force at the tip of the other singular manipulator, and the acceleration of the reference member is still unique.

4.6 Summary and Conclusions

A new approach to the solution of the dynamics for the simulation problem has been presented in this chapter for a multiple-manipulator system grasping a common object where an arbitrary number of manipulators are in singular configurations. Disregarding an algorithm to compute open chain dynamics such as the one presented in Chapter 2, the efficient algorithm presented for computing the tip forces and accelerations required in this problem has a computational complexity of $O(s^3)$ where s is the number of the manipulators that are in singular configurations. This is a significant improvement over previous results in which the complexity was $O(m^3)$.

With respect to the algorithms presented in Chapter 2, the simpler set of equations, Eqs. (2.7) and (2.8), were used as the basis for for the algorithm developed in this chapter. It should be emphasized that a version of the resulting algorithms to match the efficient procedure developed in Section 2.3 can be obtained by performing the same procedures presented in this chapter but starting with Eqs. (2.16) and (2.20) in place of Eqs. (2.7) and (2.8).

Several special cases were also examined that apply to systems that are used extensively in research. Since the presence of singularities in a system of manipulators that are grasping a common object is undesirable, the cases with only one

or two singularities were also emphasized. In addition, the case for a dual-arm system was also closely examined, and a robust solution procedure was found that required fewer total computations than previous algorithms which did not take singular chains into account. This is a particularly useful result because systems such as these are receiving more research attention recently.

In the context of parallelization of the simulation algorithm that was presented in Chapter 3, the effort to reduce the amount of extra computation performed by these methods becomes even more important. Remember that the RMA computation in the previous algorithm is a computation that must still be performed serially (or redundantly in parallel), and has a significant negative effect on the resulting speedup that can be achieved even under ideal conditions. It is very important to minimize this serial computation even though it may be a small fraction of the computation that is parallelizable. Due to the nature of the computations when chains are singular, tip forces exerted on singular chains must also be computed serially along with the reference member computation. The results in this chapter describe a very efficient method for performing these computations.

CHAPTER V

Efficient Simulation of Multilegged Vehicles

5.1 Introduction

In the previous three chapters, the simulation of multiple manipulator systems categorized as Type 2 systems [13] has been discussed. Efficient algorithms, parallelization, and the problems of singularities have all been addressed. In this chapter, dynamic simulation of Type 1 systems is examined. This category of simple multiple chain mechanisms is characterized by a single central body, also called a reference member, which acts as the mobile base for one or more chains. Legged vehicles, such as the Adaptive Suspension Vehicle [12], are the typical example of Type 1 systems. Whereas, Type 2 systems can vary the topology of the system by making and breaking contact between the chains and the reference member, Type 1 systems maintain this connection and make and break contact with the ground at the opposite end of the chains. The purpose of this chapter is to develop an efficient algorithm for the computation of the dynamics for such systems.

One method to accomplish this task was presented by Oh and Orin [4]. It is based on a method presented in [14] for computing the joint-space inertia matrix of a chain. It is somewhat less efficient than Walker and Orin's Composite Rigid

Body (CRB) method (also in [14]) and yet has the same $O(N^3)$ complexity for a serial chain. Oh and Orin used this approach to develop a method based on equation augmentation using the constraint force equations arising from the closed kinematic loops. Since the size of this system grows very large and the algorithm has a complexity of $O(m^3N^3)$, it is computationally costly and the approach was abandoned in favor of more efficient methods.

With legged vehicles, however, contacts with the ground may be modeled differently from the hard constraints imposed in a multiple manipulator system. It is more realistic to model ground compliance with spring/damper systems as Shih, Frank, and Ravani did in [5] thus imposing soft constraints. Differences in terrain conditions could then be easily modeled by differences in spring and damper constants or even simple nonlinear equations. Shih, Frank and Ravani's work actually goes one step further by modeling the compliant constraints between adjacent links so that forces between bodies is a function of the state only. While this simplifies the dynamics by decoupling every rigid body in the system, this approach requires a larger number of system states and can cause numerical problems if joints are to be modeled very stiffly in the constrained directions.

Freeman and Orin develop an improved modeling technique which is a compromise between hard constraints and a completely decoupled system with soft constraints by introducing compliance at the ground contacts only [7]. As a result, the kinematic loops are broken leaving a tree-structured, open chain system. An advantage of this approach is that singularities do not affect open chain computations

and are no longer an issue in the simulation of this system. With this Decoupled Tree-Structure (DTS) approach, Freeman and Orin go on to describe a very efficient $O(mN)$ simulation algorithm for legged vehicles that is based on Featherstone's $O(N)$ Articulated Body (AB) algorithm [15].

The results in Chapter 2 showed that an $O(mN^3)$ algorithm using the efficient, open chain algorithms from Appendix B was more efficient than corresponding $O(mN)$ algorithms when $N \leq 21$ for the simulation of multiple, closed chain manipulators. In this chapter, these algorithms are also used as the basis for a new approach to legged vehicle simulation. The resulting algorithm is called the Composite Rigid Body/Decoupled Tree-Structure (CRB/DTS) Method and also has a computational complexity of $O(mN^3)$.

In the next section, the formulation of the dynamic equations of motion for this system are developed. This requires that multiple chain/mobile base algorithms for the computation of the joint space inertia matrix, bias forces (using the inverse dynamics computation), and accelerations be developed. Section 5.3 presents the details of these three parts. Also included in this discussion is the efficient kinematic modeling and extensions of the efficient algorithms from Appendix B for the tree-structure topology of legged vehicles. Then, Section 5.4 details the computational requirements of the resulting $O(mN^3)$ algorithm. For the typical legged vehicle which has three degrees of freedom in each leg, the CRB/DTS Method requires less computation than any of the previous simulation algorithms developed for legged vehicles including the more popular $O(mN)$ approaches using the AB methods.

5.2 Dynamics Formulation

In this section, dynamic equations of motion for legged vehicles are developed based on the CRB approach, and Figure 16 illustrates the model that is used. It contains m serial chains that correspond to the legs of the vehicle which make and break contact with the ground. When in contact spring/damper systems can be used to compute the state-dependent force, $\mathbf{f}_k$, on the last link of each leg. At the other end of each chain, the first joint of the chain connects the leg to the body of the vehicle which is also referred to as the reference member, r, of the system. Finally, a fictitious (virtual) six degree-of-freedom joint, called the *body joint* connects the reference member to the inertial coordinate frame. If an external control force can be applied to the reference member, not by the legs or gravity but through thrusters attached to the body for example, this force can be thought of as the input force of the virtual body joint. In legged vehicles on land, however, this joint is generally unpowered.

This development begins with the joint space equations of motion for a serial chain with a fixed base and is repeated here for reference:

$$\boldsymbol{\tau}_k = \mathbf{H}_k(\mathbf{q}_k)\ddot{\mathbf{q}}_k + \mathbf{C}_k(\mathbf{q}_k, \dot{\mathbf{q}}_k) + \mathbf{G}_k(\mathbf{q}_k) + \mathbf{J}_k^T(\mathbf{q}_k)\mathbf{f}_k, \tag{5.1}$$

where the various terms have been defined in Chapter 2. This would correspond to the equations of motion for leg k provided the body of the vehicle, acting as the base of the chain, is motionless. When working with legged vehicles, however, the body of the vehicle acts as a mobile base whose motion is unknown. As a result, augmented chains with fixed bases could be formed which contain the links of a leg

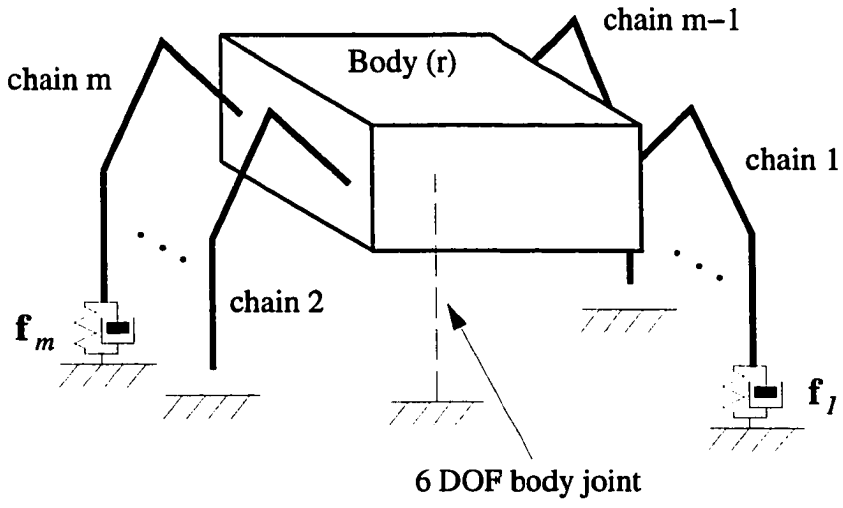


Figure 16: DTS model for a legged vehicle with a body joint.

and an additional link (the body) with a six degree-of-freedom joint (the body joint) connecting it to the inertial coordinate frame.

To dynamically model each one of these augmented chains, additional terms accounting for the base's position (linear and angular), velocity and acceleration must be added to the basic equation above. First an extra term relating its spatial acceleration, $\mathbf{a}_r$, to leg k's joint torques must be included. For a given state, this relationship is linear and is defined by an $N \times 6$ *coupling inertia* matrix, $\mathbf{P}_k$, which is a function of the joint positions in the leg. Second, the foot contact force, if present, along with coriolis, centrifugal, and gravitational forces in the leg are grouped into a bias force vector, $\mathbf{b}_k$. Unlike, the derivation in Chapter 2, however, this vector is now a function of the body's state (position and velocity) as well as the chain's joint

positions and velocities. This leads to the following equations of motion:

$$\tau_k = \mathbf{H}_k\ddot{\mathbf{q}}_k + \mathbf{P}_k\mathbf{a}_r + \mathbf{b}_k. \tag{5.2}$$

Repeated for all m legs in the system, a system of mN equations is obtained containing the $(mN+6)$ unknown accelerations ($\ddot{\mathbf{q}}_1, \ldots, \ddot{\mathbf{q}}_m$, and $\mathbf{a}_r$).

As in Eq. (2.8) of Chapter 2, a force balance equation for the reference member is required to complete the system of equations. This equates the forces exerted *onto* the body to its acceleration. Written in the body fixed coordinate system, this equation can be expressed as follows:

$$\mathbf{f}_r + \mathbf{f}_r^g - \sum_{k=1}^{m} {}^r\mathbf{f}_k = \mathbf{I}_r\mathbf{a}_r + \bar{\mathbf{f}}_r, \tag{5.3}$$

where $\mathbf{f}_r$ is an external control force (if present) exerted onto the body, and ${}^r\mathbf{f}_k$ is the force exerted *by* the body onto the first link of each leg k. The remaining terms are the same as those used in Eq. (2.8).

Expressions for the body-to-chain forces, ${}^r\mathbf{f}_k$, need to be derived as a function of the unknown accelerations in the system. These forces are linearly related to the accelerations in the system. Therefore, the forces for each leg are decomposed into terms related to the accelerations of its own joints and that of the vehicle body, and an additional force, ${}^r\mathbf{f}'_k$, containing gravity, foot contact force, and chain velocity-dependent terms. This body-to-chain force can be written as follows:

$${}^r\mathbf{f}_k = \mathbf{Q}_k\ddot{\mathbf{q}}_k + {}^r\mathbf{K}_k\mathbf{a}_r + {}^r\mathbf{f}'_k, \tag{5.4}$$

where $\mathbf{Q}_k$ is the transpose of the inertia coupling matrix, $\mathbf{P}_k$ from Eq. (5.2) (verified below), and ${}^r\mathbf{K}_k$ is the spatial CRB inertia for all of the links in leg k and transformed

to the body's coordinate system. Note that ${}^r\mathbf{f}_k$ is not directly a function of the motion in other chains. Instead, the effects of other chains are coupled through the reference member acceleration and Eq. (5.3).

Substituting Eq. (5.4) into Eq. (5.3) and rearranging the terms to put it into a form similar to Eq. (5.2) leads to the following equation for the body of the vehicle:

$$\mathbf{f}_r = \sum_{k=1}^{m} \mathbf{Q}_k \ddot{\mathbf{q}}_k + \left(\mathbf{I}_r + \sum_{k=1}^{m} {}^r\mathbf{K}_k \right) \mathbf{a}_r + \left(\mathbf{f}'_r + \sum_{k=1}^{m} {}^r\mathbf{f}'_k \right), \tag{5.5}$$

where $\mathbf{f}'_r$ combines gravity and bias forces associated with the reference member as follows:

$$\mathbf{f}'_r = \bar{\mathbf{f}}_r - \mathbf{f}^g_r. \tag{5.6}$$

Note that in comparison with Eq. (5.2), the spatial force, $\mathbf{f}_r$, corresponds to $\boldsymbol{\tau}_r$ which can be thought of as the joint forces and moments exerted by the body joint on the reference member. This term could correspond to thruster forces exerted on the body. Since most land-based legged vehicles have no mechanisms to apply forces directly to the body, the "body joint" is said to be unpowered and this term is zero.

Combining Eqs. (5.2) and (5.5) results in the following system of equations describing the motion in the system:

$$\begin{bmatrix} \boldsymbol{\tau}_1 \\ \boldsymbol{\tau}_2 \\ \vdots \\ \boldsymbol{\tau}_m \\ \mathbf{f}_r \end{bmatrix} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{P}_1 \\ \mathbf{0} & \mathbf{H}_2 & & \mathbf{0} & \mathbf{P}_2 \\ \vdots & & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & & \mathbf{H}_m & \mathbf{P}_m \\ \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_m & \mathbf{K}_r \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}}_1 \\ \ddot{\mathbf{q}}_2 \\ \vdots \\ \ddot{\mathbf{q}}_m \\ \mathbf{a}_r \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_m \\ \mathbf{f}'_r + \sum_{k=1}^{m} {}^r\mathbf{f}'_k \end{bmatrix}, \tag{5.7}$$

where

$$\mathbf{K}_r = \mathbf{I}_r + \sum_{k=1}^{m} {}^r\mathbf{K}_k,$$

which is the CRB inertia of the entire system defined with respect to the body-fixed coordinate system. The $(mN+6)\times(mN+6)$ block matrix in Eq. (5.7) corresponds to the joint-space inertia matrix for the entire legged vehicle including the fictitious "body joint". Just as the joint space inertia matrices for each leg, $\mathbf{H}_k$, are symmetric and positive definite, so is the entire matrix. This verifies that $\mathbf{Q}_k = \mathbf{P}_k^T$ and the same inertia coupling matrix is used in both Eqs. (5.2) and (5.4).

5.3 The CRB/DTS Algorithm

In this section, the efficient algorithm to compute the desired accelerations is presented. The approach is the same as the CRB method in [14] where elements of a bias force vector and inertia matrix are computed, and then the resulting system of equations is solved for the accelerations. This approach used serial, open chain algorithms such as those found in Appendix B. Some modifications to these routines must be made in order to compute the needed terms in the simulation of a legged vehicle. First, efficient kinematic modeling for the tree-structured system is presented in Section 5.3.1. Based on this new modeling, the inverse dynamics and inertia matrix algorithms are then modified in Sections 5.3.2 and 5.3.3, respectively. These routines compute the elements of the $(mN+6)$-vector of bias forces and the non-zero blocks of the matrix in Eq. (5.7). Finally, an efficient method to solve the resulting system of equations for the accelerations is given in Section 5.3.4.

5.3.1 Kinematic Modeling of Legged Vehicles

The serial chain algorithms in Appendix B assume that the base of the serial chain is fixed, and that its coordinate system can be arbitrarily placed. With this freedom, modified Denavit-Hartenberg (MDH) parameters can be specified between adjacent coordinate systems including the base and the first link. The computational requirements would be lower as a result. With a legged vehicle, the body can be considered the mobile base for each serial chain (leg) in the system. In general, MDH parameters cannot be specified from this single coordinate system to the one attached to the first link of each chain.

The answer to this problem is to specify an additional coordinate system for each chain that is attached to the reference member and acts as its base coordinate system. This is illustrated in Figure 17 for one such leg k. The reference member coordinate system is specified with the r subscript and the base coordinate system for chain k is specified with the $k,0$ subscripts. However, the placement of the base coordinate system is not arbitrary, and follows guidelines for kinematic modeling of tree-structured mechanisms that were developed by Khalil and Kleinfinger in [44].

First, the $\hat{\boldsymbol{x}}_{k,0}$ axis is placed along the common normal between $\hat{\boldsymbol{z}}_r$ and $\hat{\boldsymbol{z}}_{k,1}$, and $\hat{\boldsymbol{z}}_{k,0}$ is placed collinear with $\hat{\boldsymbol{z}}_r$ and pointing in the same direction. This is repeated for all of the legs in the system, and results in a simplified transformation between the reference member coordinate system and all of the base coordinate systems. These transformations are planar screws along/about the z-axis and can be specified by two parameters which are the same as the second two used in the

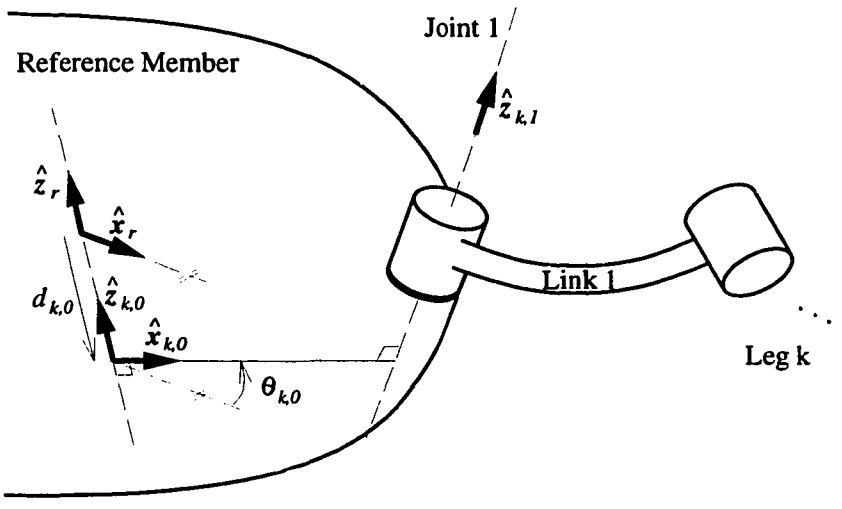


Figure 17: Efficient coordinate system assignments and transformations between the reference member and each chain.

MDH notation presented in Appendix A:

$d_{k,0}$ = the perpendicular distance along $\hat{z}_{k,0}$ from $\hat{x}_r$ to $\hat{x}_{k,0}$,

$\theta_{k,0}$ = the angle about $\hat{z}_{k,0}$ from $\hat{x}_r$ to $\hat{x}_{k,0}$.

A full set of MDH parameters are then specified between the base and the first link of each chain.

5.3.2 Inverse Dynamics

To compute the $(mN+6)$ elements of the bias vector, an inverse dynamics algorithm for the tree-structured topology of legged vehicles is executed with the system's accelerations set to zero. The resulting joint torques/forces and reference member force that are computed are the desired elements [see Eq. (5.7)]. The inverse dynamics algorithm for a single chain mechanism from Appendix B is used as the basis for

the new algorithm developed in this section.

To begin, the Forward Kinematics recursion is examined. Its purpose is to compute each rigid body's angular velocity, spatial acceleration, and bias force. In the appendix, the angular velocity of the base is set to zero because it is fixed. In this development the base is the mobile reference member, and therefore the angular velocity is specified by the state of the reference member. Due to the addition of separate reference member and chain base coordinates in the previous discussion, the angular velocity of the base is then obtained as follows:

$$\omega_{k,0} = {}^{0}\boldsymbol{R}_{k,r}\,\omega_{r}, \tag{5.8}$$

where ${}^{0}\boldsymbol{R}_{k,r}$ is the matrix representing a z-planar rotation about the angle, $\theta_{k,0}$.

Modification of the acceleration computation is more involved. When the inverse dynamics algorithm is used to compute bias terms, the joint accelerations are set to zero. Correspondingly, the acceleration of the reference member is set to zero. At the same time, the base acceleration is biased by negative gravitational acceleration. This implies that the starting condition for the spatial acceleration recursion in the new algorithm is given as follows:

$$\mathbf{a}_{r} = \begin{bmatrix} \mathbf{0} \\ -g\ {}^{r}\boldsymbol{R}_{e}\,\hat{\mathbf{z}}_{e} \end{bmatrix}, \tag{5.9}$$

where g is the magnitude of the gravitational acceleration (e.g., 9.81 m/s), ${}^{r}\boldsymbol{R}_{e}$ is the rotation matrix between the inertial and reference member coordinate systems, and $\hat{\mathbf{z}}_{e}$ is the inertial coordinate system's z-axis assumed to be pointing "down." To obtain, the resulting acceleration of the base coordinate system of each chain, the

transformation for accelerations shown in Eq. (2.13) is used:

$$\mathbf{a}_{k,0} = {}^{0}\mathbf{X}_{k,r}\,\mathbf{a}_r + \begin{bmatrix} \mathbf{0} \\ {}^{0}\boldsymbol{R}_{k,r}\left[\boldsymbol{\omega}_r \times \left(\boldsymbol{\omega}_r \times {}^{r}\boldsymbol{p}_{k,0}\right)\right] \end{bmatrix}, \tag{5.10}$$

where ${}^{0}\mathbf{X}_{k,r}$ is the spatial screw transformation defined by $d_{k,0}$ and $\theta_{k,0}$, and ${}^{r}\boldsymbol{p}_{k,0}$ is the position of the base coordinate system with respect to the reference member coordinate system and is given by $[0\ 0\ d_{k,0}]^T$.

The final modification to the Forward Kinematics recursion, is the addition of the computation of the gravity and bias forces for the reference member, $\mathbf{f}'_r$. It is the same as the computation for a link in the serial chain algorithm and is written as follows:

$$\mathbf{f}'_r = \mathbf{I}_r\mathbf{a}_r + \begin{bmatrix} \boldsymbol{\omega}_r \times \bar{\boldsymbol{I}}_r\boldsymbol{\omega}_r \\ \boldsymbol{\omega}_r \times (\boldsymbol{\omega}_r \times \boldsymbol{h}_r) \end{bmatrix}, \tag{5.11}$$

where $\bar{\boldsymbol{I}}_r$ and $\boldsymbol{h}_r$ are the moment of inertia and first moment of mass for the reference member as defined in Chapter 1. Since $\mathbf{a}_r$ has been biased by a gravitational acceleration term, the first term on the right hand side of this equation computes the desired gravitational force.

The purpose of the Backward Dynamics recursion, is to compute the spatial force exerted on each link by its inboard neighbor in the chain. The first iteration requires the force exerted by the foot onto the ground, ${}^{N}\mathbf{f}_{k,N+1}$, which can be any function of the state of the system (positions and velocities) as specified by the DTS approach. An example of a compliant foot force computation is given in [7]. These forces are projected onto the joint axis to determine the joint torque. When this algorithm is used as a bias computation, accelerations have been set to zero, the force on the link

consists of only state-dependent terms, and the joint torque computed for link i of leg k is $[\mathbf{b}_k]_i$, the ith element of $\mathbf{b}_k$. This computation remains largely unchanged from the appendix for the links of each leg. To compute the desired bias forces exerted by the reference member onto each chain, ${}^r\mathbf{f}'_k$, the force exerted onto the first link must be transformed to the reference member coordinate system. This involves two spatial transformations as follows:

$$ {}^r\mathbf{f}'_k = {}^0\mathbf{X}^T_{k,r}\,{}^1\mathbf{X}^T_{k,0}\mathbf{f}_{k,1}. \tag{5.12} $$

The Bias algorithm can now be written as shown in Table 10. Note that the transformations in the Backward Dynamics recursion have been slightly rearranged.

5.3.3 Inertia Matrix

In this section, the algorithm to compute the non-zero blocks of the inertia matrix in Eq. (5.7) is developed. The single chain algorithm in Table 27 of Appendix B is used as the basis for this development. To begin, a physical interpretation is given to the algorithm in Table 27. Then, appropriate extensions to accommodate the topology of legged vehicles are added to the algorithm.

The first equation of Table 27 computes the ith CRB inertia, $\mathbf{K}_i$, which is the spatial inertia of links i through N assuming the joints are locked in the position specified by the current state. A unit acceleration is applied to joint i and the resulting force on link i is given by:

$$ \mathbf{f}_i = \mathbf{K}_i\boldsymbol{\phi}_i. \tag{5.13} $$

Projecting this force onto joint i yields element (i,i) of the joint space inertia matrix

Table 10: Bias algorithm for the CRB/DTS Method.

Given: ${}^{r}\boldsymbol{R}_{e}$, $\boldsymbol{\omega}_{r}$, $\dot{\boldsymbol{\omega}}_{r}$.

$$\mathbf{a}_{r} = \begin{bmatrix} \mathbf{0} \\ -g\ {}^{r}\boldsymbol{R}_{e}\,\hat{\mathbf{z}}_{e} \end{bmatrix}$$

$$\mathbf{f}'_{r} = \mathbf{I}_{r}\mathbf{a}_{r} + \begin{bmatrix} \boldsymbol{\omega}_{r} \times \bar{\boldsymbol{I}}_{r}\boldsymbol{\omega}_{r} \\ \boldsymbol{\omega}_{r} \times (\boldsymbol{\omega}_{r} \times \boldsymbol{h}_{r}) \end{bmatrix}$$

for all $k = 1, \ldots, m$. Given: ${}^{0}\boldsymbol{R}_{k,r}$.

$$\boldsymbol{\omega}_{k,0} = {}^{0}\boldsymbol{R}_{k,r}\,\boldsymbol{\omega}_{r}$$

$$\mathbf{a}_{k,0} = {}^{0}\mathbf{X}_{k,r}\,\mathbf{a}_{r} + \begin{bmatrix} \mathbf{0} \\ {}^{0}\boldsymbol{R}_{k,r}\left[\boldsymbol{\omega}_{r} \times \left(\boldsymbol{\omega}_{r} \times {}^{r}\boldsymbol{p}_{k,0}\right)\right] \end{bmatrix}$$

for $i = 1, \ldots, N$. Given: ${}^{i}\boldsymbol{R}_{k,i-1}$, ${}^{i-1}\boldsymbol{p}_{k,i}$, $\dot{q}_{k,i}$.

$$\boldsymbol{\omega}_{k,i} = {}^{i}\boldsymbol{R}_{k,i-1}\boldsymbol{\omega}_{k,i-1} + \sigma_{k,i}\dot{q}_{k,i}\hat{\mathbf{z}}_{k,i}$$

$$\mathbf{a}_{k,i} = {}^{i}\mathbf{X}_{k,i-1}\mathbf{a}_{k,i-1} + \begin{bmatrix} \mathbf{0} \\ {}^{i}\boldsymbol{R}_{k,i-1}\left[\boldsymbol{\omega}_{k,i-1} \times \left(\boldsymbol{\omega}_{k,i-1} \times {}^{i-1}\boldsymbol{p}_{k,i}\right)\right] \end{bmatrix} + \begin{bmatrix} \sigma_{k,i}\left(\boldsymbol{\omega}_{k,i} \times \dot{q}_{k,i}\hat{\mathbf{z}}_{k,i}\right) \\ \bar{\sigma}_{k,i}\left(2\boldsymbol{\omega}_{k,i} \times \dot{q}_{k,i}\hat{\mathbf{z}}_{k,i}\right) \end{bmatrix}$$

$$\mathbf{f}'_{k,i} = \mathbf{I}_{k,i}\mathbf{a}_{k,i} + \begin{bmatrix} \boldsymbol{\omega}_{k,i} \times \bar{\boldsymbol{I}}_{k,i}\boldsymbol{\omega}_{k,i} \\ \boldsymbol{\omega}_{k,i} \times (\boldsymbol{\omega}_{k,i} \times \boldsymbol{h}_{k,i}) \end{bmatrix}$$

end for i.

for $i = N, \ldots, 1$. Given: ${}^{N}\mathbf{f}_{k,N+1}$ (external tip force).

$$\mathbf{f}_{k,i} = \mathbf{f}'_{k,i} + {}^{i}\mathbf{f}_{k,i+1}$$

$$[\mathbf{b}_{k}]_{i} = \boldsymbol{\phi}_{k,i}^{T}\mathbf{f}_{k,i}$$

$${}^{i-1}\mathbf{f}_{k,i} = {}^{i}\mathbf{X}_{k,i-1}^{T}\mathbf{f}_{k,i}$$

end for i.

$${}^{r}\mathbf{f}'_{k} = {}^{0}\mathbf{X}_{k,r}^{T}\,{}^{0}\mathbf{f}_{k,1}$$

end for all k.

for the chain:

$$[\mathbf{H}]_{i,i} = \phi_i^T \mathbf{f}_i. \tag{5.14}$$

This spatial force is transformed to the inboard links and projected to their joint axes to obtain the off-diagonal elements of $\mathbf{H}$ as follows:

$${}^{j}\mathbf{f}_i = {}^{j+1}\mathbf{X}_j^T \, {}^{j+1}\mathbf{f}_i \tag{5.15}$$

$$[\mathbf{H}]_{i,j} = \phi_j^T {}^{j}\mathbf{f}_i. \tag{5.16}$$

This step ends at $j = 1$ for a single serial chain with a fixed base. This computation is repeated for each serial chain in the legged vehicle, and is sufficient to produce the elements of the desired $\mathbf{H}_k$ matrices.

To obtain the ith column of the inertia coupling matrix, $\mathbf{Q}_k$, the force on link 1 of leg k as computed above (${}^{1}\mathbf{f}_{k,i}$ for leg k), must also be projected onto the body joint. This is accomplished with two spatial force transformations to the reference member coordinate system:

$${}^{r}\mathbf{f}_{k,i} = {}^{0}\mathbf{X}_{k,r}^T \, {}^{1}\mathbf{X}_{k,0}^T \, {}^{1}\mathbf{f}_{k,i}, \tag{5.17}$$

and a projection onto the body joint as follows:

$$[\mathbf{Q}_k]_i = \phi_r \, {}^{r}\mathbf{f}_{k,i}. \tag{5.18}$$

Since the body joint has six degrees of freedom, ϕ_r is actually the 6×6 identity matrix; therefore, the ith column of $\mathbf{Q}_k$ is ${}^{r}\mathbf{f}_{k,i}$.

The last matrix to be computed is the CRB inertia of the entire system, $\mathbf{K}_r$. The serial chain algorithm computes, $\mathbf{K}_{k,1}$, the CRB inertia of links 1 through N for leg k.

First, these must be expressed with respect to a common coordinate system, the reference member coordinate system. Two successive congruence transformations are required as follows:

$$ {}^{r}\mathbf{K}_k = {}^{0}\mathbf{X}_{k,r}^{T}\left({}^{1}\mathbf{X}_{k,0}^{T}\mathbf{K}_{k,1}{}^{1}\mathbf{X}_{k,0}\right){}^{0}\mathbf{X}_{k,r}. \tag{5.19} $$

The CRB inertia for the system, $\mathbf{K}_r$, is the sum of these and the spatial inertia of the reference member, $\mathbf{I}_r$. Adding these steps to the algorithm in Table 27 and rearranging some of the transformation operations, the desired algorithm is obtained as shown in Table 11.

5.3.4 Computation of Accelerations

With the results from the previous two sections and the simulation's input, joint torques and forces, Eq. (5.7) becomes an $(mN+6)\times(mN+6)$ linear system of equations with joint and reference member accelerations as unknown quantities. The accelerations could be determined by employing a standard linear system solver for Eq. (5.7) which results in an $O(m^3N^3)$ algorithm as was done in [4]. In this section, the structure of the system matrix is taken into account to develop an alternative method and significantly reduce the complexity and the computational requirements of this step.

First Eq. (5.7) is rewritten noting that $\mathbf{P}_k = \mathbf{Q}_k^T$:

$$ \begin{bmatrix} \mathbf{H}_1 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{Q}_1^T \\ \mathbf{0} & \mathbf{H}_2 & & \mathbf{0} & \mathbf{Q}_2^T \\ \vdots & & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & & \mathbf{H}_m & \mathbf{Q}_m^T \\ \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_m & \mathbf{K}_r \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}}_1 \\ \ddot{\mathbf{q}}_2 \\ \vdots \\ \ddot{\mathbf{q}}_m \\ \mathbf{a}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_1 - \mathbf{b}_1 \\ \boldsymbol{\tau}_2 - \mathbf{b}_2 \\ \vdots \\ \boldsymbol{\tau}_m - \mathbf{b}_m \\ \mathbf{f}_r - \mathbf{f}_r' - \sum_{k=1}^{m} {}^{r}\mathbf{f}_k' \end{bmatrix}. \tag{5.20} $$

Table 11: Inertia matrix algorithm for the CRB/DTS Method.

for all $k = 1, \ldots, m$. Given: ${}^{N}\mathbf{K}_{k,N+1} = \mathbf{0}$.

for $i = N, \ldots, 1$.

$$\begin{aligned}
\mathbf{K}_{k,i} &= \mathbf{I}_{k,i} + {}^{i}\mathbf{K}_{k,i+1} \\
\mathbf{f}_{k,i} &= \mathbf{K}_{k,i}\boldsymbol{\phi}_{k,i} \\
[\mathbf{H}_k]_{i,i} &= \boldsymbol{\phi}_{k,i}^{T}\mathbf{f}_{k,i} \\
{}^{i-1}\mathbf{K}_{k,i} &= {}^{i}\mathbf{X}_{k,i-1}^{T}\mathbf{K}_{k,i}{}^{i}\mathbf{X}_{k,i-1} \\
{}^{i-1}\mathbf{f}_{k,i} &= {}^{i}\mathbf{X}_{k,i-1}^{T}\mathbf{f}_{k,i}
\end{aligned}$$

for $j = i - 1, \ldots, 1$.

$$\begin{aligned}
[\mathbf{H}_k]_{i,j} &= \boldsymbol{\phi}_{k,j}^{T}{}^{j}\mathbf{f}_{k,i} \\
{}^{j-1}\mathbf{f}_{k,i} &= {}^{j}\mathbf{X}_{k,j-1}^{T}{}^{j}\mathbf{f}_{k,i}
\end{aligned}$$

end for j.

$$[\mathbf{Q}_k]_i = {}^{0}\mathbf{X}_{k,r}^{T}{}^{0}\mathbf{f}_{k,i}$$

end for i.

$${}^{r}\mathbf{K}_k = {}^{0}\mathbf{X}_{k,r}^{T}{}^{0}\mathbf{K}_{k,1}{}^{0}\mathbf{X}_{k,r}$$

end for all k.

$$\mathbf{K}_r = \mathbf{I}_r + \sum_{k=1}^{m} {}^{r}\mathbf{K}_k$$

Upon examination of this equation, the sparse structure of the matrix is found to be the same as the one that resulted from modeling the dynamics for multiple manipulator systems using an operational space formulation in Eq. (4.3) of the previous chapter. In that problem, block elimination techniques were developed to achieve more efficient and robust methods for solving the system of equations. The same technique can be applied to this problem to achieve substantial improvements over a standard linear system solution method.

The first step in this technique is to invert the joint space inertia matrices of the chains, $\mathbf{H}_k$, and use them to block-eliminate the coupling inertia matrices $\mathbf{Q}_k$ along the bottom row. This is accomplished by subtracting from the bottom row, $\mathbf{Q}_k\mathbf{H}_k^{-1}$ multiplied times row k, for all $k = 1, \ldots, m$. The resulting system of equations appears as follows:

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{Q}_1^T \\ \mathbf{0} & \mathbf{H}_2 & & \mathbf{0} & \mathbf{Q}_2^T \\ \vdots & & \ddots & & \vdots \\ \mathbf{0} & \mathbf{0} & & \mathbf{H}_m & \mathbf{Q}_m^T \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{K}_r^* \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}}_1 \\ \ddot{\mathbf{q}}_2 \\ \vdots \\ \ddot{\mathbf{q}}_m \\ \mathbf{a}_r \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau}_1 - \mathbf{b}_1 \\ \boldsymbol{\tau}_2 - \mathbf{b}_2 \\ \vdots \\ \boldsymbol{\tau}_m - \mathbf{b}_m \\ \mathbf{f}_r^* \end{bmatrix}, \tag{5.21}$$

where

$$\mathbf{K}_r^* = \mathbf{K}_r - \sum_{k=1}^{m} \mathbf{Q}_k\mathbf{H}_k^{-1}\mathbf{Q}_k^T, \tag{5.22}$$

$$\mathbf{f}_r^* = \mathbf{f}_r - \mathbf{f}_r' - \sum_{k=1}^{m} \left[{}^r\mathbf{f}_k' + \mathbf{Q}_k\mathbf{H}_k^{-1}\left(\boldsymbol{\tau}_k - \mathbf{b}_k\right) \right]. \tag{5.23}$$

In Chapter 4, this step involved the inversion of matrices that could become singular. In this development, however, the joint space inertia matrices, $\mathbf{H}_k$, are always nonsingular so the inverses always exist.

The accelerations in Eq. (5.21) can now be determined. The acceleration of the reference member can be computed as follows:

$$\mathbf{a}_r = [\mathbf{K}_r^*]^{-1} \mathbf{f}_r^*, \tag{5.24}$$

where $\mathbf{K}_r^*$ is invertible because the original matrix in Eq. (5.7) is positive definite which implies that the matrices along the diagonal of the reduced system in Eq. (5.21) are nonsingular. With $\mathbf{a}_r$ known, back substitution in Eq. (5.21) decouples the rest of the system of unknowns, and the chain's joint accelerations are computed as follows:

$$\ddot{\mathbf{q}}_k = \mathbf{H}_k^{-1} \left[(\boldsymbol{\tau}_k - \mathbf{b}_k) - \mathbf{Q}_k^T \mathbf{a}_r \right], \qquad \text{for all } k = 1, \ldots, m, \tag{5.25}$$

which completes the computation of the dynamics. Again, a numerical integration routine is needed in the simulation of the system in order to update the system's state based on these new accelerations.

5.4 Computational Requirements

The final task of this chapter is to determine the computational requirements of the algorithm described in the previous sections. For discussions of the inverse dynamics and inertia matrix algorithms, the additions to the algorithms from Appendix B are discussed in detail while minor changes made to the steps in the algorithms taken from the appendix are summarized. Also, the computational requirements will focus on legs with revolute joints. A similar analysis may be made for other types of joints.

Inverse Dynamics

This discussion begins with the inverse dynamics routine. Assuming the orientation of the reference member is specified with Z-Y-X Euler angles, the computation of the gravitational acceleration for $\mathbf{a}_r$ requires [4M, 0A] as was found in Chapter 2. The computational requirements for $\mathbf{f}'_r$ depend on where the reference member's coordinate system is placed. In this case, it is placed coincident with its principal axes, the inertia matrices, $\bar{\boldsymbol{I}}_r$ and $\mathbf{I}_r$, are diagonal and the first moment of mass, $\boldsymbol{h}_r$ is zero. Since $\dot{\boldsymbol{\omega}}_r$ is also zero, this step requires [12M, 3A].

To transform kinematic quantities from the reference member to the base coordinate system for each leg, translation along and rotation about the z-axis is required. With $\boldsymbol{\omega}_r$ given, the z-planar rotation to obtain $\boldsymbol{\omega}_{k,0}$ costs [4M, 2A] per leg as discussed in Appendix A. Computation of $\mathbf{a}_{k,0}$ can be reduced to the computation of the linear acceleration portion since angular acceleration is zero. This computation simplifies to:

$$\boldsymbol{a}_{k,0} = {}^0\boldsymbol{R}_{k,r}\left[\boldsymbol{a}_r + \boldsymbol{\omega}_r \times \left(\boldsymbol{\omega}_r \times {}^r\boldsymbol{p}_{k,0}\right)\right]. \tag{5.26}$$

Since only the third element of ${}^r\boldsymbol{p}_{k,0}$ is non-zero, this step requires [10M, 6A] per leg. In this step, $\boldsymbol{\Omega}_{k,0}$ as defined in the appendix, must also be computed. Since the angular velocity used in this computation is zero, only $\tilde{\boldsymbol{\omega}}_{k,0}\tilde{\boldsymbol{\omega}}_{k,0}$ needs to be computed which requires an additional [6M, 3A] per leg.

Once these terms have been computed, the bias computation for each leg in the system can be performed. This involves the equations in the nested loops with the i index, and are much the same as the ones presented in Appendix B but with the

joint accelerations set to zero. In this case with a moving base, many of the vectors that were zero in the appendix are now generally non-zero, with the exception of the angular acceleration at the base coordinate system of each leg. This causes the computational requirements to jump from [$(94N - 114)$M, $(81N - 105)$A] to [$(94N - 10)$M, $(81N - 10)$A] for a leg with N revolute joints.

Finally, the force on the first link of the leg must be transformed to the reference member's coordinate system. The spatial transformation to the leg's base coordinate system, involving a full set of MDH parameters, is included in the inverse dynamics requirements above. So only the z-planar screw is necessary from the base to the reference member system which requires [10M, 6A] per leg. A summary of these computational requirements is given in the first part of Table 12 at the end of this section.

Inertia Matrix

As with the inverse dynamics computation above, a significant amount of the computational requirements for the inertia matrix computation can be ascertained from the algorithm described in Appendix B. In this case, many of the reductions in the number of operations due to unneeded elements of the quantities for the first link cannot be applied. Additional transformations of $\mathbf{K}_{k,1}$ and ${}^{j}\mathbf{f}_{k,1}$ to the leg's base coordinate system have been added to the algorithm, which results in a computational requirement of [$(10N^2 + 39N)$M, $(6N^2 + 51N - 9)$A] per leg.

The additional computation to find each column of $\mathbf{Q}_k$ requires a z-planar screw

transformation requiring [10M, 6A]. This constant planar congruence transformation of the spatial CRB inertia matrix, ${}^0\mathbf{K}_{k,1}$ can be shown to require only [15M, 18A] per leg by transforming the moment of inertia matrix and first moment of mass vector separately and using some results from Appendix A. The summation of $m+1$ spatial inertia matrices to obtain $\mathbf{K}_r$ requires only [0M, $(9m-9)$A]. This addition involves six unique elements from the upper triangular portions of the 3×3 moment of inertia matrices and the three elements of each first moment of mass vector for each chain and the reference member. Since the reference member coordinate system is placed at the principal axes, $\bar{\boldsymbol{I}}_r$ is diagonal and $\boldsymbol{h}_r$ is zero, nine fewer additions are needed for these quantities. The composite mass of the entire system remains constant and can be computed off-line as was done in Appendix B. A summary of the overall cost of the computations to obtain the inertia matrix are listed in the second part of Table 12.

Accelerations

The most efficient method of computing $\mathbf{K}_r^*$ and $\mathbf{f}_r^*$ does not require the inverse of $\mathbf{H}_k$. As discussed in the previous chapter, a root-free Cholesky decomposition of the matrix is computed requiring $[(\frac{1}{6}N^3+\frac{1}{2}N^2-\frac{2}{3}N)\text{M}, (\frac{1}{6}N^3-\frac{1}{6}N)\text{A}]$. The six columns of $\mathbf{H}_k^{-1}\mathbf{Q}_k^T$ can then be determined from back substitutions with each of the six columns of $\mathbf{Q}_k^T$, requiring $[N^2\text{M}, (N^2-N)\text{A}]$ per column [37]. The result is used in matrix-matrix multiplications to compute $\mathbf{K}_r^*$. Its transpose is equal to $\mathbf{Q}_k\mathbf{H}_k^{-1}$ and is also used in matrix-vector multiplications to compute $\mathbf{f}_r^*$.

The acceleration of the reference member can be computed as follows:

$$\mathbf{a}_r = [\mathbf{K}_r^*]^{-1} \mathbf{f}_r^*. \tag{5.27}$$

Since the original matrix in Eq. (5.7) is positive definite, the matrices along the diagonal of the reduced system in Eq. (5.21) are also positive definite. Therefore, root-free Cholesky decomposition and a back substitution step is also the most efficient method of determining $\mathbf{a}_r$. Since the matrix is 6×6, the cost is [86M, 65A]. The chain's joint accelerations are computed as follows:

$$\ddot{\mathbf{q}}_k = \mathbf{H}_k^{-1} \left[(\boldsymbol{\tau}_k - \mathbf{b}_k) - \mathbf{Q}_k^T \mathbf{a}_r \right], \qquad \text{for all } k = 1, \ldots, m. \tag{5.28}$$

Since the Cholesky decomposition of $\mathbf{H}_k$ is determined for the block elimination step, this step requires only matrix-vector multiplies, a vector addition, and a back substitution step. The cost is $[(N^2+6N)\text{M}, (N^2+5N)\text{A}]$. A summary of these steps are listed in the third part of the table.

The last line of Table 12, lists the cost of computing the dynamics of a system with six chains containing three revolute degrees of freedom each at [4440M, 3944A]. This corresponds to a number of experimental legged vehicles including the OSU Hexapod. The alternative algorithm is the DTS algorithm based on the AB (AB/DTS) method that was developed by Freeman and Orin [7]. Part II of this dissertation presents this algorithm in much more detail for the development of underwater simulation. An efficient implementation for land-based vehicles is shown to cost $[(224mN+59m+99)\text{M}, (205mN+66m+65)\text{A}]$. For the vehicles under consideration, this corresponds to [4485M, 4151A] and higher than the method developed

Table 12: Computational requirements for the CRB/DTS Method.

	×	+
$\mathbf{a}_r$	4	0
$\mathbf{f}'_r$	12	3
$\boldsymbol{\omega}_{k,0}$	$4m$	$2m$
$\mathbf{a}_{k,0}$	$16m$	$9m$
Inverse dynamics: (bias computation)		
$\boldsymbol{\omega}_{k,i}$	$8mN$	$5mN$
$\mathbf{a}_{k,i}$	$m(33N-10)$	$m(28N-10)$
$\mathbf{f}'_{k,i}$	$33mN$	$30mN$
$\mathbf{f}_{k,i}$	—	$6mN$
$[\mathbf{b}_k]_i$	—	—
${}^{i-1}\mathbf{f}_{k,i}$	$20mN$	$12mN$
${}^{r}\mathbf{f}'_k$	$10m$	$6m$
Inertia matrix:		
$\mathbf{K}_{k,i}$	—	$m(9N-9)$
$\mathbf{f}_{k,i}$	—	—
$[\mathbf{H}_k]_{i,i}$	—	—
${}^{i-1}\mathbf{K}_{k,i}$	$32mN$	$39mN$
${}^{i-1}\mathbf{f}_{k,i}$	$17mN$	$9mN$
$[\mathbf{H}_k]_{i,j}$	—	—
${}^{j-1}\mathbf{f}_{k,i}$	$m(10N^2-10N)$	$m(6N^2-6N)$
$[\mathbf{Q}_k]_i$	$10mN$	$6mN$
${}^{r}\mathbf{K}_k$	$15m$	$18m$
$\mathbf{K}_r$	—	$9m-6$
$\mathbf{K}^*_r$	$m(\frac{1}{6}N^3+6\frac{1}{2}N^2+20\frac{1}{3}N)$	$m(\frac{1}{6}N^3+6N^2+14\frac{5}{6}N)-6$
$\mathbf{f}^*_r$	$6mN$	$7mN+6m+6$
$\mathbf{a}_r$	86	65
$\ddot{\mathbf{q}}_k$	$m(N^2+6N)$	$m(N^2+5N)$
Total:	$m(\frac{1}{6}N^3+17\frac{1}{2}N^2+$ $175\frac{1}{3}N+35)+102$	$m(\frac{1}{6}N^3+13N^2+$ $164\frac{5}{6}N+31)+62$
$m=6,\ N=3$	4440	3944

in this chapter. For chains with more than three degrees of freedom, however, the AB method becomes more efficient.

5.5 Summary and Conclusions

In this chapter, the simulation of a multilegged vehicle is discussed. It is different from the multiple manipulator problem discussed in previous chapters primarily because the tips of the chains in the vehicle system contact the environment in a compliant way. With appropriate compliant foot contact modeling as in the Decoupled Tree-Structure (DTS) approach described by Freeman and Orin in [7], closed kinematic loops are removed from the legged vehicle system and the dynamics algorithm is simplified. Most notably the problem of singularities disappears.

The Composite-Rigid-Body (CRB) method is used as the basis for the dynamics algorithm developed in this chapter. The resulting CRB/DTS algorithm has a computational complexity of $O(mN^3)$. Since legged vehicles typically have few degrees of freedom in each chain, this method proves to be more computationally efficient than Articulated Body (AB)-based methods with a complexity of $O(mN)$. Specifically, computation of the dynamics for the OSU Hexapod requires [4440M, 3944A] which is somewhat better than the efficient implementation of the AB/DTS method. Another advantage to using the approach in this chapter is the ability to use previous research to parallelize the computation of the chain's joint space inertia matrices [38, 19, 45] to further increase computational rates.

Also note that the structure of the computation is the same as the multiple manipulator algorithm examined in the previous three chapters. The bias quantities,

$\mathbf{b}_k$ and ${}^r\mathbf{f}'_k$; inertia matrices, $\mathbf{H}_k$, $\mathbf{Q}_k$ and ${}^r\mathbf{K}_k$; and some of the intermediate computations for the reference member acceleration can all be performed independently for each chain. Then after the reference member acceleration is computed, the computation of the joint accelerations can also be performed independently. This is the same as the OCD/RMA/CCD structure discussed in Chapter 3. Hence, parallelization of this algorithm can be implemented in much the same way. Parallelization of the inertia matrix computation, as mentioned in the previous paragraph, would introduce a third dimension of parallelism which occurs *within* the spatial level.

Part II

Underwater Robotic Vehicle Simulation

CHAPTER VI

Hydrodynamic Forces on a Submerged Rigid Body

6.1 Introduction

In Part I, dynamic simulation algorithms were developed for land-based robotic systems. In this part, simulation algorithms for underwater robotic vehicle (URV) systems are developed. The demand for such systems has risen sharply in recent years and the applications are many and varied. Industrial applications include oil exploration and rig maintenance, cable laying, surveying, and construction. Deep-sea salvage operations as evidenced by the recent Titanic expeditions are also on the rise. Scientific exploration, both geological and biological, at great ocean depths is also increasing. A recent example of this is the study of the new volcanic activity along the Juan de Fuca ridge off the coast of Oregon.

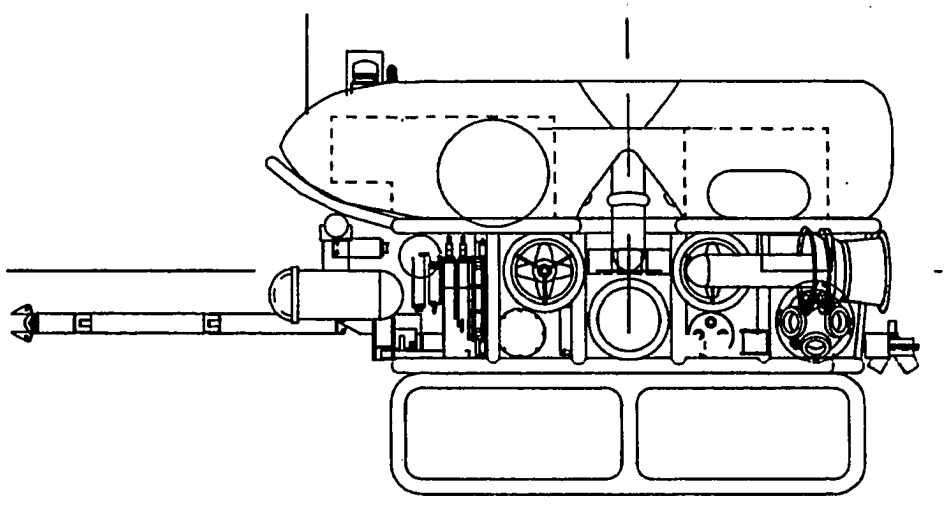
Some of the tasks can be performed by human divers, but underwater work is much more hazardous than on land. In addition, human divers are limited in the amount of time and the depth at which they can work. Much of the deep sea tasks are currently carried out with manned submarines sometimes equipped with robot manipulators that are controlled on-board. The cost of maintaining life-support suitable for the human operators within these subs, however, is quite high. As

such, replacement with unmanned underwater vehicles (UUVs) is desirable, and the development of these systems has been on the rise.

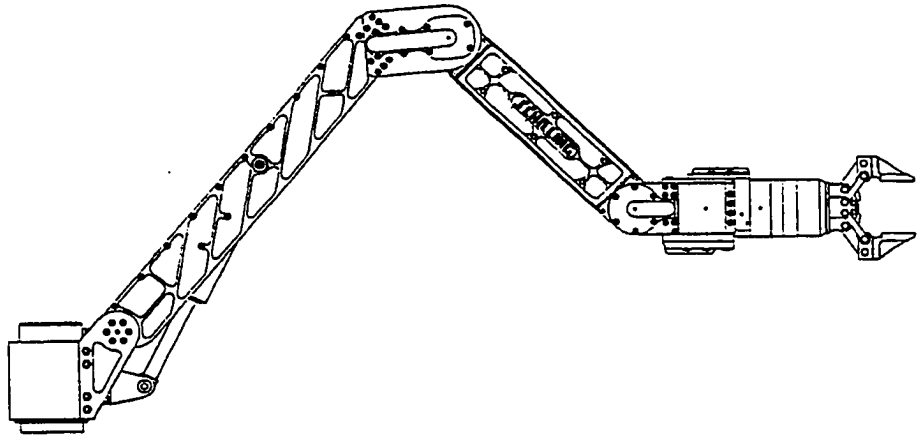
One system currently under development at the Monterey Bay Aquarium Research Institute (MBARI) is a new remotely-operated vehicle (ROV), *Tiburon*, for scientific research in and exploration of the Monterey Canyon. This vehicle is designed to be operated at depths of up to 4000 meters [46]. A CAD drawing of it is shown in Figure 18. Also shown is the Schilling Titan II manipulator [47] which is mounted on the front and will be used to place and retrieve scientific equipment, acquire geological samples, and even capture live biological samples.

Another system is a six legged vehicle, called *Aquarobot* (see Figure 19), being developed by the Port and Harbor Research Institute (PHRI) at Yokosuka, Japan for surveying and inspection during construction of sea walls in Japan [1]. The severe tidal currents that are present increase the danger to the human divers that currently perform the tasks. The legged design was chosen to meet these needs because its ability to accurately position itself under these conditions. It can also be made smaller, weigh less, and have better mobility than other types of vehicles that move on the seabed (wheeled, tracked, and screw-driven). The primary advantage, however, is its ability to walk over uneven terrain without stirring up the sediment. This keeps visibility around the vehicle high and enables the use of underwater cameras to monitor progress.

The need for better maneuverability or agility for these and other URV systems is driving the need to perform more research in and develop more sophisticated,



(a)



(b)

Figure 18: The MBARI UUV system: (a) *Tiburon*, and (b) the Schilling *Titan II* manipulator.

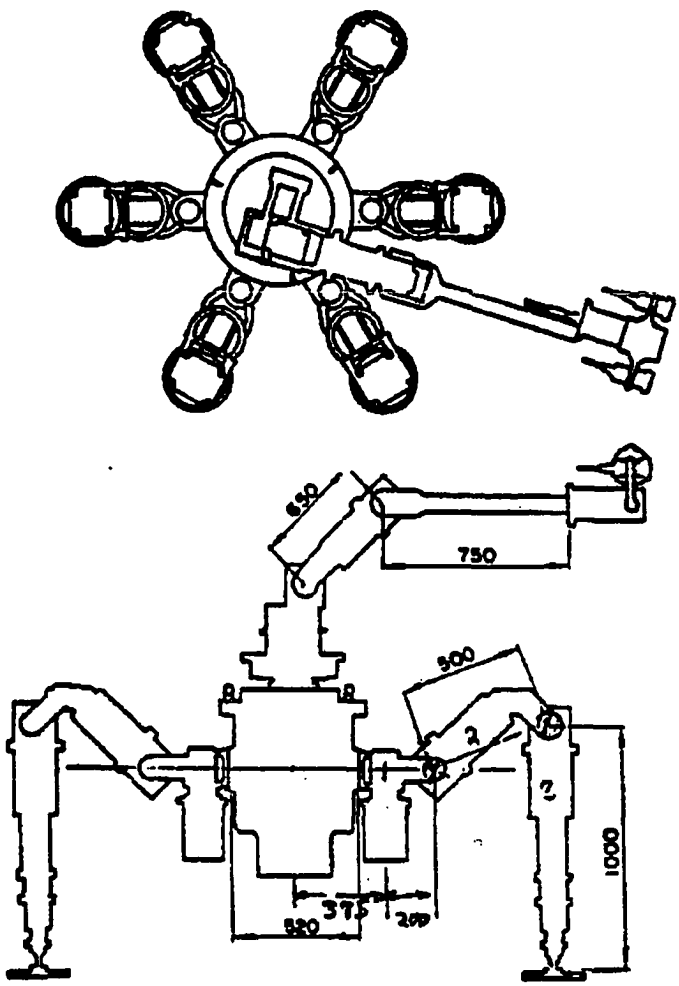


Figure 19: The *Aquarobot* hexapod (dimensions in mm) [1].

real-time control algorithms. Indeed, a number of papers have been written about the development of new control schemes for URVs [48, 49, 50, 51]. Some of this work addresses the problems for a single rigid body, and the hydrodynamic models used as the basis for justifying these control algorithms are simplified. This motivates the development of the best possible dynamic and hydrodynamic simulations of URVs, which has already been proven as a cost-effective development tool for equivalent land- and space-based systems. With such a tool, preliminary control algorithms can be designed and evaluated off-line without exposing expensive prototypes to completely untested algorithms that could lead to damaging instabilities. The advantage is increased when testing must proceed in the more hazardous and unpredictable underwater environments. A high fidelity simulation could also be used to evaluate the accuracy of simplified dynamics algorithms themselves which may be eventually used in model-based control algorithms.

The development and implementation of a dynamic and hydrodynamic simulation algorithm for URV systems with articulated bodies like the two described above is undertaken in this part of the dissertation. Because a significant amount of research has already been accomplished to achieve very efficient algorithms for land-based robotic vehicles, they will be used as the basis for this development. In order to use these algorithms for URV simulation, the computation of hydrodynamic forces must be accomplished first. Therefore, the purpose of this chapter is to develop detailed algorithms for computing hydrodynamic forces. Then, incorporation of these forces into an efficient dynamics simulation algorithm will be presented

in Chapter 7. To complete this task, an object-oriented approach is taken to efficiently implement an algorithm that is capable of simulating URV systems with multiple serial chains (especially the two systems mentioned above) and is presented in Chapter 8.

In order to incorporate hydrodynamics into existing dynamic simulation algorithms, algorithms to compute the hydrodynamic forces on each rigid body in the system must be developed. These account for a number of additional effects when the motion of rigid bodies is to be simulated in an underwater environment. These forces result from incompressible fluid flow determined by the Navier-Stokes (distributed fluid-flow) equations [52]. The solution of these equations is extremely costly especially for irregularly shaped bodies undergoing general motion, and where solutions are possible, supercomputers must generally be employed.

In this chapter, "lumped" approximations to these forces are used in order to approach the desired simulation rates. To this end, work by Yuh [49] and Ioi and Itoh [50] have identified the most significant forces on a rigid body, which include added mass, viscous drag and lift, buoyancy, and fluid acceleration as illustrated in Figures 20 and 21. To facilitate incorporation into the multibody simulation algorithms in Chapter 7, particular emphasis is placed on developing notation that is consistent with the spatial notation used throughout this dissertation. In the next section, the equation for the added mass force is derived. In Section 6.3, lift forces are discussed and a procedure based on strip theory is developed to compute a drag force. Due to the similarity of the resulting equations, discussion of buoyancy and

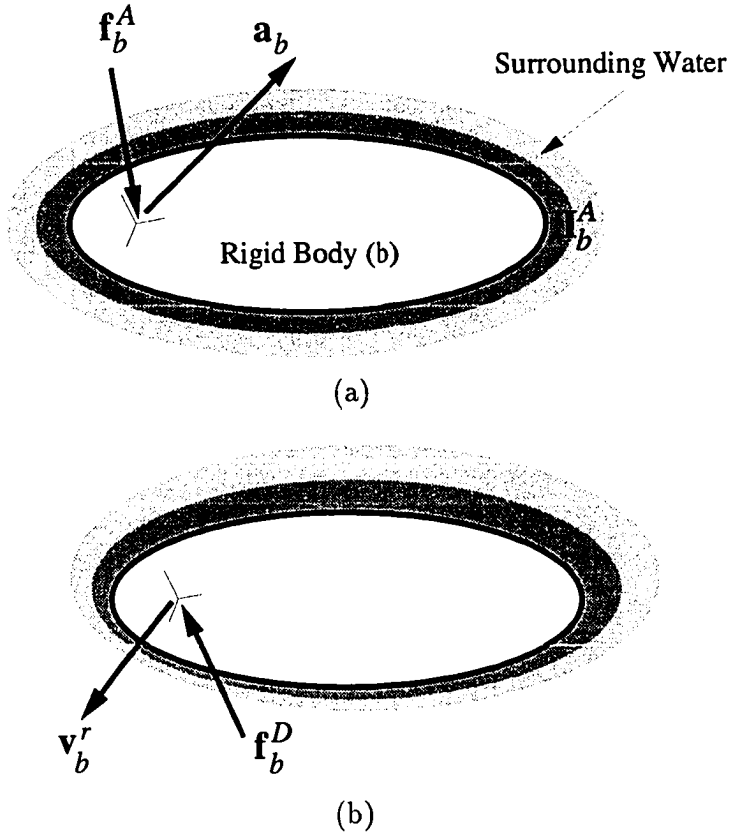
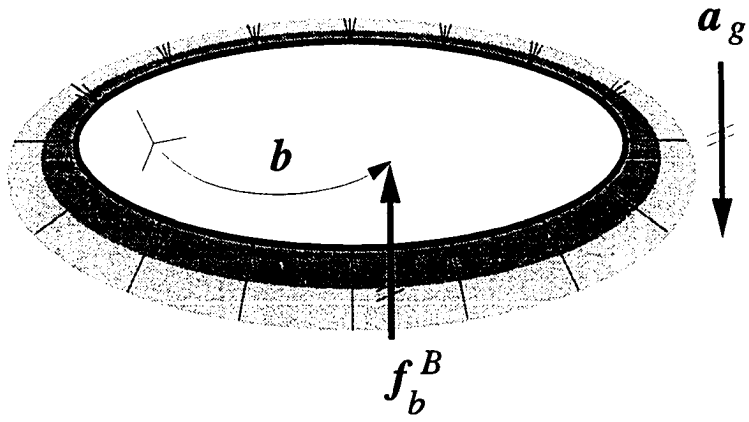


Figure 20: Hydrodynamic forces: (a) added mass and (b) drag.

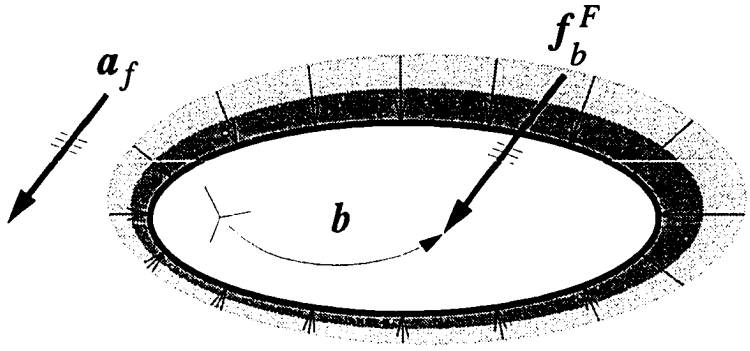
forces arising from fluid acceleration are combined in Section 6.4.

6.2 Added Mass

To those acquainted with the dynamics of manipulators in space or on land, probably the most surprising hydrodynamic effect is the added mass force. When a body is accelerated through a fluid, some of the surrounding fluid is also accelerated with the body. This fluid has mass/inertia properties that can be described with a 6×6 added



(c)



(d)

Figure 21: Hydrodynamic forces (continued): (c) buoyancy and (d) fluid acceleration.

mass matrix. A force is exerted on this surrounding fluid to achieve the acceleration, and the reaction force exerted on the body is the added mass force which is equal in magnitude and opposite in direction. The most important characteristic of this force is that it is also a function of the acceleration of the rigid body. In this section, the added mass force equation is derived using spatial notation, and is modified to a form that can be easily incorporated into robot dynamics algorithms. Then some

comments about the coefficients of the added mass matrix are made.

6.2.1 Derivation of the Added Mass Force Equation

Newman [53] derives a set of equations to compute the added mass force that is exerted on a rigid body accelerating through an unbounded, inviscid fluid undergoing steady, irrotational flow (that is, not accelerating). This can be found by taking the derivative of the total momentum of the fluid. The results are repeated here and are translated to spatial notation. First some of Newman's notation is defined. The ijth element of the added mass matrix is denoted by m_{ij}. The body's translational velocity expressed in the body-fixed coordinate system is given by the following vector:

$$\boldsymbol{v}_b = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}, \tag{6.1}$$

and a redundant notation is used to define the elements of angular velocity:

$$\boldsymbol{\omega}_b = \begin{bmatrix} U_4 \\ U_5 \\ U_6 \end{bmatrix} = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \end{bmatrix}. \tag{6.2}$$

Finally an "indicial notation" for the cross product operation is defined. The jth component of the result is given as follows:

$$(\boldsymbol{b} \times \boldsymbol{c})_j = \epsilon_{jkl} b_k c_l, \tag{6.3}$$

where summations are implied on the right hand side for both k and l from 1 to 3.

The equation to compute the three components of translational force from [53] is given by [Eq. (115), Ch. 4]:

$$F_j = -\dot{U}_i m_{ji} - \epsilon_{jkl} U_i \Omega_k m_{li}, \quad j = 1, 2, 3, \tag{6.4}$$

where, again, summations are implied for k and l from 1 to 3, and for i from 1 to 6. To obtain the vector equation, the summation on i is made explicit and the cross product term is isolated as follows:

$$F_j = -\sum_{i=1}^{6} m_{ji}\dot{U}_i - \sum_{i=1}^{6} U_i\left(\epsilon_{jkl}\Omega_k m_{li}\right), \quad j = 1,2,3. \tag{6.5}$$

Then, the summations are expanded and the cross products are expressed in vector notation:

$$F_j = -\left(m_{j1}\dot{U}_1 + m_{j2}\dot{U}_2 + \cdots + m_{j6}\dot{U}_6\right) - \left[U_1\left(\boldsymbol{\omega}_b \times \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \end{bmatrix}\right)_j + U_2\left(\boldsymbol{\omega}_b \times \begin{bmatrix} m_{12} \\ m_{22} \\ m_{32} \end{bmatrix}\right)_j + \cdots + U_6\left(\boldsymbol{\omega}_b \times \begin{bmatrix} m_{16} \\ m_{26} \\ m_{36} \end{bmatrix}\right)_j\right]. \tag{6.6}$$

Finally, this is written in a form consistent with our spatial notation as follows:

$$\boldsymbol{f}_b^A = -\left[\mathbf{M}_{12} \;\; \mathbf{M}_{11}\right]\begin{bmatrix} \dot{\boldsymbol{\omega}}_b \\ \dot{\boldsymbol{v}}_b \end{bmatrix} - \tilde{\boldsymbol{\omega}}_b\left[\mathbf{M}_{12} \;\; \mathbf{M}_{11}\right]\begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{bmatrix}, \tag{6.7}$$

where $\boldsymbol{f}_b^A = [F_1 \; F_2 \; F_3]^T$ and the 3×3 blocks of added mass components are defined as follows:

$$\left[\mathbf{M}_{12} \;\; \mathbf{M}_{11}\right] = \left[\begin{array}{ccc|ccc} m_{14} & m_{15} & m_{16} & m_{11} & m_{12} & m_{13} \\ m_{24} & m_{25} & m_{26} & m_{21} & m_{22} & m_{23} \\ m_{34} & m_{35} & m_{36} & m_{31} & m_{32} & m_{33} \end{array}\right]. \tag{6.8}$$

Note that these blocks are "swapped" because spatial notation requires that angular components of spatial vectors appear above the translational components which is the opposite of the convention used in [53].

The equations for the moment exerted on the body due to added mass is given in [53] as follows [Eq. (116), Ch. 4]:

$$N_j = -\dot{U}_i m_{j+3,i} - \epsilon_{jkl}U_i\Omega_k m_{l+3,i} - \epsilon_{jkl}U_i U_k m_{li}, \quad j = 1,2,3, \tag{6.9}$$

where the components of moment have been specified with N_j instead of the M_j used in [53] to avoid confusion with mass coefficients, and the implicit summations of Eq. (6.4) also apply. This can be rewritten, as before, to highlight the summation over i and the cross product operation, as follows:

$$N_j = -\sum_{i=1}^{6} m_{j+3,i}\dot{U}_i - \sum_{i=1}^{6} U_i\left(\epsilon_{jkl}\Omega_k m_{l+3,i}\right) - \sum_{i=1}^{6} U_i\left(\epsilon_{jkl}U_k m_{li}\right). \tag{6.10}$$

In Cartesian vector notation, this is written as follows:

$$\begin{aligned} N_j = & -\left(m_{j+3,1}\dot{U}_1 + m_{j+3,2}\dot{U}_2 + \cdots + m_{j+3,6}\dot{U}_6\right) - \\ & \left[U_1\left(\boldsymbol{\omega}_b \times \begin{bmatrix} m_{41} \\ m_{51} \\ m_{61} \end{bmatrix}\right)_j + U_2\left(\boldsymbol{\omega}_b \times \begin{bmatrix} m_{42} \\ m_{52} \\ m_{62} \end{bmatrix}\right)_j + \cdots + U_6\left(\boldsymbol{\omega}_b \times \begin{bmatrix} m_{46} \\ m_{56} \\ m_{66} \end{bmatrix}\right)_j\right] - \\ & \left[U_1\left(\boldsymbol{v}_b \times \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \end{bmatrix}\right)_j + U_2\left(\boldsymbol{v}_b \times \begin{bmatrix} m_{12} \\ m_{22} \\ m_{32} \end{bmatrix}\right)_j + \cdots + U_6\left(\boldsymbol{v}_b \times \begin{bmatrix} m_{16} \\ m_{26} \\ m_{36} \end{bmatrix}\right)_j\right], \end{aligned} \tag{6.11}$$

where $j = 1, 2, 3$. In spatial notation, this reduces to:

$$\begin{aligned} \boldsymbol{n}_b^A = & -\left[\mathbf{M}_{22} \;\; \mathbf{M}_{21}\right]\begin{bmatrix} \dot{\boldsymbol{\omega}}_b \\ \dot{\boldsymbol{v}}_b \end{bmatrix} - \tilde{\boldsymbol{\omega}}_b\left[\mathbf{M}_{22} \;\; \mathbf{M}_{21}\right]\begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{bmatrix} - \\ & \tilde{\boldsymbol{v}}_b\left[\mathbf{M}_{12} \;\; \mathbf{M}_{11}\right]\begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{bmatrix}, \end{aligned} \tag{6.12}$$

where the added mass coefficients used in [53] are placed in the following order:

$$\left[\mathbf{M}_{22} \;\; \mathbf{M}_{21}\right] = \left[\begin{array}{ccc|ccc} m_{44} & m_{45} & m_{46} & m_{41} & m_{42} & m_{63} \\ m_{54} & m_{55} & m_{56} & m_{51} & m_{52} & m_{53} \\ m_{64} & m_{65} & m_{66} & m_{61} & m_{62} & m_{43} \end{array}\right], \tag{6.13}$$

and $\boldsymbol{n}_b^A = [N_1 \; N_2 \; N_3]^T$ is the added mass moment exerted on the rigid body.

By combining Eqs. (6.7) and (6.12), the desired equation for the added mass force is obtained in spatial notation:

$$\mathbf{f}_b^A = \begin{bmatrix} \boldsymbol{n}_b^A \\ \boldsymbol{f}_b^A \end{bmatrix} = -\mathbf{I}_b^A\begin{bmatrix} \dot{\boldsymbol{\omega}}_b \\ \dot{\boldsymbol{v}}_b \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_b & \tilde{\boldsymbol{v}}_b \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_b \end{bmatrix}\mathbf{I}_b^A\begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{bmatrix}, \tag{6.14}$$

where the 6×6 added mass matrix, $\mathbf{I}_b^A$, is written in terms of Newman's coefficients as:

$$\mathbf{I}_b^A = \begin{bmatrix} \mathbf{M}_{22} & \mathbf{M}_{21} \\ \mathbf{M}_{12} & \mathbf{M}_{11} \end{bmatrix}. \tag{6.15}$$

A brief discussion about their computation appears below.

In Eq. (6.14), the negative signs are needed to compute the reaction force that is exerted *onto* the rigid body, and $\boldsymbol{\omega}_b$ and $\boldsymbol{v}_b$ are the angular and translational velocities of the body, respectively. The translational acceleration term, $\dot{\boldsymbol{v}}_b$, is not the true acceleration of the rigid body, but is rather the time derivative of $\boldsymbol{v}_b$ with respect to the body's rotating reference frame.*

Since the efficient robot simulation algorithms use the true acceleration of the body, $\boldsymbol{a}_b$, Eq. (6.14) must be modified before it can be efficiently incorporated into the algorithm. This is accomplished by using the following relationship:

$$\boldsymbol{a}_b = \dot{\boldsymbol{v}}_b + \boldsymbol{\omega}_b \times \boldsymbol{v}_b, \tag{6.16}$$

where $\dot{\boldsymbol{v}}_b$ is frequently referred to as the *rate of growth* of the acceleration vector, and $\boldsymbol{\omega}_b \times \boldsymbol{v}_b$ its *rate of transport* [55]. Substituting this into Eq (6.14) leads to the following equation for the added mass force:

$$\mathbf{f}_b^A = -\mathbf{I}_b^A \begin{bmatrix} \dot{\boldsymbol{\omega}}_b \\ \boldsymbol{a}_b \end{bmatrix} + \mathbf{I}_b^A \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\omega}_b \times \boldsymbol{v}_b \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_b & \tilde{\boldsymbol{v}}_b \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_b \end{bmatrix} \mathbf{I}_b^A \begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b \end{bmatrix}. \tag{6.17}$$

Now, the effects of fluid translational acceleration on the force resulting from added mass can be incorporated by replacing the translational acceleration of the

*The elements of which are $\dot{u}$, $\dot{v}$, and $\dot{w}$ from [54].

rigid body with its translational acceleration *relative* to the surrounding fluid, $\boldsymbol{a}_b^r$ ([53], p. 150). This is defined as follows:

$$\boldsymbol{a}_b^r = \boldsymbol{a}_b - {}^b\boldsymbol{a}_f, \tag{6.18}$$

where ${}^b\boldsymbol{a}_f$ is the translational acceleration of the fluid expressed in the body-fixed coordinate system. Likewise, the translational velocity term is replaced with the relative translational velocity, $\boldsymbol{v}_b^r$, as follows:

$$\mathbf{f}_b^A = -\mathbf{I}_b^A \begin{bmatrix} \dot{\boldsymbol{\omega}}_b \\ \boldsymbol{a}_b \end{bmatrix} + \mathbf{I}_b^A \begin{bmatrix} \mathbf{0} \\ {}^b\boldsymbol{a}_f + \boldsymbol{\omega}_b \times \boldsymbol{v}_b^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_b & \tilde{\boldsymbol{v}}_b^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_b \end{bmatrix} \mathbf{I}_b^A \begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b^r \end{bmatrix}. \tag{6.19}$$

Finally, it is desirable to write the equation in the following form which will facilitate incorporation into the robot dynamics algorithms:

$$\mathbf{f}_b^A = -\mathbf{I}_b^A \mathbf{a}_b + \boldsymbol{\beta}_b^A, \tag{6.20}$$

where

$$\boldsymbol{\beta}_b^A = \mathbf{I}_b^A \begin{bmatrix} \mathbf{0} \\ {}^b\boldsymbol{a}_f + \boldsymbol{\omega}_b \times \boldsymbol{v}_b^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_b & \tilde{\boldsymbol{v}}_b^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_b \end{bmatrix} \mathbf{I}_b^A \begin{bmatrix} \boldsymbol{\omega}_b \\ \boldsymbol{v}_b^r \end{bmatrix}, \tag{6.21}$$

which is the added mass bias force and contains all of the terms that are a function of the state and the known fluid acceleration and velocity.

Efficient computation of the bias force, assuming $\mathbf{I}_b^A$ is full, is straightforward. The first term is computed with a cross product, Cartesian vector addition, and the equivalent of two Cartesian matrix-vector multiplies for a total of [24M, 18A]. The second term is obtained with a spatial matrix-vector multiply, three cross products, and a Cartesian vector addition for a total of [54M, 42A]. Assuming the fluid velocity and acceleration as well as the body's relative velocity have already been computed

with respect to the body's coordinate system, adding the two results together is an additional spatial vector addition for a total of [78M, 66A].

6.2.2 Added Mass Coefficients

The added mass matrix describes the mass and inertia properties of the fluid surrounding the body that undergoes acceleration with the body. It is a distributed characteristic because the fluid nearer to the body undergoes greater acceleration than more distant fluid; therefore, the local fluid contributes more to the added mass. Various ways may be used to compute the coefficients of the added mass matrix as is shown in both Newman [53] and Sarpkaya and Isaacson [56]. One method involves the integral of the velocity potentials over the surface of the body. Another, involves an integral of the kinetic energy of the fluid within an appropriate volume surrounding the body. Both imply a dependence on the shape and motion of the body. For symmetric shapes, these equations are simplified, and for some of these shapes undergoing unseparated motion, both references, [53, 56], give tables of these coefficients. In general, however, the relationships are very complex and the coefficients can vary over time for separated motion. The determination of these coefficients has been an ongoing topic of research for over 50 years, and these authors refer to extensive lists of references.

Using lumped approximations, the fluid that is accelerated is assumed to have fixed inertia properties, and $\mathbf{I}_b^A$ is constant. As with the spatial inertia for a rigid body, this added mass matrix is symmetric and positive definite. Since this inertia is still a function of the body's surface geometry, however, there is no concept of

principal axes as in rigid body analysis, along which, torque and angular momentum are collinear. In fact, with added mass, unlike the rigid body's mass, a translational force applied to a submerged rigid body can result in a non-collinear acceleration of its center of gravity as well. Consequently, the added mass matrix does not have the structure shown in Eq. (1.7) for the spatial inertia of a rigid body in space or air. For a general body shape, the matrix will be full which leads to notably different dynamic behavior.

6.3 Viscous Forces

When an object rotates and translates in a viscous fluid, which includes both air and water, *drag* and *lift* forces are also exerted on it. Both of these forces are a function of the spatial velocity of the body relative to the fluid and are discussed in this section. For simple motion, drag is a friction-like force that is in-line with the relative velocity of the body and opposite in direction; whereas, lift is a force perpendicular to the relative velocity. For most land-based robotics, the motions are slow enough and the air is thin enough that drag (air resistance) can be neglected. For underwater robotics, the fluid is significantly denser and even reasonably slow motions can result in large drag forces. This force is accentuated if the arm must move quickly to, for example, track living organisms for sample acquisition, or when operation in a strong current is desired. For the applications targeted in this development, drag is assumed to be the dominant affect and is discussed below. Lift effects including vortex shedding are discussed briefly at the end of this section.

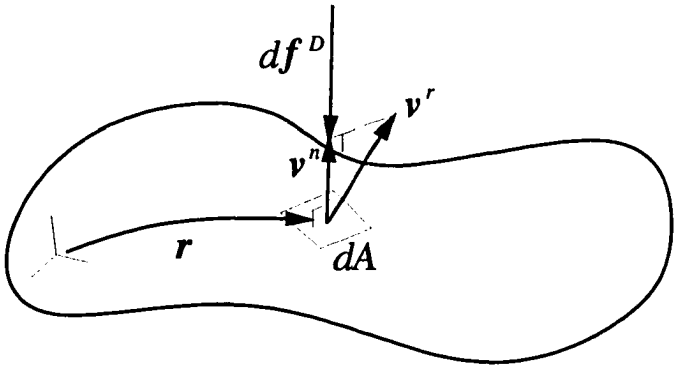


Figure 22: Drag force on a general body.

6.3.1 Drag

Drag has been simplistically defined as a force proportional to the square of the relative velocity, in a direction opposite to the flow, and related by the drag coefficient, C_D. More generally, drag force is distributed over the surface of the body which can be decomposed into pressure (or profile) drag which is normal to the surface of the body, and shear (or skin) drag which is the tangential component. For the underwater manipulators, shear drag forces are much smaller than pressure drag forces so that the emphasis here is on the modeling of the latter. Pressure drag is a function of the components of the relative velocity between the body and the fluid that are normal to the surface of the body. From the distributed viewpoint, these velocity components are illustrated Figure 22 for an element of the surface of a general body. For each element of surface area, dA, on the body, the component of its relative velocity, $\boldsymbol{v}^r$, which is normal to the surface must be determined. This

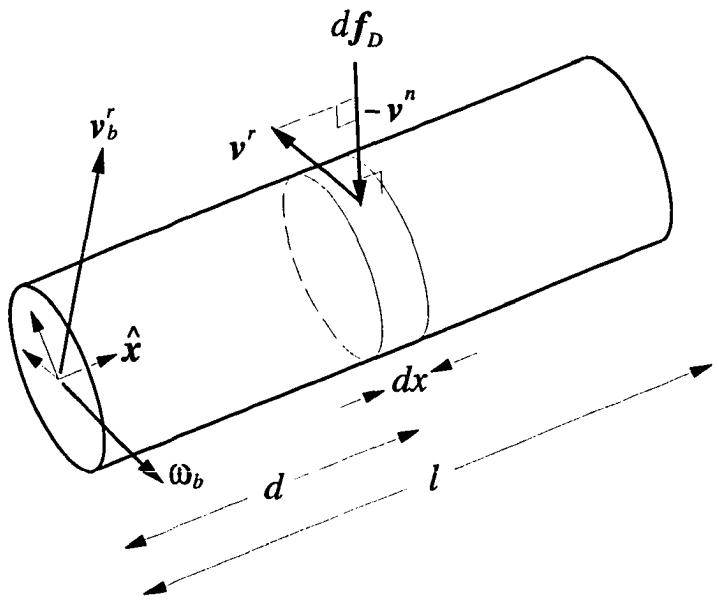


Figure 23: Drag force on a cylinder.

is labeled $\boldsymbol{v}^n$ in the figure. The partial drag force on this element, $d\boldsymbol{f}^D$, is parallel to $\boldsymbol{v}^n$ and can be computed as follows:

$$d\boldsymbol{f}^D = -0.5\rho C_D \, ||\boldsymbol{v}^n|| \, \boldsymbol{v}^n dA, \tag{6.22}$$

where ρ is the fluid density, C_D is the drag coefficient, and $||\boldsymbol{v}^n||$ is the magnitude of the velocity vector normal to the surface. Likewise, the partial moment about the body fixed coordinate system is given as follows:

$$d\boldsymbol{n}^D = -0.5\rho C_D \, ||\boldsymbol{v}^n|| \, (\boldsymbol{r} \times \boldsymbol{v}^n) \, dA. \tag{6.23}$$

Finally, surface integrals over the entire body are required to compute the total drag force and moment exerted on the body.

This surface integral is computationally expensive, and the drag coefficients are generally poorly known and variable. Therefore, simplifying assumptions are made to significantly reduce the amount of computation while still computing a "reasonable" drag force. The most significant assumption in this development is that the links can be approximated by cylinders or other simple symmetrical shapes. It is also desirable that one of the link's coordinate axes lies along an axis of symmetry as shown in Figure 23 for a cylinder. This is not an unreasonable assumption since, most often, the x- or y-axis points along the length of links with revolute joints, and the z-axis points along prismatic links when using modified Denavit-Hartenberg parameters. Otherwise, a simple transformation can be added to the procedure to achieve this condition. The resulting procedure is based on one by Sarpkaya and Isaacson [56] to compute a drag force on a stationary cylinder arbitrarily oriented with respect to the fluid velocity. It has been extended in this section to include the effects of arbitrary translational and angular velocity of the cylinder as well.

With cylinders, strip theory is utilized to replace the surface integral with a line integral along the length of the cylinder. Therefore, the cylinder is partitioned into circular disk elements with width dx as shown in the figure, and the translational velocity relative to the fluid and normal to the edge of each disk, $\boldsymbol{v}^n$, must be determined. The translational velocity of a disk relative to the fluid at a distance d along the cylinder's axis (the x-axis in this example) is approximated, assuming the radius of the cylinder, r, is small compared to the length, as follows:

$$\boldsymbol{v}^r(d) = \boldsymbol{v}_b^r + \boldsymbol{\omega}_b \times \begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}. \tag{6.24}$$

where ω_b is the angular velocity of the cylinder which is also the angular velocity relative to the fluid since it is assumed to be irrotational, and v_b^r is the translational velocity of the cylinder relative to the fluid at the body-fixed coordinate system. The relative velocity normal to this disk, $v^n(d)$, is this vector's projection onto the yz-plane of the link's coordinate system which is the y and z components of $v^r(d)$.

The equation to compute partial drag force exerted on the edge of this disk is obtained by modifying Eq. (6.22) for circular disks as follows:

$$df_b^D(d) = -0.5\rho C_D \, ||v^n(d)|| \, v^n(d) \, (2r \, dx), \tag{6.25}$$

where the rightmost term inside the parentheses is the projected area of the disk normal to the fluid flow which corresponds to dA in the initial equations. The equation for the corresponding partial drag moment about the body-fixed coordinate system is similarly obtained as follows:

$$dn_b^D(d) = -0.5\rho C_D \, ||v^n(d)|| \left(\begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix} \times v^n(d) \right) 2r \, dx. \tag{6.26}$$

Eqs. (6.25) and (6.26) must then be integrated along the length of the cylinder to obtain the resultant drag force and moment as follows:

$$f_b^D = -\rho C_D r \int_0^l ||v^n(x)|| \, v^n(x) dx \tag{6.27}$$

$$n_b^D = -\rho C_D r \int_0^l ||v^n(x)|| \left(\begin{bmatrix} x \\ 0 \\ 0 \end{bmatrix} \times v^n(x) \right) dx, \tag{6.28}$$

which correspond to the bottom and top three elements of the spatial drag force, respectively.

A numerical solution to these integrals can be obtained by using a Gauss-Legendre formula [57] which approximates the integral by computing a weighted

sum of the integrand at a number of points across its range. In general, the integrand is well approximated by a polynomial, and as the number of points increases, so does the accuracy of the result. In a tradeoff between accuracy and the amount of computation required, which also increases with the number of points, a four-point version of the formula has been chosen in this research which yields an eighth order result. For an arbitrary integrand, $\boldsymbol{h}(x)$, this formula is given as follows:

$$\int_0^l \boldsymbol{h}(x)\,dx \;\approx\; l \sum_{k=1}^{4} g_k\, \boldsymbol{h}(l\, x_k), \tag{6.29}$$

where the x_k are the four points at which the differential equation should be evaluated, and g_k are the weights. Both of these have been determined for the four-point Gauss-Legendre (also called Gauss quadrature) method as follows:

k	x_k	g_k
1	0.069431844...	0.173927422...
2	0.330009478...	0.326072577...
3	0.669990521...	0.326072577...
4	0.930568155...	0.173927422...

An off-line procedure for determining these values to the desired accuracy is also given in [57].

The result of this procedure yields zero values for the x-components of both the moment and force vectors. For smooth cylinders, a moment about the x-axis exists which would only be caused by x-components of angular velocity and shear drag. Therefore, a zero value in this case is consistent with the earlier assumption that shear drag is negligible. A drag force along the x-axis can exist, however, due to a drag force exerted on the flat ends of the cylinder to relative translational velocity

along the x-axis. In this case, the x-component of $\boldsymbol{f}_b^D$ is simply computed as follows:

$$(f_b^D)^x = -0.5\rho C_D \left(\pi r^2\right) ||(v_b^r)^x|| \, (v_b^r)^x, \tag{6.30}$$

where $(v_b^r)^x$ is the x-component of the link's translational velocity with respect to the fluid and πr^2 is the area projected normal to the flow in this direction. The assumption that this component of the drag force can be computed in this manner from normal components of relative velocity for this surface while ignoring components computed for the other surfaces is consistent with the the *independence principle* [56] and the work of Chakrabarti, Tam, and Wolbert [58].

This force is determined most efficiently by computing the results of Eqs. (6.27), (6.28), and (6.30). Eq. (6.30) requires [2M, 0A] for the on-line part of the computation. The integrals in the first two equations, can be numerically determined using the Gauss-quadrature method which replaces the integral with a weighted sum of the integrand evaluated at four points over its range. The terms within both integrals require [9M, 3A] and a square root (which will be counted as a multiplication) which also includes the multiplication by the weight and the computation of $\boldsymbol{v}^r(d)$ from Eq. (6.24). Note that both of the vectors within the integrals contain one zero element. This step is performed four times, [40M, 12A], the results are added together [0M, 12A], and then multiplied by the constant outside of the integral, [4M, 0A]. Again, assuming that the relative velocity of the body has been computed, this procedure requires [46M, 24A] and the result contains five nonzero elements.

6.3.2 Lift and Vortex Shedding

Lift force is usually described as a force in a direction normal to the fluid flow, and is caused by non-zero net circulation of the fluid around the body [53]. It is similar to drag because it is also proportional to the square of the relative velocity. Analogous to C_D, the proportionality is defined by the coefficient of lift C_L. Like drag, this definition does not seem to be adequate for the three-dimensional derivations needed in this paper. The line integrals that are used to compute the drag force, can, in the case of general translation and rotation, result in components normal and parallel to the fluid velocity. We have called this the computation for drag forces only. Significant, steady lift forces that are distinct from the forces in that derivation are present when the body is a hydrofoil. Since the rigid bodies that make up the URVs under consideration are not foils, computation of these lift forces is omitted.

While hydrofoils with reasonable angles of attack lead to significant, steady lift forces, very large angles of attack or bluff bodies can experience significant forces due to vortex shedding. This is a second lift effect that has not been considered in this discussion. When the flow past a body becomes separated, pairs of vortices build up behind it. As the relative velocity of the body increases, these vortices continue to grow in size until instabilities cause them to alternately detach from the body. This introduces additional periodic lift forces on the body. The amplitude of this force can become large [59] if it happens to excite one of the natural frequencies of the system. For general motion of a single body, computation of these effects is extremely difficult, and precise modeling of these effects for underwater

manipulators would be even more lengthy. A straight-forward check of the Strouhal Number/Reynolds Number map, however, can yield estimates of the magnitude and frequency of this force, and can indicate if vortex shedding effects are a potential problem. Engineering solutions to reduce or even eliminate the problem using spoilers can be effective, so this phenomenon can be made small enough to ignore in most robotic applications.

6.4 Total Buoyancy: Buoyancy and Fluid Acceleration

Because of the similarity between buoyancy and fluid acceleration forces, they are presented together in this section. Both are translational forces as illustrated in Figure 21. They are both exerted through the center of buoyancy of the body, which is the center of volume of the body or equivalently the center of mass of the fluid that is displaced by the body. Finally, they are proportional to the mass of the fluid that is displaced by the body, m_b^f.

The buoyant force, $\boldsymbol{f}_b^B$, is exerted on the body in the direction opposite of gravity. This force is a result of Archimedes principle which states that a body immersed in a fluid is buoyed up with a force equal to the weight of the fluid displaced by the body. Therefore, this force is specified as follows:

$$\boldsymbol{f}_b^B = -m_b^f \, {}^b\boldsymbol{a}_g, \tag{6.31}$$

where ${}^b\boldsymbol{a}_g$ is the gravitational acceleration expressed in the body's coordinate system. In general, this acceleration is defined as follows:

$$\boldsymbol{a}_g = g \, \hat{\mathbf{z}}_e, \tag{6.32}$$

where $\hat{\mathbf{z}}_e$ is the unit vector that points "down" as is traditional in marine mechanics [54], and g is the gravitational constant. This can also be thought of as the resultant force due to pressure on the surface of the body which increases with depth as illustrated by the thin arrows surrounding the body in Figure 21(c).

A similar equation was given by Newman ([53], p. 152) for the fluid acceleration force which is given as follows:

$$\boldsymbol{f}_b^F = m_b^f \, {}^b\boldsymbol{a}_f, \tag{6.33}$$

where ${}^b\boldsymbol{a}_f$ is the acceleration of the fluid also expressed in the body's coordinate system. It can also be thought of as the resultant force due to pressure that is greater at the leading edge of the body as shown in Figure 21(d). Because of the similarities, this force is often referred to as the "horizontal buoyancy" force.

For increased computational efficiency, both forces are combined as follows:

$$\boldsymbol{f}_b^{TB} = m_b^f \left({}^b\boldsymbol{a}_f - {}^b\boldsymbol{a}_g \right), \tag{6.34}$$

and will be referred to as the "total buoyancy" force. To use this in existing robot dynamics algorithms, the equivalent spatial force exerted at the origin of the body-fixed coordinate system must be found. The resulting force is computed by the following equation:

$$\mathbf{f}_b^{TB} = \begin{bmatrix} \boldsymbol{b}_b \times \boldsymbol{f}_b^{TB} \\ \boldsymbol{f}_b^{TB} \end{bmatrix}, \tag{6.35}$$

where $\boldsymbol{b}_b$ is the vector from the body-fixed coordinate system to its center of buoyancy. With ${}^b\boldsymbol{a}_g$ and ${}^b\boldsymbol{a}_f$ already computed, the computation of this force requires [9M, 6A].

6.5 Summary and Conclusions

In this chapter, the most significant hydrodynamic effects present in most underwater robotic activities have been presented. Of these, computational procedures for computing added mass, drag, fluid acceleration, and buoyancy effects have been presented. Because computation of precise forces is extremely difficult and time consuming (computational fluid dynamics is a task usually relegated to supercomputers), a number of simplifying assumptions have been made. The most notable assumptions are that the fluid is irrotational and unbounded. The former is acceptable since rotation due to any vortices in the fluid are generally small compared to rotation of the body, or limited to such a small scale (as in the case of shed vortices) compared to the extent of the body as to be negligible. The exception is circular wave action in shallow depths, which is not an environment that is encountered in applications reserved for UUV systems. The unbounded assumption poses more of a problem, but for first order approximations it should generally be acceptable.

Another assumption is that the added mass matrix and drag coefficients are known and constant. In actuality, these quantities are extremely difficult to compute with a high degree of accuracy, and vary non-linearly with respect to velocity and other parameters [56]. However, we believe that over the range of operating conditions typically encountered by a UUV, these coefficients vary by only small amounts such that a constant coefficient assumption is a reasonable approach and is adequate for the purposes of the desired application. With these results, the next chapter develops a multibody dynamics algorithm and incorporates these hy-

drodynamic effects in order to develop a simulator for the URVs illustrated in the introduction to this chapter.

CHAPTER VII

Efficient Simulation of Underwater Robotic Vehicle Systems

7.1 Introduction

The primary goal of this chapter is to develop a general, but efficient algorithm to compute the dynamics of underwater robotic vehicle (URV) systems with multiple chains. The remotely operated vehicle (ROV), *Tiburon*, with a single Schilling manipulator and the six-legged Aquarobot, as illustrated in Figures 18 and 19, are specifically targeted in this development. The Aquarobot has the same general topology as legged vehicles examined in Chapter 5, so the decoupled tree-structure (DTS) approach is a general, but efficient, approach for modeling this system. This approach can also be applied to the ROV/manipulator system in which the reference member has a single chain attached.

Using DTS modeling, the resulting systems require an open chain dynamics algorithm, and in Chapter 5, Walker and Orin's Composite Rigid Body (CRB) method [14] was used in the development for such systems on land (or in space). Compared to results obtained later in this chapter for an equivalent algorithm based on Featherstone's Articulated Body (AB) method [15], the CRB method is more efficient only when the number of degrees of freedom in each chain is less than

four. Disregarding the additional computation for hydrodynamic effects, the CRB method is more efficient for the Aquarobot where $N = 3$, but the AB algorithm is more efficient for the ROV/manipulator system because the number of links in the chain is greater than this crossover point.

This conclusion is correct under the assumption that the incorporation of the hydrodynamic effects adds the same amount of computation to both types of algorithms. This, however, is not the case. While, the computation of the added mass bias, drag, and total buoyancy forces does add the same amount to both algorithms, the inclusion of the added mass matrix has a very different effect on both. As will be shown later in this chapter, this matrix will be added to spatial inertias of each rigid body in the system and the special structure of the latter [shown in Eq. (1.7)] is replaced by a full symmetric matrix. In the AB algorithm, this matrix is added to AB inertia matrices which are already full symmetric matrices and the procedure has no effect on the computational requirements of subsequent operations. In the CRB algorithm, subsequent operations using this matrix depended on the special structure in Eq. (1.7) to achieve greater efficiency and compete with the AB algorithms. Transforms of the new matrix (with the added mass) require nearly twice as many operations and the resulting CRB algorithm for URV systems will always be less efficient than the AB algorithm.

Therefore, the AB method is used as the basis for the efficient URV dynamics algorithm developed in this chapter which uses the procedures for computing hydrodynamic forces developed in the previous chapter. In the next section, a derivation

of the AB dynamic equations to model systems with mobile bases and multiple serial chains is presented using the spatial notation in this dissertation. In Section 7.3, an efficient procedure for incorporating the hydrodynamics terms of Chapter 6 is presented. Then, the complete AB/DTS algorithm for general multiple chain URV systems is presented along with its computational requirements. This will be used in Chapter 8, as the basis for an efficient implementation of the dynamics for general robotic systems using an object oriented language.

This algorithm is capable of simulating both the Aquarobot and the ROV with a single manipulator. However, like the latter, many URV systems are characterized by a single serial chain. In this case, the general algorithm is not necessarily the most efficient approach. In order to obtain more efficient algorithms for these systems, two modifications to the general algorithm are examined in Section 7.5. The first is a more efficient kinematic model specifically for single chains. With additional assumptions, reasonable simplifications to the dynamic model can also be made which result in further reductions in the amount of computation. Finally, the results using these two approaches are compared to the general simulation algorithm.

7.2 Articulated Body Dynamics

In this section, the equations used in the AB method for computing robot dynamics are derived using the spatial notation presented in this dissertation. This discussion begins with an overview of the equations describing the kinematics and dynamics of one rigid body (link) within a serial chain. These are used to derive the principal equations in the algorithm to compute the inertias of articulated bodies and the

forces exerted on them. Then, additional equations are presented to model more complex systems having mobile bases with multiple serial chains attached. Finally, the efficient procedure for including gravitational effects is presented.

7.2.1 The Basics

The derivation of the AB algorithm begins with two basic equations for each link in the chain. The first is a kinematic equation to compute the spatial acceleration of each link i, $\mathbf{a}_i$. This is a function of the acceleration of the inboard link, $\mathbf{a}_{i-1}$, and the relative acceleration between the two links or the joint acceleration, $\ddot{q}_i$:

$$\mathbf{a}_i = {}^i\mathbf{X}_{i-1}\,\mathbf{a}_{i-1} + \boldsymbol{\phi}_i\ddot{q}_i + \boldsymbol{\zeta}_i, \tag{7.1}$$

where $\boldsymbol{\zeta}_i$ is the vector of coriolis and centripetal accelerations that are a function of known link velocities. For the single degree of freedom revolute and prismatic joints, this term is computed as follows:

$$\boldsymbol{\zeta}_i = \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{R}_{i-1}\left[\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{p}_i)\right] \end{bmatrix} + \begin{bmatrix} \sigma_i(\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}_i) \\ \bar{\sigma}_i(2\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}_i) \end{bmatrix}, \tag{7.2}$$

where $\sigma_i = 1$ if joint i is revolute, and $\sigma_i = 0$ if the joint is prismatic.

The second equation computes the dynamics for each link. As illustrated in Figure 24, this force balance on link i, ignoring gravity, is given as follows:

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T\mathbf{f}_{i+1} = \mathbf{I}_i\mathbf{a}_i - \boldsymbol{\beta}_i, \tag{7.3}$$

where

$$\boldsymbol{\beta}_i = -\begin{bmatrix} \boldsymbol{\omega}_i \times \bar{\boldsymbol{I}}_i\boldsymbol{\omega}_i \\ \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{h}_i) \end{bmatrix} \tag{7.4}$$

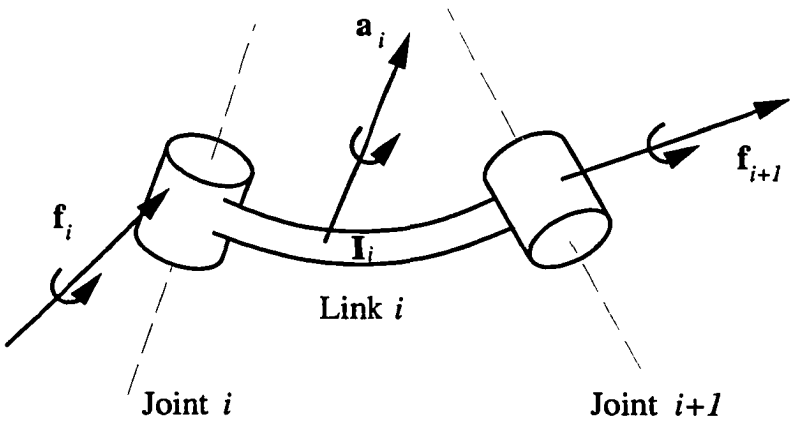


Figure 24: Rigid body dynamics for a link.

is the vector of bias forces that are a function of link velocities. The vector, $\mathbf{f}_i$, is the spatial force exerted on link i by its inboard neighbor and contains the effect of the joint force/torque, τ_i, as follows:

$$\tau_i = \phi_i^T \mathbf{f}_i, \tag{7.5}$$

which is given in the simulation problem. Likewise, $\mathbf{f}_{i+1}$ is the force exerted by link i on the outboard link and contains the effect of τ_{i+1}.

Eq. (7.1) implies that a recursion from the base of the chain to the end-effector may be used to compute the link spatial accelerations when the acceleration of the base, $\mathbf{a}_0$, is known (it is zero for a fixed base), and the joint accelerations, $\ddot{q}_i$, are known. Eq. (7.3) implies a recursion from the end-effector to the base to determine all of the link forces once the link accelerations have been determined. These two sets of equations define the outward and inward recursions associated with the *inverse* dynamics algorithm which computes the joint torques to achieve a given motion,

and is presented in Appendix B.

Taken together, Eqs. (7.1) and (7.3) also define the set of equations that need to be solved for the *forward* dynamics problem in which the joint state (position and velocity) and inputs (force/torque) are known and the joint accelerations are computed. For a serial chain with N links, they define the $12N$ equations where the $12N$ unknowns consist of the $6N$ elements of the spatial acceleration vectors, $\mathbf{a}_i$, the $5N$ unknown elements of the spatial force vectors, $\mathbf{f}_i$ (the sixth element of each vector is the known joint torque/force), and the N joint accelerations, $\ddot{q}_i$. This implies that a $12N \times 12N$ system of equations can be constructed and solved simultaneously for the unknown quantities. Featherstone has derived a much more efficient approach which uses the concept of Articulated Body (AB) inertias [15].

7.2.2 Articulated Body Inertias and Forces

The most important step in the AB algorithm involves the computation of AB inertias. Instead of using the force balance equation for a single link as in Eq. (7.3), an expression relating $\mathbf{f}_i$ to the combined dynamic properties of links i through N is desired. As illustrated in Figure 25, this relationship is given as follows:

$$\mathbf{f}_i = \mathbf{I}_i^* \mathbf{a}_i - \boldsymbol{\beta}_i^*. \tag{7.6}$$

The matrix, $\mathbf{I}_i^*$, is the ith AB inertia which is the inertia of links i through N that is "felt" at the ith coordinate system when joints $i+1$ to N are free to move (that is, the joint torques/forces are zero). Likewise, the vector, $\boldsymbol{\beta}_i^*$, is the bias force exerted on the ith link due to the resultant bias forces within the articulated body including

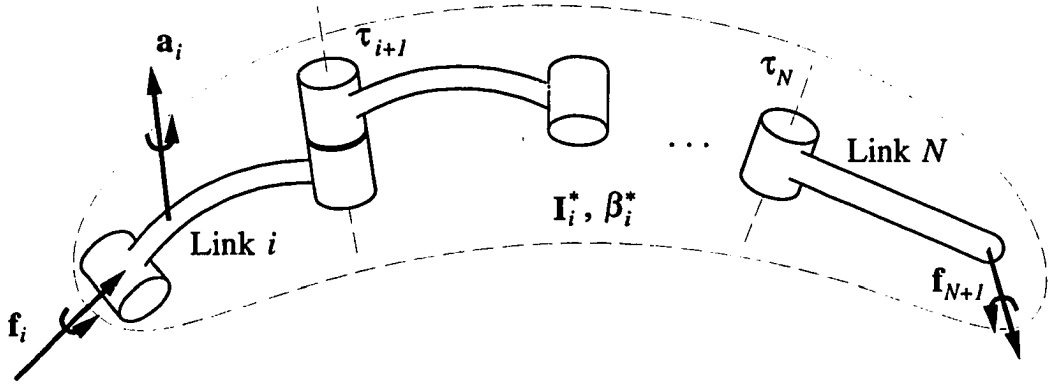


Figure 25: Articulated body dynamics.

the effects of all the joint torques/forces outboard of this link and a contact force, $\mathbf{f}_{N+1}$, exerted by the end-effector.

Equations needed to compute $\mathbf{I}_i^*$ and $\boldsymbol{\beta}_i^*$ can be found in [15, 17] and a brief derivation using spatial notation is included here. The goal of this derivation is to find expressions for $\mathbf{I}_{i-1}^*$ and $\boldsymbol{\beta}_{i-1}^*$. The first step is to substitute Eq. (7.1) into Eq. (7.6) as follows:

$$\mathbf{f}_i = \mathbf{I}_i^* \left({}^i\mathbf{X}_{i-1}\, \mathbf{a}_{i-1} + \boldsymbol{\phi}_i \ddot{q}_i + \boldsymbol{\zeta}_i \right) - \boldsymbol{\beta}_i^*. \tag{7.7}$$

This force is projected onto the joint axis as in Eq. (7.5) and solved for the unknown joint acceleration, $\ddot{q}_i$. This yields the following equation:

$$\ddot{q}_i = \left(\boldsymbol{\phi}_i^T \mathbf{I}_i^* \boldsymbol{\phi}_i \right)^{-1} \left[\tau_i + \boldsymbol{\phi}_i^T \boldsymbol{\beta}_i^* - \boldsymbol{\phi}_i^T \mathbf{I}_i^* \left({}^i\mathbf{X}_{i-1}\, \mathbf{a}_{i-1} + \boldsymbol{\zeta}_i \right) \right]. \tag{7.8}$$

This is substituted back into Eq. (7.7) to remove the unknown joint acceleration, and the equation that results can be simplified to:

$$\mathbf{f}_i = \mathbf{N}_i\, {}^i\mathbf{X}_{i-1}\, \mathbf{a}_{i-1} - \left[\boldsymbol{\beta}_i^* - \mathbf{N}_i\, \boldsymbol{\zeta}_i - \mathbf{n}_i (m_i^*)^{-1} \tau_i^* \right], \tag{7.9}$$

where

$$\mathbf{n}_i = \mathbf{I}_i^* \boldsymbol{\phi}_i, \tag{7.10}$$

$$m_i^* = \boldsymbol{\phi}_i^T \mathbf{I}_i^* \boldsymbol{\phi}_i, \tag{7.11}$$

$$\mathbf{N}_i = \mathbf{I}_i^* - \mathbf{I}_i^* \boldsymbol{\phi}_i \left(\boldsymbol{\phi}_i^T \mathbf{I}_i^* \boldsymbol{\phi}_i \right)^{-1} \boldsymbol{\phi}_i^T \mathbf{I}_i^*, \text{ and} \tag{7.12}$$

$$\tau_i^* = \tau_i + \boldsymbol{\phi}_i^T \boldsymbol{\beta}_i^*. \tag{7.13}$$

The inertia matrix, $\mathbf{N}_i$, is called the *reflected inertia* because it is the AB inertia of links i through N after it has been reflected to the inboard side of joint i when no joint torque/force is applied and the joint is free to move. The quantity, m_i^*, is the inertial load from the outboard links that is felt by the joint motor. It defines the linear, time-varying relationship between the joint torque/force and the resulting acceleration as shown in Eq. (7.8).

Finally, Eq. (7.9) is substituted into the force balance equation [Eq. (7.3)] for link $i-1$. Grouping terms to put the result into the same form as Eq. (7.6) yields the desired equation:

$$\mathbf{f}_{i-1} = \left(\mathbf{I}_{i-1} + {}^i\mathbf{X}_{i-1}^T \mathbf{N}_i \, {}^i\mathbf{X}_{i-1} \right) \mathbf{a}_{i-1} - \left\{ \boldsymbol{\beta}_{i-1} + {}^i\mathbf{X}_{i-1}^T \left[\boldsymbol{\beta}_i^* - \mathbf{N}_i \, \boldsymbol{\zeta}_i - \mathbf{n}_i (m_i^*)^{-1} \tau_i^* \right] \right\}. \tag{7.14}$$

This completes the derivation, and the recursive equations for the AB inertia and force in Eq. (7.6) are summarized as follows:

$$\mathbf{I}_{i-1}^* = \mathbf{I}_{i-1} + {}^i\mathbf{X}_{i-1}^T \, \mathbf{N}_i \, {}^i\mathbf{X}_{i-1}, \tag{7.15}$$

$$\boldsymbol{\beta}_{i-1}^* = \boldsymbol{\beta}_{i-1} + {}^i\mathbf{X}_{i-1}^T \left[\boldsymbol{\beta}_i^* - \mathbf{N}_i \, \boldsymbol{\zeta}_i - \mathbf{n}_i (m_i^*)^{-1} \tau_i^* \right]. \tag{7.16}$$

The initial values for this recursion are obtained by directly comparing Eqs. (7.3) and (7.6) for the last link. This implies that:

$$\mathbf{I}_N^* = \mathbf{I}_N, \tag{7.17}$$

$$\boldsymbol{\beta}_N^* = \boldsymbol{\beta}_N - {}^{N+1}\mathbf{X}_N^T \, \mathbf{f}_{N+1}, \tag{7.18}$$

where $\mathbf{f}_{N+1}$ is the external force, if present, that is exerted by the last link on the environment.

With expressions for the computation of the AB inertias, $\mathbf{I}_i^*$, and forces, $\boldsymbol{\beta}_i^*$, the AB dynamics algorithm to compute the joint accelerations can be summarized by examining Eq. (7.8) in detail. In this equation, $\boldsymbol{\beta}_i^*$ depends on $\boldsymbol{\beta}_i$ which, along with $\boldsymbol{\zeta}_i$, is a function of link velocities. The computation of the link velocities requires a recursion from the base of the chain to the end-effector and must be completed first. The computation of $\boldsymbol{\beta}_i$ and $\boldsymbol{\zeta}_i$ are also included in this step. Then, a backward recursion is needed to compute $\mathbf{I}_i^*$ and $\boldsymbol{\beta}_i^*$. The final forward recursion to compute the joint accelerations begins with the known base acceleration, $\mathbf{a}_0 = \mathbf{0}$, and evaluates Eqs. (7.8) and (7.1), in that order, from the base of the chain to the end-effector. Using this approach, Appendix B presents an efficient implementation of this procedure for the computation of the dynamics of a serial chain with single degree-of-freedom joints and a fixed base. For a chain with N revolute joints, the algorithm requires [$(224N - 254)$M, $(205N - 241)$A], and when $N = 6$ it requires [1090M, 989A]. This is approximately 15% less than the most efficient algorithm previously reported by Brandl, Johanni, and Otter [17].

7.2.3 Mobile Base and Multiple Chains

When simulating the URV systems discussed at the beginning of Chapter 6, the base of the chain is no longer fixed with respect to an inertial frame and the equations must be augmented to model this characteristic. In the case of the underwater legged vehicle, multiple serial chains attached to this base, resulting in a tree-structured topology, must also be modeled. In this section, the equations needed to extend the AB dynamics algorithm derived in the previous section to model these characteristics are presented. The decoupled tree-structure (DTS) approach to modeling the kinematics of these systems which was presented in Section 5.3.1 for land-based legged vehicles and illustrated in Figure 17 is also used.

Since the base is mobile, a force balance equation for this rigid body is used to relate its unknown acceleration, $\mathbf{a}_r$, with applied forces. First, Eq. (7.3) is augmented to account for multiple links attached to the reference member as follows:

$$\mathbf{f}_r - \sum_{k=1}^{m} {}^r\mathbf{f}_k = \mathbf{I}_r\,\mathbf{a}_r - \boldsymbol{\beta}_r, \tag{7.19}$$

where $\mathbf{f}_r$ is an external force applied to the reference member and could correspond to thruster forces, and ${}^r\mathbf{f}_k$ is the force exerted by the reference member onto the first link of chain k. It is important to note that ${}^r\mathbf{f}_k$ is the same as $\mathbf{f}_1$ (from the previous section), which has been transformed to the reference member's coordinate system. In this section, $\mathbf{f}_1$ is referred to as $\mathbf{f}_{k,1}$ to distinguish between multiple chains, and ${}^r\mathbf{f}_k$ is a shortened notation for ${}^r\mathbf{f}_{k,1}$.

When using the DTS kinematic modeling, the transformation between the first link of a chain and the reference member is most efficiently implemented with a full

set of MDH parameters (planar x and z transformations) from the first link's coordinate system to a base coordinate system attached to the reference member, labeled $(k,0)$, and then a z-planar transformation to the reference member's coordinate system. Therefore, the force balance equation is rewritten as follows:

$$\mathbf{f}_r - \sum_{k=1}^{m} {}^0\mathbf{X}_{k,r}^T \, {}^1\mathbf{X}_{k,0}^T \, \mathbf{f}_{k,1} \;=\; \mathbf{I}_r \, \mathbf{a}_r - \boldsymbol{\beta}_r, \tag{7.20}$$

where ${}^1\mathbf{X}_{k,0}^T$ is the spatial transformation defined by a full set of MDH parameters and ${}^0\mathbf{X}_{k,r}^T$ is the additional z-planar transformation.

The goal of this section is to derive an expression for the reference member's acceleration, $\mathbf{a}_r$, in terms of known quantities. Therefore, the force vectors, $\mathbf{f}_{k,1}$, must be replaced with functions of $\mathbf{a}_r$ and known quantities. The desired expression for this force can be derived from Eq. (7.9) which is rewritten here for the first link of chain k as follows:

$$\mathbf{f}_{k,1} \;=\; \mathbf{N}_{k,1} \, {}^1\mathbf{X}_{k,0} \, \mathbf{a}_{k,0} - \mathbf{f}_{k,1}^*, \tag{7.21}$$

where

$$\mathbf{f}_{k,1}^* \;=\; \boldsymbol{\beta}_{k,1}^* - \mathbf{N}_{k,1} \, \boldsymbol{\zeta}_{k,1} - \mathbf{n}_{k,1} (m_{k,1}^*)^{-1} \tau_{k,1}^*. \tag{7.22}$$

This is the component of the force exerted on the link which, through recursion, is a known function of the state of chain k, its joint forces/torques, and the contact force on the tip of the chain. The vector, $\mathbf{a}_{k,0}$, is the spatial acceleration of the base coordinate system of chain k. This is a point fixed to the reference member, so this acceleration can be written in terms of $\mathbf{a}_r$ (shown in Section 5.3.1) as follows:

$$\mathbf{a}_{k,0} \;=\; {}^0\mathbf{X}_{k,r} \, \mathbf{a}_r + \boldsymbol{\zeta}_{k,0}, \tag{7.23}$$

where $\boldsymbol{\zeta}_{k,0}$ is defined similar to Eq. (7.2) as:

$$\boldsymbol{\zeta}_{k,0} = \begin{bmatrix} \mathbf{0} \\ {}^{0}\boldsymbol{R}_{k,r}\left[\boldsymbol{\omega}_r \times \left(\boldsymbol{\omega}_r \times {}^{r}\boldsymbol{p}_{k,0}\right)\right] \end{bmatrix}. \tag{7.24}$$

Therefore, $\mathbf{f}_{k,1}$ is written in terms of $\mathbf{a}_r$ as follows:

$$\mathbf{f}_{k,1} = \mathbf{N}_{k,1}\,{}^{1}\mathbf{X}_{k,0}\left({}^{0}\mathbf{X}_{k,r}\,\mathbf{a}_r + \boldsymbol{\zeta}_{k,0}\right) - \mathbf{f}^{*}_{k,1}. \tag{7.25}$$

Finally, this is substituted into Eq. (7.20) and the terms multiplying $\mathbf{a}_r$ are grouped together as follows:

$$\begin{aligned} \mathbf{f}_r = & \left[\mathbf{I}_r + \sum_{k=1}^{m} {}^{0}\mathbf{X}^{T}_{k,r}\left({}^{1}\mathbf{X}^{T}_{k,0}\mathbf{N}_{k,1}\,{}^{1}\mathbf{X}_{k,0}\right){}^{0}\mathbf{X}_{k,r}\right]\mathbf{a}_r - \\ & \boldsymbol{\beta}_r - \sum_{k=1}^{m} {}^{0}\mathbf{X}^{T}_{k,r}\left[{}^{1}\mathbf{X}^{T}_{k,0}\mathbf{f}^{*}_{k,1} - \left({}^{1}\mathbf{X}^{T}_{k,0}\mathbf{N}_{k,1}\,{}^{1}\mathbf{X}_{k,0}\right)\boldsymbol{\zeta}_{k,0}\right], \end{aligned} \tag{7.26}$$

which completes the derivation of the equation of motion for the mobile base.

Since $\mathbf{f}_r$, if present, is given in the forward dynamics problem, the AB form of the equation of motion to compute the acceleration of the mobile base in the equation above can be expressed as follows:

$$\mathbf{a}_r = \left(\mathbf{I}^{*}_r\right)^{-1}\left(\mathbf{f}_r + \boldsymbol{\beta}^{*}_r\right), \tag{7.27}$$

which was also stated by Featherstone ([27], p. 122). The needed quantities, $\mathbf{I}^{*}_r$ and $\boldsymbol{\beta}^{*}_r$, can be determined given the state of the system and joint inputs which are taken from Eq. (7.26) as follows:

$$\mathbf{I}^{*}_r = \mathbf{I}_r + \sum_{k=1}^{m} {}^{0}\mathbf{X}^{T}_{k,r}\left({}^{1}\mathbf{X}^{T}_{k,0}\mathbf{N}_{k,1}\,{}^{1}\mathbf{X}_{k,0}\right){}^{0}\mathbf{X}_{k,r}, \tag{7.28}$$

which is the AB inertia of the entire system and expressed with respect to the reference member's coordinate system, and

$$\boldsymbol{\beta}^{*}_r = \boldsymbol{\beta}_r + \sum_{k=1}^{m} {}^{0}\mathbf{X}^{T}_{k,r}\left[{}^{1}\mathbf{X}^{T}_{k,0}\mathbf{f}^{*}_{k,1} - \left({}^{1}\mathbf{X}^{T}_{k,0}\mathbf{N}_{k,1}\,{}^{1}\mathbf{X}_{k,0}\right)\boldsymbol{\zeta}_{k,0}\right], \tag{7.29}$$

which is the resultant bias force exerted on the reference member from the chains in the system.

7.2.4 Gravity

Up to this point, gravitational effects have been ignored. They can be included in the previous derivations in two ways. Conceptually, the easiest method is to add a gravitational force term to the force balance equation, Eq. (7.3), for each rigid body as follows:

$$\mathbf{f}_i + \mathbf{f}_i^g - {}^{i+1}\mathbf{X}_i^T \mathbf{f}_{i+1} = \mathbf{I}_i \mathbf{a}_i - \boldsymbol{\beta}_i. \tag{7.30}$$

The force due to gravity can be computed for the body as follows:

$$\mathbf{f}_i^g = \mathbf{I}_i {}^i\mathbf{a}_g, \tag{7.31}$$

where ${}^i\mathbf{a}_g$ is the gravitational acceleration expressed in the body's coordinate system and consists of only a translational component:

$${}^i\mathbf{a}_g = \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{a}_g \end{bmatrix}. \tag{7.32}$$

The simplest way to incorporate this into the previous derivations is to redefine $\boldsymbol{\beta}_i$ to include this force, and the appearance of the previous derivations will remain unchanged. From a computational efficiency standpoint, the disadvantage to this approach is that the gravitational acceleration must be determined with respect to each rigid body's coordinate system.

The alternate approach, which is also used in efficient inverse dynamics algorithms, is to bias the spatial acceleration of each link by negative gravitational acceleration. This is possible by substituting Eq. (7.31) into Eq (7.30) and collecting

the terms multiplying the spatial inertia:

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T \mathbf{f}_{i+1} = \mathbf{I}_i \left(\mathbf{a}_i - {}^i\mathbf{a}_g \right) - \boldsymbol{\beta}_i \tag{7.33}$$

$$= \mathbf{I}_i \mathbf{a}'_i - \boldsymbol{\beta}_i. \tag{7.34}$$

This becomes the replacement for Eq. (7.3) in the previous derivations. The propagation of this acceleration is accomplished with Eq. (7.1):

$$\mathbf{a}'_i = {}^i\mathbf{X}_{i-1}\, \mathbf{a}'_{i-1} + \boldsymbol{\phi}_i \ddot{q}_i + \boldsymbol{\zeta}_i. \tag{7.35}$$

Therefore, the previous derivations proceed as before except every occurrence of spatial acceleration, $\mathbf{a}$, is changed to the biased spatial acceleration, $\mathbf{a}'$.

The only change made to the single chain algorithm with a fixed base is to bias the zero base acceleration as follows:

$$\mathbf{a}'_0 = - \begin{bmatrix} \mathbf{0} \\ {}^0\boldsymbol{a}_g \end{bmatrix}. \tag{7.36}$$

Since the base coordinate system is fixed, this quantity is constant and can be computed off line. The rest of the algorithm remains unchanged and is presented in Appendix B.

For the multiple chain system, the reference member acceleration computation in Eq. (7.27) will now yield a biased result:

$$\mathbf{a}'_r = \left(\mathbf{I}_r^*\right)^{-1} \left(\mathbf{f}_r + \boldsymbol{\beta}_r^*\right). \tag{7.37}$$

This result is used in the Forward Acceleration recursion for each chain to compute joint accelerations. For the purpose of simulation, the unbiased reference member

acceleration must also be computed as follows:

$$\mathbf{a}_r = \mathbf{a}'_r + \begin{bmatrix} \mathbf{0} \\ {}^r\boldsymbol{R}_e\, {}^e\boldsymbol{a}_g \end{bmatrix}, \tag{7.38}$$

where ${}^e\boldsymbol{a}_g$ is the constant Cartesian vector of gravitational acceleration expressed in the earth (inertial) coordinate system, and ${}^r\boldsymbol{R}_e$ is the rotation matrix describing the orientation of the reference member with respect to the inertial coordinate system.

7.2.5 Algorithm Summary

Finally, the development of the dynamics algorithm for a system with multiple chains and a mobile base on land can be completed. It uses the single chain algorithm in Appendix B which is performed for each serial chain that is attached to the reference member. However, the derivation for the mobile base implies a number of changes that must be made to the three recursions in this algorithm.

Before the Forward Kinematics recursion is executed for each chain, the angular velocity of the reference member, $\boldsymbol{\omega}_r$, must be transformed to the base coordinate system of each chain to obtain $\boldsymbol{\omega}_{k,0}$. This quantity becomes the starting condition of this recursion, and replaces $\boldsymbol{\omega}_0 = \mathbf{0}$ in the fixed base algorithm. The vector, $\boldsymbol{\zeta}_{k,0}$, from Eq. (7.24) is added to the beginning of this recursion for each chain, and the computation of $\boldsymbol{\beta}_r$ is performed before these recursions.

The end of the Backward Dynamics recursion for each chain must include the necessary computations to obtain $\mathbf{I}^*_r$ and $\boldsymbol{\beta}^*_r$. The former involves performing congruence transformations on the reflected inertia, $\mathbf{N}_1$, to express it in the reference member's coordinate system and shown in the rightmost term on Eq. (7.28). Com-

putation of β_r^* requires the computation of the bias forces exerted on the reference member from each chain as expressed in the Eq. (7.29).

Before the Forward Accelerations recursion for each chain can proceed, the biased reference member acceleration, $\mathbf{a}'_r$, must be computed as shown by Eq. (7.37). Then, $\mathbf{a}'_{k,0}$ from Eq. (7.23) must be determined which becomes the initial value of this recursion for each chain. In addition, the unbiased reference member acceleration, $\mathbf{a}_r$, from Eq. (7.38) must be determined.

The resulting algorithm is called the Articulated Body/Decoupled Tree-Structure (AB/DTS) algorithm which was first developed by Freeman and Orin [7]. A detailed listing of this algorithm is not included here but can be easily extracted from the version with hydrodynamics which is developed in the next section and listed in Tables 13–15. The computational requirements for the more efficient algorithm derived here is shown in Table 16. For an m-chain system with N degrees of freedom per chain, the algorithm requires [$(224mN+59m+99)$M, $(205mN+63m+68)$A]. For the land-based, legged vehicle discussed in Chapter 5, with $m = 6$ and $N = 3$, this corresponds to [4485M, 4136A] which is slightly more than the CRB/DTS algorithm. The AB/DTS algorithm is more efficient when $N > 3$.

7.3 Incorporating Hydrodynamic Effects

The final and most important step in the derivation of the underwater dynamics algorithm for URV systems is to incorporate the computation of hydrodynamic forces on a single rigid body that was developed in the previous chapter into the algorithm developed above. First, the spatial force balance equation for a single link

that is submerged in a fluid is developed which includes these hydrodynamic forces. The differences between this and the equation used in the previous derivation provide the information necessary to modify the AB algorithm to simulate URV systems.

Including the link-to-link forces, gravity and hydrodynamic effects, the force balance on link i can be written as follows:

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T\, \mathbf{f}_{i+1} + \mathbf{f}_i^A + \mathbf{f}_i^D + \mathbf{f}_i^{TB} \;=\; \mathbf{I}_i\, \mathbf{a}_i' - \boldsymbol{\beta}_i, \tag{7.39}$$

where, again:

$$\mathbf{a}_i' \;=\; \mathbf{a}_i - {}^i\mathbf{a}_g. \tag{7.40}$$

The added mass force, $\mathbf{f}_i^A$, is also a function of the unknown link acceleration, $\mathbf{a}_i$. Substituting in from Eq. (6.20) and rearranging the terms leads to the following equation:

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T\, \mathbf{f}_{i+1} \;=\; \mathbf{I}_i\mathbf{a}_i' + \mathbf{I}_i^A\mathbf{a}_i - \boldsymbol{\beta}_i - \boldsymbol{\beta}_i^A - \mathbf{f}_i^D - \mathbf{f}_i^{TB}. \tag{7.41}$$

Since the efficient AB algorithm is implemented using the biased spatial acceleration, $\mathbf{a}_i'$, this equation must be modified to remove $\mathbf{a}_i$. This is accomplished by substituting for $\mathbf{a}_i$ from Eq. (7.40):

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T\, \mathbf{f}_{i+1} \;=\; \mathbf{I}_i\mathbf{a}_i' + \mathbf{I}_i^A\left(\mathbf{a}_i' + {}^i\mathbf{a}_g\right) - \boldsymbol{\beta}_i - \boldsymbol{\beta}_i^A - \mathbf{f}_i^D - \mathbf{f}_i^{TB}. \tag{7.42}$$

The terms multiplying the biased acceleration are combined and the new term, $\mathbf{I}_i^A\, {}^i\mathbf{a}_g$, is most efficiently included with the added mass bias force, $\boldsymbol{\beta}_i^A$, from Eq. (6.21). The resulting equation is given as:

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T\, \mathbf{f}_{i+1} \;=\; \left(\mathbf{I}_i + \mathbf{I}_i^A\right)\mathbf{a}_i' - \boldsymbol{\beta}_i - \boldsymbol{\beta}_i^{A'} - \mathbf{f}_i^D - \mathbf{f}_i^{TB}. \tag{7.43}$$

where

$$\boldsymbol{\beta}_i^{A'} = \boldsymbol{\beta}_i^A - \mathbf{I}_i^A \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{a}_g \end{bmatrix} \tag{7.44}$$

$$= \mathbf{I}_i^A \begin{bmatrix} \mathbf{0} \\ ({}^i\boldsymbol{a}_f - {}^i\boldsymbol{a}_g) + \boldsymbol{\omega}_i \times \boldsymbol{v}_i^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i & \tilde{\boldsymbol{v}}_i^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_i \end{bmatrix} \mathbf{I}_i^A \begin{bmatrix} \boldsymbol{\omega}_i \\ \boldsymbol{v}_i^r \end{bmatrix}. \tag{7.45}$$

Combining the terms in this fashion results in no additional computation because the quantity, ${}^i\boldsymbol{a}_f - {}^i\boldsymbol{a}_g$, is already obtained in the computation of the total buoyancy force, $\mathbf{f}_i^{TB}$.

Rewriting the equation to resemble Eq. (7.33) leads to the following hydrodynamic equation of motion for the link:

$$\mathbf{f}_i - {}^{i+1}\mathbf{X}_i^T \mathbf{f}_{i+1} = \mathbf{I}_i^H \mathbf{a}_i' - \boldsymbol{\beta}_i^H, \tag{7.46}$$

where

$$\mathbf{I}_i^H = \mathbf{I}_i + \mathbf{I}_i^A, \text{ and} \tag{7.47}$$

$$\boldsymbol{\beta}_i^H = \boldsymbol{\beta}_i + \boldsymbol{\beta}_i^{A'} + \mathbf{f}_i^D + \mathbf{f}_i^{TB}. \tag{7.48}$$

The consequence of this equation is that the spatial acceleration of the body and the resultant force are now linearly related through the sum of the rigid body's spatial inertia and the added mass of the fluid surrounding the body. The hydrodynamic AB algorithm can be derived as before by using Eq. (7.46) in place of Eq. (7.33) [or Eq. (7.3)]. Since the form of this equation is the same, the derivation will proceed as before, and the resulting efficient algorithm is listed in Tables 13–15 for multiple chain URVs.

A comparison of Eqs. (7.46) and (7.33) indicates two changes that are to be made to the resulting algorithm to incorporate the hydrodynamic effects. First, the use of

Table 13: Articulated Body algorithm for multiple-chain URV systems.

I. Forward Kinematics Given: ${}^r\boldsymbol{R}_e$, $\boldsymbol{\omega}_r$, $\boldsymbol{v}_r$, ${}^e\boldsymbol{v}_f$, ${}^e\boldsymbol{a}_f$.

$$
\begin{aligned}
\boldsymbol{v}_r^r &= \boldsymbol{v}_r - {}^r\boldsymbol{R}_e{}^e\boldsymbol{v}_f \\
{}^r\boldsymbol{a}_g &= {}^r\boldsymbol{R}_e(g\hat{\mathbf{z}}_e) \qquad \text{where } \hat{\mathbf{z}}_e = [0\ 0\ 1]^T \\
{}^r\boldsymbol{a}_{f-g} &= {}^r\boldsymbol{R}_e{}^e\boldsymbol{a}_f - {}^r\boldsymbol{a}_g \\
\boldsymbol{\beta}_r^H &= -\begin{bmatrix} \boldsymbol{\omega}_r \times \bar{\mathbf{I}}_r\boldsymbol{\omega}_r \\ \mathbf{0} \end{bmatrix} + \mathbf{f}_r^D(\boldsymbol{\omega}_r, \boldsymbol{v}_r^r) + m_r^f \begin{bmatrix} \boldsymbol{b}_r \times {}^r\boldsymbol{a}_{f-g} \\ {}^r\boldsymbol{a}_{f-g} \end{bmatrix} + \\
&\qquad \mathbf{I}_r^A \begin{bmatrix} \mathbf{0} \\ {}^r\boldsymbol{a}_{f-g} + \boldsymbol{\omega}_r \times \boldsymbol{v}_r^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_r & \tilde{\boldsymbol{v}}_r^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_r \end{bmatrix} \mathbf{I}_r^A \begin{bmatrix} \boldsymbol{\omega}_r \\ \boldsymbol{v}_r^r \end{bmatrix}
\end{aligned}
$$

for all $k = 1, \ldots, m$. Given: ${}^0\boldsymbol{R}_{k,r}$, ${}^r\boldsymbol{p}_{k,0}$.

$$
\begin{aligned}
\begin{bmatrix} \boldsymbol{\omega}_{k,0} \\ \boldsymbol{v}_{k,0}^r \end{bmatrix} &= {}^0\mathbf{X}_{k,r} \begin{bmatrix} \boldsymbol{\omega}_r \\ \boldsymbol{v}_r^r \end{bmatrix} \\
{}^0\boldsymbol{a}_{k,f-g} &= {}^0\boldsymbol{R}_{k,r}{}^r\boldsymbol{a}_{f-g} \\
\boldsymbol{\zeta}_{k,0} &= \begin{bmatrix} \mathbf{0} \\ {}^0\boldsymbol{R}_{k,r}\left[\boldsymbol{\omega}_r \times \left(\boldsymbol{\omega}_r \times {}^r\boldsymbol{p}_{k,0}\right)\right] \end{bmatrix}
\end{aligned}
$$

for $i = 1, \ldots, N$. Given: ${}^i\boldsymbol{R}_{k,i-1}$, ${}^{i-1}\boldsymbol{p}_{k,i}$, $\dot{q}_{k,i}$.

$$
\begin{aligned}
\begin{bmatrix} \boldsymbol{\omega}_{k,i} \\ \boldsymbol{v}_{k,i}^r \end{bmatrix} &= {}^i\mathbf{X}_{k,i-1} \begin{bmatrix} \boldsymbol{\omega}_{k,i-1} \\ \boldsymbol{v}_{k,i-1}^r \end{bmatrix} + \boldsymbol{\phi}_{k,i}\dot{q}_{k,i} \\
{}^i\boldsymbol{a}_{k,f-g} &= {}^i\boldsymbol{R}_{k,i-1}{}^{i-1}\boldsymbol{a}_{k,f-g} \\
\boldsymbol{\zeta}_{k,i} &= \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{R}_{k,i-1}\left[\boldsymbol{\omega}_{k,i-1} \times \left(\boldsymbol{\omega}_{k,i-1} \times {}^{i-1}\boldsymbol{p}_{k,i}\right)\right] \end{bmatrix} + \begin{bmatrix} \sigma_{k,i}(\boldsymbol{\omega}_{k,i} \times \dot{q}_{k,i}\hat{\mathbf{z}}_{k,i}) \\ \bar{\sigma}_{k,i}(2\boldsymbol{\omega}_{k,i} \times \dot{q}_{k,i}\hat{\mathbf{z}}_{k,i}) \end{bmatrix} \\
\boldsymbol{\beta}_{k,i}^H &= -\begin{bmatrix} \boldsymbol{\omega}_{k,i} \times \bar{I}_{k,i}\boldsymbol{\omega}_{k,i} \\ \boldsymbol{\omega}_{k,i} \times (\boldsymbol{\omega}_{k,i} \times \boldsymbol{h}_{k,i}) \end{bmatrix} + \mathbf{f}_{k,i}^D(\boldsymbol{\omega}_{k,i}, \boldsymbol{v}_{k,i}^r) + m_{k,i}^f \begin{bmatrix} \boldsymbol{b}_{k,i} \times {}^i\boldsymbol{a}_{k,f-g} \\ {}^i\boldsymbol{a}_{k,f-g} \end{bmatrix} \\
&\qquad + \mathbf{I}_{k,i}^A \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{a}_{k,f-g} + \boldsymbol{\omega}_{k,i} \times \boldsymbol{v}_{k,i}^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_{k,i} & \tilde{\boldsymbol{v}}_{k,i}^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_{k,i} \end{bmatrix} \mathbf{I}_{k,i}^A \begin{bmatrix} \boldsymbol{\omega}_{k,i} \\ \boldsymbol{v}_{k,i}^r \end{bmatrix}
\end{aligned}
$$

end for i.

end for all k.

Table 14: Articulated Body algorithm for multiple-chain URV systems (continued).

II. Backward Dynamics

for all $k = 1, \ldots, m$. Given: ${}^{N}\mathbf{N}_{k,N+1} = \mathbf{0}$, ${}^{N}\mathbf{f}^{*}_{k,N+1} = -{}^{N+1}\mathbf{X}^{T}_{N}\,\mathbf{f}_{N+1}$ (tip forces).

for $i = N, .., 1$. Given: $\tau_{k,i}$.

$$
\begin{aligned}
\mathbf{I}^{*}_{k,i} &= \underline{\mathbf{I}^{H}_{k,i}} + {}^{i}\mathbf{N}_{k,i+1} \\
\mathbf{n}_{k,i} &= \mathbf{I}^{*}_{k,i}\boldsymbol{\phi}_{k,i} \\
m^{*}_{k,i} &= \boldsymbol{\phi}^{T}_{k,i}\mathbf{I}^{*}_{k,i}\boldsymbol{\phi}_{k,i} \\
\mathbf{N}_{k,i} &= \mathbf{I}^{*}_{k,i} - \mathbf{n}_{k,i}(m^{*}_{k,i})^{-1}\mathbf{n}^{T}_{k,i} \\
\boldsymbol{\beta}^{*}_{k,i} &= \underline{\boldsymbol{\beta}^{H}_{k,i}} + {}^{i}\mathbf{f}^{*}_{k,i+1} \\
\tau^{*}_{k,i} &= \boldsymbol{\phi}^{T}_{k,i}\boldsymbol{\beta}^{*}_{k,i} + \tau_{k,i} \\
{}^{i-1}\mathbf{N}_{k,i} &= {}^{i}\mathbf{X}^{T}_{k,i-1}\mathbf{N}_{k,i}{}^{i}\mathbf{X}_{k,i-1} \\
{}^{i-1}\mathbf{f}^{*}_{k,i} &= {}^{i}\mathbf{X}^{T}_{k,i-1}\left[\boldsymbol{\beta}^{*}_{k,i} - \mathbf{N}_{k,i}\boldsymbol{\zeta}_{k,i} - \mathbf{n}_{k,i}(m^{*}_{k,i})^{-1}\tau^{*}_{k,i}\right]
\end{aligned}
$$

end for i.

$$
\begin{aligned}
{}^{r}\mathbf{N}_{k,1} &= {}^{0}\mathbf{X}^{T}_{k,r}\,{}^{0}\mathbf{N}_{k,1}{}^{0}\mathbf{X}_{k,r} \\
{}^{r}\mathbf{f}^{*}_{k,1} &= {}^{0}\mathbf{X}^{T}_{k,r}\left({}^{0}\mathbf{f}^{*}_{k,1} - {}^{0}\mathbf{N}_{k,1}\boldsymbol{\zeta}_{k,0}\right)
\end{aligned}
$$

end for all k.

---

each link's spatial inertia, $\mathbf{I}_{k,i}$, is replaced with its "hydrodynamic inertia," $\mathbf{I}^{H}_{k,i}$. The second change replaces $\boldsymbol{\beta}_{k,i}$ with $\boldsymbol{\beta}^{H}_{k,i}$. Both of these steps have been underlined in the Backward Dynamics recursion of Table 14, and in the computation of $\mathbf{a}'_{r}$ in the Forward Acceleration recursion of Table 15. The use of the alternative terms has little effect on the computation requirements of these recursions of the AB algorithm as will be shown in the next section.

Table 15: Articulated Body algorithm for multiple-chain URV systems (continued).

III. Forward Accelerations Given: $\mathbf{f}_r$.

$$\mathbf{a}'_r = \left[\underline{\mathbf{I}}_r^H + \sum_{k=1}^{m} {}^r\mathbf{N}_{k,1}\right]^{-1} \left[\mathbf{f}_r + \underline{\boldsymbol{\beta}}_r^H + \sum_{k=1}^{m} {}^r\mathbf{f}^*_{k,1}\right]$$

$$\mathbf{a}_r = \mathbf{a}'_r + \begin{bmatrix} \mathbf{0} \\ {}^r\boldsymbol{a}_g \end{bmatrix}$$

for all $k = 1, \ldots, m$. Given: ${}^0\mathbf{X}_{k,r}$.

$$\mathbf{a}'_{k,0} = {}^0\mathbf{X}_{k,r}\mathbf{a}'_r + \boldsymbol{\zeta}_{k,0}$$

for $i = 1, \ldots, N$. Given: ${}^i\mathbf{X}_{k,i-1}$.

$$\begin{aligned} \bar{\mathbf{a}}_{k,i} &= {}^i\mathbf{X}_{k,i-1}\mathbf{a}'_{k,i-1} + \boldsymbol{\zeta}_{k,i} \\ \ddot{q}_{k,i} &= (m^*_{k,i})^{-1}\tau^*_{k,i} - \left[\mathbf{n}_{k,i}(m^*_{k,i})^{-1}\right]^T \bar{\mathbf{a}}_{k,i} \\ \mathbf{a}'_{k,i} &= \bar{\mathbf{a}}_{k,i} + \boldsymbol{\phi}_{k,i}\ddot{q}_{k,i} \end{aligned}$$

end for i.

end for all k.

It is the computation of $\boldsymbol{\beta}^H_{k,i}$ itself that adds a significant amount to the Forward Kinematics recursion in Table 13. In this recursion, the computation of $\boldsymbol{\beta}_{k,i}$ is replaced with the computation of $\boldsymbol{\beta}^H_{k,i}$. While this equation still computes $\boldsymbol{\beta}_{k,i}$ (the first term of the equation in the table), the hydrodynamic forces and added mass bias term are also included. Computation of the total buoyancy force, $\mathbf{f}^{TB}_{k,i}$, requires computation of ${}^i\boldsymbol{a}_f - {}^i\boldsymbol{a}_g$. This term is most efficiently computed by performing the subtraction once at the reference member coordinate system and then transforming

a single vector (referred to as ${}^{i}\boldsymbol{a}_{k,f-g}$ in the table) to successive coordinate systems. The computation of the drag force, $\mathbf{f}^{D}_{k,i}$, requires the computation of the linear velocity of the body relative to the fluid, $\boldsymbol{v}^{r}_{k,i}$. This is combined with the angular velocity of the bodies, and the resulting spatial velocity vector is transformed in one operation, rather than transforming the two separately. Computation of the added mass bias force, $\boldsymbol{\beta}^{A'}_{i}$, requires both of these terms. To obtain them with respect to each link's coordinate system, their computation is most efficiently incorporated into the Forward Kinematics recursion as shown in Table 13.

7.4 Computational Requirements

In this section the computational requirements for this implementation of the AB/DTS algorithm are determined. Both land-based and underwater versions are examined, and the results are listed in Table 16. In the top section of the table, the requirements for the Forward Kinematics recursion are listed. In this step, velocity and acceleration terms are determined with respect to the reference member's coordinate system and then this computation propagates along each chain. Terms dependent on these quantities are also computed for each body in the system.

In this implementation, the orientation of the reference member is specified by three Euler angles [60], which is represented by the rotation matrix, ${}^{r}\boldsymbol{R}_{e}$. To achieve the most efficient transformation of these quantities, this is stored as three separate planar rotations which requires no multiplications or additions to compute (only sines and cosines of each angle). With this form, the rotations of Cartesian vectors from inertial to body-fixed frames is accomplished most efficiently with three succes-

sive planar rotations each requiring [4M, 2A] (see Appendix A). For the underwater algorithm, the vector rotations of fluid velocity and acceleration cost [12M, 6A] each, and therefore, the cost of computing $\boldsymbol{v}_r^r$ or ${}^r\boldsymbol{a}_{f-g}$ is [12M, 9A]. Both versions of the algorithm require the computation of the gravitational acceleration, ${}^r\boldsymbol{a}_g$, which only requires [4M, 0A] when using Z-Y-X Euler angles with the inertial z-axis pointing "down."

Finally, $\boldsymbol{\beta}_r$ or $\boldsymbol{\beta}_r^H$, for the land-based or underwater algorithm, respectively, can be computed. Using the DTS kinematic modeling, the reference member coordinate system is coincident with its principal axes which results in a diagonal spatial inertia matrix and a simpler computation ([9M, 3A]) for $\boldsymbol{\beta}_r$ because $\boldsymbol{h}_r$ is zero and $\bar{\boldsymbol{I}}_r$ is diagonal. This is also the first term of $\boldsymbol{\beta}_r^H$, and to complete its computation requires an additional [133M, 107A] for the hydrodynamic terms.

Computational requirements for $\boldsymbol{\omega}_{k,0}$, $\boldsymbol{v}_{k,0}^r$, ${}^0\boldsymbol{a}_{k,f-g}$, and $\boldsymbol{\zeta}_{k,0}$ are straightforward, realizing that the transformation between the reference member coordinate system and the base coordinate system for each chain involves a single z-planar screw which also implies that ${}^r\boldsymbol{p}_{k,0}$ has one nonzero element. Compared to the fixed base algorithm in the appendix, the angular velocity, $\boldsymbol{\omega}_{k,0}$, is now the starting value for the recursion along the chains. It is no longer zero in the case of a mobile base; therefore, all of the reductions in computation in the first few iterations of the fixed base algorithm, because of these zero elements, cannot be utilized here. Finally, $\boldsymbol{\beta}_{k,i}$ requires [27M, 15A], and the computation of $\boldsymbol{\beta}_{k,i}^H$ requires an additional [133M, 110A] which accounts for about 90% of the additional computation for the underwater algorithm.

The computation of additional velocity and acceleration terms account for nearly all of the rest.

The second section of the table lists the computational requirements for the Backward Dynamics recursion. Most of the numbers for this recursion are easily derived from the single chain algorithm in Appendix B; however, the computation in this step has been substantially rearranged from the fixed base algorithm in the appendix. Many of the computational requirements are still the same. Since ${}^{N}\mathbf{N}_{k,N+1} = \mathbf{0}$, this implies that $\mathbf{I}^{*}_{k,N}$, $\mathbf{n}_{k,N}$, $m^{*}_{k,N}$, $\mathbf{N}_{k,N}$, $(m^{*}_{k,N})^{-1}$ and $\mathbf{n}_{k,N}\,(m^{*}_{k,N})^{-1}$ are all constant and can be computed off-line. Unlike the fixed base algorithm, $\mathbf{N}_{k,1}$, ${}^{0}\mathbf{N}_{k,1}$, and ${}^{0}\mathbf{f}^{*}_{k,1}$ have been added to this recursion. To complete the computation the latter two must also be transformed to the reference member coordinate system. The constant z-planar congruence transform of ${}^{0}\mathbf{N}_{k,1}$ was shown in Appendix A to require [42M, 43A], and the computation of ${}^{r}\mathbf{f}^{*}_{k,1}$ requires [28M, 24A] per chain.

Computationally, the only difference in this recursion between land-based and the underwater versions of the algorithms involves the computation of the AB inertias, $\mathbf{I}^{*}_{k,i}$. The addition to obtain $\mathbf{I}^{H}_{k,i}$ does not add to the amount of computation because both components are constant and the sum is computed off-line. However, the result is a full symmetric matrix which contains six more nonzero elements in the upper triangular portion than the spatial rigid body inertia, $\mathbf{I}_{k,i}$ which increases the requirements to compute $\mathbf{I}^{*}_{k,i}$ by as many additions. Since $\mathbf{I}^{*}_{k,N}$ is also a full symmetric matrix, $\mathbf{N}_{k,N}$ no longer has two extra zero elements, and the additional efficiencies found in the computation of ${}^{N-1}\mathbf{N}_{k,N}$ and ${}^{N-1}\mathbf{f}^{*}_{k,N}$ cannot be implemented.

Table 16: Computational requirements of the multiple-chain URV simulation algorithm.

		Land Based		Underwater	
		×	+	×	+
I.	$\boldsymbol{v}_r^r$	—	—	12	9
	${}^r\boldsymbol{a}_g$	4	—	4	—
	${}^r\boldsymbol{a}_{f-g}$	—	—	12	9
	$\boldsymbol{\beta}_r/\boldsymbol{\beta}_r^H$	9	3	142	110
	$\boldsymbol{\omega}_{k,0}/\mathbf{v}_{k,0}^r$	$4m$	$2m$	$10m$	$6m$
	${}^0\boldsymbol{a}_{k,f-g}$	—	—	$4m$	$2m$
	$\boldsymbol{\zeta}_{k,0}$	$10m$	$3m$	$10m$	$3m$
	$\boldsymbol{\omega}_{k,i}/\mathbf{v}_{k,i}^r$	$8mN$	$5mN$	$20mN$	$13mN$
	${}^i\boldsymbol{a}_{k,f-g}$	—	—	$8mN$	$4mN$
	$\boldsymbol{\zeta}_{k,i}$	$22mN$	$10mN$	$22mN$	$10mN$
	$\boldsymbol{\beta}_{k,i}/\boldsymbol{\beta}_{k,i}^H$	$27mN$	$15mN$	$160mN$	$125mN$
II.	$\mathbf{I}_{k,i}^*$	—	$15m(N-1)$	—	$21m(N-1)$
	$\mathbf{n}_{k,i}$	—	—	—	—
	$(m_{k,i}^*)^{-1}$	$m(N-1)$	—	$m(N-1)$	—
	$\mathbf{n}_{k,i}\,(m_{k,i}^*)^{-1}$	$5m(N-1)$	—	$5m(N-1)$	—
	$\mathbf{N}_{k,i}$	$15m(N-1)$	$15m(N-1)$	$15m(N-1)$	$15m(N-1)$
	$\boldsymbol{\beta}_{k,i}^*$	—	$6mN$	—	$6mN$
	$\tau_{k,i}^*$	—	mN	—	mN
	${}^{i-1}\mathbf{N}_{k,i}$	$m(70N-10)$	$m(71N-10)$	$70mN$	$71mN$
	${}^{i-1}\mathbf{f}_{k,i}^*$	$m(50N-4)$	$m(43N-4)$	$50mN$	$43mN$
	${}^r\mathbf{N}_{k,1}$	$42m$	$43m$	$42m$	$43m$
	${}^r\mathbf{f}_{k,1}^*$	$28m$	$24m$	$28m$	$24m$
III.	$\mathbf{a}_r'$	86	$27m+62$	86	$27m+71$
	$\mathbf{a}_r$	—	3	—	3
	$\mathbf{a}_{k,0}'$	$10m$	$9m$	$10m$	$9m$
	$\bar{\mathbf{a}}_{k,i}$	$20mN$	$17mN$	$20mN$	$17mN$
	$\ddot{q}_{k,i}$	$6mN$	$6mN$	$6mN$	$6mN$
	$\mathbf{a}_{k,i}'$	—	$m(N-1)$	—	$m(N-1)$
Total:		$224mN+$ $59m+99$	$205mN+$ $63m+68$	$377mN+$ $83m+256$	$333mN+$ $77m+202$
$m, N = 6, 3$		4485	4136	7540	6658
$m, N = 1, 6$		1502	1359	2601	2277

In the third section of the table, the computational requirements for the Forward Accelerations recursion are listed. To construct the 6×6 system of equations to compute the biased reference member acceleration requires [0M, $(27m - 3)$A] for the land-based algorithm and an additional [0M, 9A] for the underwater algorithm due to additional nonzero elements in $\mathbf{I}_r^H$ and $\boldsymbol{\beta}_r^H$. The resulting matrix is positive definite, so its solution can be completed using Cholesky decomposition and back substitution which requires [86M, 65A] as was determined in Part I for a 6×6 system of equations. The true accelcration of the reference member, is obtained by adding gravitational acceleration to this result ([0M, 3A]).

Unlike the fixed-base algorithm in Appendix B, the biased reference member acceleration is a general six-vector, and additional reductions in computation during the first few iterations of the forward recursions along the chains cannot be employed. Otherwise the amount of computation in the recursion is similar to the fixed base algorithm, and is the same for both land-based and underwater versions. Note that $\mathbf{a}'_{k,N}$ is not needed.

The resulting AB algorithm for land-based systems requires [$(224mN + 59m + 99)$M, $(205mN + 63m + 68)$A] for an m-chain system with N revolute joints each. For the OSU Hexapod or the Aquarobot on land ($m = 6$, $N = 3$), this corresponds to [4485M, 4136A] which is slightly more than the CRB/DTS method that was developed in Chapter 5. For a mobile base system with a 6 DOF arm ($m = 1$, $N = 6$), this corresponds to [1502M, 1359A] which is much less than the CRB/DTS approach.

When hydrodynamics is introduced, the hydrodynamic inertia matrices destroy the special structure of the rigid body spatial inertias. Since the CRB/DTS algorithms depend heavily on this structure to achieve efficient congruence transformations (see Appendix A), a hydrodynamic version of this algorithm is much less efficient than the AB/DTS algorithm developed here. It has little effect on the AB/DTS algorithm because the AB inertia matrices, to which they are added, are already full symmetric matrices. The computational requirements for the underwater AB/DTS algorithm are $[(377mN + 83m + 256)\mathrm{M}, (333mN + 77m + 202)\mathrm{A}]$. For the Aquarobot this corresponds to [7540M, 6658A], and for *Tiburon* it is [2601M, 2277A]. Compared to the land-based algorithm, hydrodynamics increases the computation between 65% and 70% depending on the topology.

7.5 Efficient ROV/Manipulator System Simulation

Thus far, an AB simulation algorithm has been developed for mobile base systems with multiple chains. The DTS kinematic modeling scheme was employed to cope with general connections of multiple chains to the base in an efficient manner. Another prevalent topology in URV systems contains a single serial chain of rigid bodies, which arises with an underwater vehicle with a single manipulator. One example is *Tiburon* with the Schilling manipulator attached. In this case, the simulation algorithm can be modified specifically for this topology in order to reduce the amount of computation. In this section, more efficient algorithms for such systems are described. In addition, a simplified dynamics algorithm, called decoupled dynamics, to further reduce the amount of computation is examined for these systems.

7.5.1 Full Dynamics: Efficient Kinematic Modeling

With a single chain, the coordinate system attached to the reference member can be placed in any convenient location. Since the goal is to minimize computation, the best location minimizes transformations between the first link of the chain and the reference member. The obvious choice is to place the origin and z-axis of the reference member's coordinate system coincident with the first link's coordinate system. To simplify the notation, the reference member is called link 0 in the serial chain and reference member quantities will be indicated with a trailing subscript of 0. The resulting algorithm is listed in Tables 17 and 18. The computations have been rearranged from the multiple chain algorithm of the previous sections. Besides the additional hydrodynamics computations, it is very similar to the single chain, fixed-base algorithm that was presented in Appendix B. The computational requirements for this algorithm are listed in Table 19 at the end of this section under the column labeled "Full Dynamics."

In the Forward Kinematics step, the orientation of link 0, the reference member, is still specified with three Euler angles so the computation of $\boldsymbol{v}_0^r$, ${}^0\boldsymbol{a}_g$, and ${}^0\boldsymbol{a}_{f-g}$ remains the same. Because the reference member's coordinate system is no longer at its center of gravity, the computation requirements for $\boldsymbol{\beta}_0^H$ is the same as the other links, [160M, 125A], which is more than in the previous section. However, the computational efficiency gained with this approach becomes apparent with the forward recursion along the chain. Multiple transformations no longer exist between the coordinate systems of the reference member and first link; therefore, computa-

Table 17: Articulated Body algorithm for a URV with a single manipulator.

I. Forward Kinematics Given: ${}^0\boldsymbol{R}_e$, $\boldsymbol{\omega}_0$, $\boldsymbol{v}_0$, ${}^e\boldsymbol{v}_f$, ${}^e\boldsymbol{a}_f$.

$$
\begin{aligned}
\boldsymbol{v}_0^r &= \boldsymbol{v}_0 - {}^0\boldsymbol{R}_e\,{}^e\boldsymbol{v}_f \\
{}^0\boldsymbol{a}_g &= g\,{}^0\boldsymbol{R}_e\hat{\mathbf{z}}_e \qquad \text{where } \hat{\mathbf{z}}_e = [0\ 0\ 1]^T \\
{}^0\boldsymbol{a}_{f-g} &= {}^0\boldsymbol{R}_e\,{}^e\boldsymbol{a}_f - {}^0\boldsymbol{a}_g \\
\boldsymbol{\beta}_0^H &= -\begin{bmatrix} \boldsymbol{\omega}_0 \times \bar{\boldsymbol{I}}_0\boldsymbol{\omega}_0 \\ \boldsymbol{\omega}_0 \times (\boldsymbol{\omega}_0 \times \boldsymbol{h}_0) \end{bmatrix} + \mathbf{f}_0^D(\boldsymbol{\omega}_0, \boldsymbol{v}_0^r) + m_0^f \begin{bmatrix} \boldsymbol{b}_0 \times {}^0\boldsymbol{a}_{f-g} \\ {}^0\boldsymbol{a}_{f-g} \end{bmatrix} + \\
&\qquad \mathbf{I}_0^A \begin{bmatrix} \mathbf{0} \\ {}^0\boldsymbol{a}_{f-g} + \boldsymbol{\omega}_0 \times \boldsymbol{v}_0^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_0 & \tilde{\boldsymbol{v}}_0^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_0 \end{bmatrix} \mathbf{I}_0^A \begin{bmatrix} \boldsymbol{\omega}_0 \\ \boldsymbol{v}_0^r \end{bmatrix}.
\end{aligned}
$$

for $i = 1, \ldots, N$. Given: ${}^i\boldsymbol{R}_{i-1}$, ${}^{i-1}\boldsymbol{p}_i$, $\dot{q}_i$.

$$
\begin{aligned}
\begin{bmatrix} \boldsymbol{\omega}_i \\ \boldsymbol{v}_i^r \end{bmatrix} &= {}^i\mathbf{X}_{i-1} \begin{bmatrix} \boldsymbol{\omega}_{i-1} \\ \boldsymbol{v}_{i-1}^r \end{bmatrix} + \boldsymbol{\phi}_i\dot{q}_i \\
{}^i\boldsymbol{a}_{f-g} &= {}^i\boldsymbol{R}_{i-1}\,{}^{i-1}\boldsymbol{a}_{f-g} \\
\boldsymbol{\zeta}_i &= \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{R}_{k,i-1}\left[\boldsymbol{\omega}_{k,i-1} \times (\boldsymbol{\omega}_{k,i-1} \times {}^{i-1}\boldsymbol{p}_i)\right] \end{bmatrix} + \begin{bmatrix} \sigma_i(\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}_i) \\ \bar{\sigma}_i(2\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}_i) \end{bmatrix} \\
\boldsymbol{\beta}_i^H &= -\begin{bmatrix} \boldsymbol{\omega}_i \times \bar{\boldsymbol{I}}_i\boldsymbol{\omega}_i \\ \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{h}_i) \end{bmatrix} + \mathbf{f}_i^D(\boldsymbol{\omega}_i, \boldsymbol{v}_i^r) + m_i^f \begin{bmatrix} \boldsymbol{b}_i \times {}^i\boldsymbol{a}_{f-g} \\ {}^i\boldsymbol{a}_{f-g} \end{bmatrix} + \\
&\qquad \mathbf{I}_i^A \begin{bmatrix} \mathbf{0} \\ {}^i\boldsymbol{a}_{f-g} + \boldsymbol{\omega}_i \times \boldsymbol{v}_i^r \end{bmatrix} - \begin{bmatrix} \tilde{\boldsymbol{\omega}}_i & \tilde{\boldsymbol{v}}_i^r \\ \mathbf{0} & \tilde{\boldsymbol{\omega}}_i \end{bmatrix} \mathbf{I}_i^A \begin{bmatrix} \boldsymbol{\omega}_i \\ \boldsymbol{v}_i^r \end{bmatrix}.
\end{aligned}
$$

end for i.

Table 18: Articulated Body algorithm for a URV with a single manipulator (continued).

II. Backward Dynamics Given: $\mathbf{I}_N^* = \mathbf{I}_N^H$, $\boldsymbol{\beta}_N^* = \boldsymbol{\beta}_N^H - {}^{N+1}\mathbf{X}_N^T\, \mathbf{f}_{N+1}$.
for $i = N, \ldots, 1$. Given: τ_i.

$$
\begin{aligned}
\mathbf{n}_i &= \mathbf{I}_i^* \boldsymbol{\phi}_i \\
m_i^* &= \boldsymbol{\phi}_i^T \mathbf{I}_i^* \boldsymbol{\phi}_i \\
\mathbf{N}_i &= \mathbf{I}_i^* - \mathbf{n}_i (m_i^*)^{-1} \mathbf{n}_i^T \\
\tau_i^* &= \boldsymbol{\phi}_i^T \boldsymbol{\beta}_i^* + \tau_i \\
\mathbf{I}_{i-1}^* &= \mathbf{I}_{i-1}^H + {}^i\mathbf{X}_{i-1}^T \mathbf{N}_i {}^i\mathbf{X}_{i-1} \\
\boldsymbol{\beta}_{i-1}^* &= \boldsymbol{\beta}_{i-1}^H - {}^i\mathbf{X}_{i-1}^T \left[\mathbf{N}_i \boldsymbol{\zeta}_i + \mathbf{n}_i (m_i^*)^{-1} \tau_i^* - \boldsymbol{\beta}_i^* \right]
\end{aligned}
$$

end for i.

III. Forward Accelerations Given: $\mathbf{f}_0$ (thruster force).

$$
\begin{aligned}
\mathbf{a}_0' &= (\mathbf{I}_0^*)^{-1} (\mathbf{f}_0 + \boldsymbol{\beta}_0^*) \\
\mathbf{a}_0 &= \mathbf{a}_0' + \begin{bmatrix} \mathbf{0} \\ {}^0\boldsymbol{a}_g \end{bmatrix}
\end{aligned}
$$

for $i = 1, \ldots, N$. Given: ${}^i\mathbf{X}_{k,i-1}$.

$$
\begin{aligned}
\bar{\mathbf{a}}_i &= {}^i\mathbf{X}_{i-1} \mathbf{a}_{i-1}' + \boldsymbol{\zeta}_i \\
\ddot{q}_i &= (m_i^*)^{-1} \tau_i^* - \left[\mathbf{n}_i (m_i^*)^{-1} \right]^T \bar{\mathbf{a}}_i \\
\mathbf{a}_i' &= \bar{\mathbf{a}}_i + \boldsymbol{\phi}_i \ddot{q}_i
\end{aligned}
$$

end for i.

tions for intermediate quantities between these, such as $\mathbf{v}^r_{k,0}$, ${}^0\boldsymbol{a}_{k,f-g}$, and $\boldsymbol{\zeta}_{k,0}$, are no longer needed.

Additional efficiencies are gained in the transformation of quantities between coordinate systems 0 and 1. Because of coordinate system 0's placement, the transformation only consists of a rotation about the z-axis when the first joint is revolute, or a translation along the z-axis when it is prismatic. The former case is considered in this discussion so that comparisons can be made with previous results using the same type of joint. Results in Appendix A shows that with a z-planar rotation, the transformation of a Cartesian vector requires [4M, 2A], and the transformation of a spatial vector becomes equivalent to planar rotations of two Cartesian vectors ([8M, 4A]). Because ${}^0\boldsymbol{p}_1$ is also zero, the computation of $\boldsymbol{\zeta}_1$ requires only [2M, 0A] and has only two nonzero elements, and the congruence transformation of the reflected AB inertia, $\mathbf{N}_1$, can be shown to require only [22M, 25A] and the result still contains a zero row and column. These conditions affect the computational requirements for $\mathbf{v}^r_1$, ${}^1\boldsymbol{a}_{f-g}$, $\boldsymbol{\zeta}_1$, $\mathbf{I}^*_0$, $\boldsymbol{\beta}^*_0$, and $\bar{\mathbf{a}}_1$ as shown in the table.

The resulting algorithm requires $[(377N + 132)\text{M}, (333N + 95)\text{A}]$. For *Tiburon* with a Schilling manipulator, $N = 6$ and it requires [2394M, 2093A]. The general algorithm requires [2601M, 2277A] for the same system. Therefore, the algorithm developed in this section specifically for a single chain system results in 8% less computation.

7.5.2 Decoupled Dynamics: Simplified Dynamics Modeling

A second method of reducing the requirements of the dynamics computation involves simplification of the dynamic equations themselves. The approach used thus far in the chapter assumes that the acceleration of the reference member is significantly affected by the forces exerted by the serial chain attached to it. In this case, this acceleration is unknown and coupled to the dynamics computation for the manipulator. This is especially true for the legged vehicles in which nearly all of the driving force on the reference member comes from the legs that are in contact with the ground. The manipulator of the ROV system considered in this section does not generally come in contact with the ground. In addition, the ROV is assumed to be massive compared to the manipulator, or it is operating within a tight servo loop to maintain its desired motion. In either or both of these cases, the motion of the manipulator has a negligible effect on the motion of the ROV, and its dynamics can be decoupled from the manipulator. Therefore, its acceleration, $\mathbf{a}_0$, is assumed known when simulating the manipulator. This approach has two advantages: its simplifies the conceptual framework especially for the development of control systems, and lowers the computational requirements of the simulation.

In addition to the omission of the linear system solution to obtain $\mathbf{a}'_0$, the most significant reductions in the amount of computation come from the fact that $\boldsymbol{\beta}_0^H$, $\mathbf{I}_0^*$, and $\boldsymbol{\beta}_0^*$ are no longer needed. As a direct result, $\mathbf{N}_1$ also does not need to be computed. Finally, only the third elements (in the case of a revolute joint) of $\boldsymbol{\beta}_1^*$ and

Table 19: Computational requirements of the single chain URV simulation algorithm.

	Full Dynamics		Decoupled Dynamics	
	×	+	×	+
$\boldsymbol{v}_0^r$	12	9	12	9
${}^0\boldsymbol{a}_g$	4	—	4	—
${}^0\boldsymbol{a}_{f-g}$	12	9	12	9
$\boldsymbol{\beta}_0^H$	160	125	—	—
$\mathbf{v}_i^r$	$20N-12$	$13N-8$	$20N-12$	$13N-8$
${}^i\boldsymbol{a}_{f-g}$	$8N-4$	$4N-2$	$8N-4$	$4N-2$
$\boldsymbol{\zeta}_i$	$22N-20$	$10N-10$	$22N-20$	$10N-10$
$\boldsymbol{\beta}_i^H$	$160N$	$125N$	$160N-79$	$125N-70$
$\mathbf{n}_i$	—	—	—	—
$(m_i^*)^{-1}$	$N-1$	—	$N-1$	—
$\mathbf{n}_i(m_i^*)^{-1}$	$5N-5$	—	$5N-5$	—
$\mathbf{N}_i$	$15N-15$	$15N-15$	$15N-30$	$15N-30$
τ_i^*	—	N	—	N
$\mathbf{I}_{i-1}^*$	$70N-48$	$92N-52$	$70N-70$	$92N-92$
$\boldsymbol{\beta}_{i-1}^*$	$50N-25$	$49N-23$	$50N-60$	$49N-59$
$\mathbf{a}_0'$	86	71	—	—
$\mathbf{a}_0$	—	3	—	—
$\bar{\mathbf{a}}_i$	$20N-12$	$17N-11$	$20N-12$	$17N-11$
$\ddot{q}_i$	$6N$	$6N$	$6N$	$6N$
$\mathbf{a}_i'$	—	$N-1$	—	$N-1$
Total:	$377N+132$	$333N+95$	$377N-265$	$333N-265$
$(N=6)$	(2394)	(2093)	(1997)	(1733)

hence β_1^H are needed which requires [40M, 39A] and [81M, 55A], respectively. The computational requirements of the resulting algorithm are shown in Table 19 under the column labeled "Decoupled Dynamics." The total computational requirements at the bottom of this table show that the amount of computation using the decoupled dynamics approach is reduced by [397M, 360A] from the Full Dynamics approach discussed in the first part of this section. For the *Tiburon*/Schilling system this results in [1997M, 1733A] which is over 15% less than the Full Dynamics approach, and nearly 25% less than the general multiple chain algorithm. Before this method is used, additional experimentation to measure the errors that this simplification incurs should be carried out, to determine the efficacy of this approach.

7.6 Summary and Conclusions

In this chapter, an efficient dynamics algorithm has been developed for simulation of a URV system with multiple serial chains. The resulting general algorithm is capable of simulating both multilegged underwater vehicles, like the Aquarobot hexapod, and ROV/manipulator systems, like the Monterey Bay Aquarium Research Institute's ROV, *Tiburon*, with a Shilling manipulator. The AB approach to computing the dynamics for these systems has been shown to be most efficient, so it was used as the basis for this development. Using the efficient transformations from Appendix A, a more efficient algorithm for land-based robotic systems was developed first which required $[(224mN + 59m + 99)\text{M}, (205mN + 63m + 68)\text{A}]$ for an m-chain system with N revolute joints each.

A significant goal of this chapter was to efficiently incorporate the hydrodynamic

effects presented in the previous chapter into this algorithm. After careful derivation, the computational requirement for the resulting underwater AB/DTS algorithm is $[(377mN + 83m + 256)\text{M}, (333mN + 77m + 202)\text{A}]$. For the Aquarobot this corresponds to [7540M, 6658A], and for *Tiburon* it is [2601M, 2277A]. Compared to the land-based algorithm, hydrodynamics increases the computation between 65% and 70% depending on the topology.

An advantage of this algorithm is that it is general enough to compute the dynamics of arbitrary mobile base, multiple chain systems. This is a disadvantage, however, when the system consists of a single serial chain such as an ROV with a single manipulator. With more efficient kinematic modeling, an algorithm specifically for single chain systems was also developed. The resulting algorithm requires $[(377N + 132)\text{M}, (333N + 95)\text{A}]$. For the *Tiburon* ROV with a Schilling manipulator ($N = 6$), this translates to [2394M, 2093A] which is 8% less computation. Often, another characteristic of these systems is that the ROV is massive compared to the manipulator, or under very tight servo control. Under either or both of these conditions, a simplified dynamics algorithm has also been developed which decouples the chain dynamics from the mobile base. The resulting algorithm requires $[(377N - 265)\text{M}, (333N - 265)\text{A}]$. For the *Tiburon*/manipulator system this corresponds to [1997M, 1733A] which as nearly 25% less than the general multiple chain algorithm.

Although hydrodynamics introduces a significant amount of computation of the dynamics for these systems, it is nearly low enough to enable real-time simulation on

reasonably priced workstations. The goal of the next chapter is to use object oriented design and programming to obtain an efficient implementation of the general mobile base, multiple chain algorithm. It is implemented on an SGI workstation with a MIPS R4400 processor, and the computation time for the dynamics is determined in order to evaluate this statement.

CHAPTER VIII

DynaMechs: An Object Oriented Robot Simulation Package

8.1 Introduction

In the previous chapter, an efficient dynamic and hydrodynamic simulation algorithm was developed for general multiple chain robotic systems. In this chapter, the implementation of this algorithm is presented. Many simulation algorithms have been implemented using third generation languages like C and Pascal. The advantage of these languages over second generation languages is their support of structured programming techniques which emphasizes procedural decomposition of the programming task. This reduces the complexity of the resulting task because it allows for the independent development of sections (modules) of a large program. As the size of desired software continues to grow however this method breaks down because maintenance costs become very high when too many internal details and interactions between the modules must be tracked. Even feature-rich applications (like a simulator for a general class of robotic mechanisms) that are moderately large can also suffer from unnecessarily large maintenance costs.

In response to the complexity management problem, a concept called *information hiding* or *encapsulation* was invented which has spurred the development of a

fourth generation of programming languages. These include object oriented programming languages (OOPLs) which include C++, Smalltalk, and Common Lisp Object System (CLOS), and object-based languages, like Ada [61]. Using OOPLs, the problem domain is decomposed in a top-down fashion into a hierarchy of component objects, also called a *part-of* decomposition. The structure of these objects are defined by class types that contain hidden data and a public interface in the form of methods (functions), whose implementation is also hidden, which act on this data. As a result, each object has a certain functionality, and now the emphasis in the top-down design focuses on the interaction of objects using only the method interfaces specified to achieve the programming goal. This significantly reduces the complexity of this task. It also removes the detailed implementation of the objects' internal data and methods from the top level design decisions, and decouples their implementation from the rest of the programming task.

The greatest obstacle to overcome in realizing the benefits of object oriented programming is the need to change ones thinking about programs in terms of data and subroutines to the OO entities called objects [62]. With the proper object decomposition, a second bottom-up design process concentrates on the hidden implementation for each class of objects. This is accomplished with a *kind-of* classification of these objects which leads naturally to a second hierarchy called the class hierarchy. With their interface defined, the implementation of these classes are not dependent on (decoupled from) one another, the amount of information needed by the programmer at any one time to solve programming subtasks is reduced, and the complexity

is significantly reduced. This method of programming also provides strong support for *code reuse*. As a result productivity is increased and maintenance of the code is much simpler [62].

For further discussion of OOD philosophy or methodologies, the reader is referred to the numerous texts and articles written on this topic (e.g., [61]). With this philosophy in mind, however, the purpose of this chapter is to present the resulting OO implementation of a dynamic and hydrodynamic simulation software package. The C++ language has been chosen because of its current popularity, availability of compilers, and its close ties with the third generation programming language, C. It is named **DynaMechs***, and is applicable to general multiple chain systems. The next two sections present the object and class hierarchies that have been developed to support simulation of a general class of robotic systems. At the same time a number of the OOD concepts that were used in this development are described. Then in Section 8.4, use of the resulting software package is presented. It describes initialization of the package, the implementation of the Articulated Body (AB) algorithm, and numerical integration. Also discussed is the implementation on a Silicon Graphics (SGI) workstation including runtime performance (since efficient implementation is important), and examples of graphical interfaces.

8.2 Object Hierarchy

Simply stated, an object is an entity that has a state (variables), behavior (methods or procedures) and identity (its name). The first step in the OOD process is to

*Pronounced dynamics, and stands for dynamics of mechanisms.

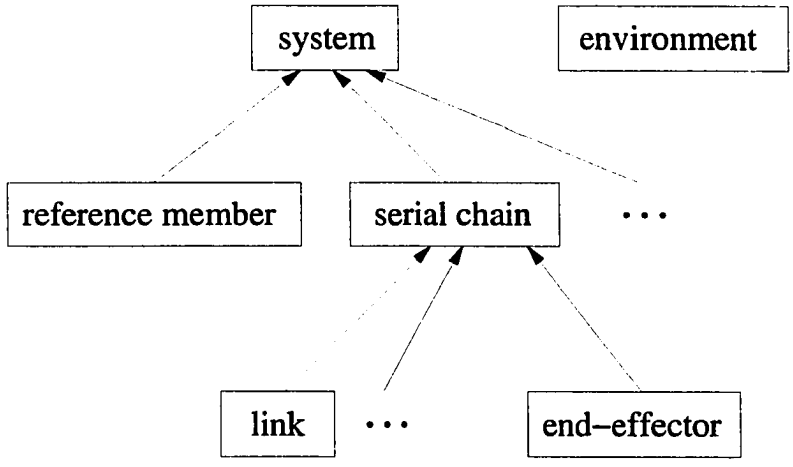


Figure 26: Object hierarchy for **DynaMechs**.

decompose the programming problem into such objects. These objects may or may not be decomposed further into more detailed objects. This procedure is repeated until the level of detail desired is reached. The resulting objects are organized into a hierarchy which represents the *part-of* decomposition. Figure 26 illustrates the object hierarchy that has been developed for the simulation of a general class of robotic systems.

A robot-centric view is taken in the first step of the decomposition where the domain is divided into two parts, the robotic system and the surrounding environment, as indicated by the two top-level objects. The former contains all procedures and data related to the simulation of the robotic system and the latter contains the information and methods related to the external environment with which the system interacts such as gravity, fluid characteristics, and terrain models (not shown). The emphasis in this chapter is on the system object which must be capable of repre-

senting robots with varying topologies from single chain, fixed base manipulators to multiple chain, mobile base legged vehicles. Therefore, the system is a composite object and is decomposed further. All of these systems have a base (fixed or mobile) called the reference member and a number (an array in this implementation) of serial chains attached to it. The serial chain objects are further decomposed into a series (array) of link objects and an end effector object.

In terms of functionality, the system object provides the user with the interface to all of the needed methods such as initialization, computation of dynamics, integration, and rendering. The reference member and serial chain objects provide the system with the necessary methods to carry out their part of these tasks. In turn, the link and end effector objects provide the functions required by the serial chain objects. For the dynamics computation, the system controls the communication of data between the reference member and the serial chain objects. The reference member object contains the data and functions necessary to compute the quantities associated with the reference member. The serial chain sets up the three recursions along the chain in the AB algorithm and passes data between the link and end effector objects. The link objects are responsible for computing the quantities associated with themselves (the equations in one iteration of the innermost loop of the dynamics algorithm), and finally, the end-effector object must detect contacts of the last link in the chain with the environment and compute contact forces, if present.

8.3 Class Hierarchy

In the previous discussion, the type of reference member (fixed or mobile) and link with an associated joint (revolute, prismatic, etc.) have not been specified. This is accomplished with the development of the class hierarchy which is the second step in the OOD process. Booch [61] states that "a class is a set of objects that share a common structure and a common behavior." Another way of describing the class-object relationship is that classes define the "types" of the objects that appear at the nodes of the object hierarchy. In this section, the class hierarchy developed for **DynaMechs** is presented. In addition, important OOD concepts that were used in this development including encapsulation, inheritance, and polymorphism are presented. To illustrate these concepts, examples are taken from the design of **DynaMechs.**

8.3.1 The C++ Class

As Booch's definition suggests, a class has two distinct features. It defines the *member variables* that store data or state, and *member functions* which correspond to a set of behaviors that operate on the member variables. As an example, Figure 27 shows the definition of a C++ class, `RigidBody`, that is used in **DynaMechs.** Examples of member variables for a rigid body include all of its mass and inertia properties, and hydrodynamic coefficients like the drag coefficients and the added mass matrix. Also note that a number of new types have been defined (using `typedef`) to make the declarations of dynamics variables

```
class RigidBody
{
protected:
    float          mass;
    CartesianVector cgPos;        // position of center of gravity.
    CartesianVector h;            // first mass moment (mass*cgPos).
    CartesianTensor Ibar;         // 3x3 inertia tensor about body axes.
    SpatialTensor   SpatialInertia;

    int            axis;          // drag parameters.
    float          Cd, length, radius;
    float          dispFluidVol;  // volume of fluid displaced by body.
    float          dispFluidMass; // mass of fluid displaced by body
    CartesianVector cbPos;        // position of center of buoyancy.
    SpatialTensor   AddedMass;    // 6x6 added mass matrix
    SpatialTensor   HydrodynamicInertia;

public:
    RigidBody(FILE *cfg_ptr);
    ~RigidBody();
    void Compute_Drag(SpatialVector vrel, SpatialVector fd);
};
```

Figure 27: `RigidBody` class.

clearer. For instance a `CartesianVector` is a one-dimensional, three element array of `floats`, `CartesianTensors` are 3×3 arrays, `SpatialVectors` are six-vectors, and `SpatialTensors` are 6×6 arrays.

As illustrated in this figure, C++ also allows the designer to control access to the member variables and functions with the use of the `private`, `protected`, and `public` keywords. The `private` and `protected` sections contain the variables and functions that cannot be accessed/invoked from outside of the class. The difference between them is made clear in the discussion on inheritance below. The `public`

section contains the user accessible variables and functions. Normally all variables and some functions are private or protected, and it is through calls to the public functions that the variables are manipulated.

These mechanisms reinforce the concept of *information hiding* or *encapsulation* which is the key to the benefits of complexity management with OOD. As a result, the emphasis is moved from the implementation of the variables or functions, which can be deferred until later, to the interface that is provided by the class. Therefore, the user only has to worry about the interface in order to use the class, and the implementation within the class is the class designer's responsibility. This emphasizes the need for early and complete interface specifications in the design process. In fact, as long as the interface specifications do not change the underlying implementation can undergo arbitrary modification without affecting other portions of the software system that use this class.

In the `RigidBody` example, three functions are provided in the public section. The first two are special functions required by C++ and are called the constructor (which is given the same name as the class) and the destructor which is preceded by a tilde. The constructor is called when the class is instantiated for an object and performs the appropriate functions to initialize the member variables. In this example, the constructor expects a pointer to a file that contains the data with which to initialize the variables. For example, an object called `anvil` would be instantiated as follows:

```
RigidBody anvil(cfg_ptr);
```

where `cfg_ptr` points to a file that has been opened and contains the object's parameters. The destructor is automatically called when the object goes out of scope or the program exits and it performs specific shutdown procedures that may be required by the class.

The third function in this example, `Compute_Drag`, is a normal member function that is provided by the designer for the users of the class. For this function, the user supplies the object's relative velocity, and the function accesses the class's private variables to perform the computation of the drag force. This function is accessed in a manner similar to accessing an element of a structure as follows:

```
anvil.Compute_Drag(vrel, fD);
```

Should the implementation of the drag force computation change at a later date, users would not be required to change their portion of the code provided the interface (the parameters and name of the function) does not change.

8.3.2 Inheritance

Another goal in the design of the classes is to increase code reuse through a mechanism called *inheritance*. The desire in this case is to move variables and methods that are shared by more than one class into another, more general, class which is then inherited by the classes requiring them. The new class is called a *superclass* and its structure and behavior are inherited by its *subclasses*. This terminology indicates a hierarchical relationship between classes and is the motivation behind the development of the class hierarchy.

```
class Link: public RigidBody
{
protected:
    RotationMatrix  _R;       // rotation matrix from inboard link
    CartesianVector _p;       // position wrt inboard link.

public:
    Link(FILE *cfg_ptr);

    // Link-to-link spatial transformation functions:
    void   tx_To_Inboard(SpatialVector curr, SpatialVector prev);
    void tx_From_Inboard(SpatialVector prev, SpatialVector curr);
};
```

Figure 28: Link class.

Since links and mobile bases are both examples of rigid bodies in robotic systems, the `RigidBody` class is an example of a superclass that must be inherited by both `Link` and `MobileBase` subclasses. An example inheritance in the definition of the `Link` class is shown in Figure 28. The first line, contains the command to inherit the `RigidBody` class definition. The `public` keyword in this line indicates how access to inherited variables and methods are determined. With public inheritance, public attributes in the superclass will remain public in the subclass, protected attributes in the superclass become similar to private attributes of the subclass, and private variables in the superclass will be inaccessible to the subclass.

In this implementation, access to superclass variables is desired within the subclasses; therefore, the `protected` keyword is used most often. When a `Link` object is instantiated, the constructor for the `RigidBody` class is called as well as the one for

the `Link` class. The resulting object contains all of the variables and methods from both classes. In other words, it will have the variables associated with a rigid body (mass, inertia, etc.) as well as a link in a serial chain (position and orientation with respect to its inboard neighbor). Methods from both classes will also be available so that the drag force on the link can be computed as well as the transformation of spatial vectors to and from the coordinate system of the inboard neighbor.

Inheritance from more than one superclass is also possible. This is desired with the `MobileBase` class. Like a link, a mobile base has the characteristics of a rigid body so it should inherit the definition of the `RigidBody` class, but it also has the added characteristics of a reference member. Because fixed and mobile base objects share common attributes a `RefMember` superclass is defined to hold these which is then inherited by the corresponding classes. An example of a `RefMember` superclass is shown in the top part of Figure 29. In contains the position and orientation relative to an inertial coordinate system (ICS), and it also provides the user with a pair of functions to transform vectors between the ICS and the reference member's coordinate system.

In contrast to a fixed base, a mobile base is also capable of motion with respect to the ICS. Therefore, the corresponding `MobileBase` class includes variables for its spatial velocity and acceleration with respect to the ICS as shown in the bottom part of Figure 29. In addition, it could provide a member function, like `Update_State`, which is responsible for numerically integrating the velocity and accelerations to obtain new values for the position, orientation, and velocity of the reference member.

```
class RefMember
{
protected:
    CartesianVector pos;   // position wrt inertial coord. sys. (ICS)
    EulerAngles     pose;  // [phi theta psi]' Euler angles

public:
    RefMember(FILE *cfg_ptr);

    void Tx_From_ICS(CartesianVector pICS, CartesianVector pRef);
    void   Tx_To_ICS(CartesianVector pRef, CartesianVector pICS);
};


class MobileBase: public RigidBody, public RefMember
{
private:
    SpatialVector vel;      // velocity and accel wrt ICS.
    SpatialVector acc;

public:
    MobileBase(FILE *cfg_ptr);

    void Update_State();
};
```

Figure 29: `RefMember` and `MobileBase` classes.

Note that the first line of the `MobileBase` class definition specifies more than one class to be inherited, which is called, quite appropriately, *multiple inheritance.* Therefore, objects that are instances of the `MobileBase` class will also contain the member variables and functions of both the `RigidBody` and `RefMember` classes. Multiple inheritance can also be achieved by inheriting classes that are themselves subclasses. Examples of this are given with the further refinement of the `Link` class in

the next subsection.

These examples show how inheritance can be used to achieve some level of code reuse. The key to successful decomposition is to move variables and functions that are common to more than one class to a superclass which is then inherited by these subclasses. The result is that attributes are moved to the most general level possible. The advantage is that the amount of code reuse by derived classes is maximized because these variables and functions need only to be implemented once. This reduces the amount of coding and testing that is required of the programmer and therefore efficiency in code development is increased.

8.3.3 Virtual Functions and Abstract Classes

Polymorphism is another important concept to be considered during the development of the class hierarchy. This is defined as the ability to refer to methods of different classes by a common name, and through this name objects of various classes can "respond to a common set of operations in different ways" [61]. One obvious example in the simulation of robotic mechanisms where this is desirable is in the implementation of dynamics computation at the link level. Different joint types (e.g., revolute and prismatic) require different efficient implementations of the equations. With a third generation language this would require conditional or case statements at the serial chain level to execute the proper code depending on the type of joint involved. This code would also have to be rewritten each time a new joint type is to be modeled. However, with polymorphism this becomes unnecessary because the dependence on the type of joint is eliminated from the serial chain code.

In this section, further refinement of the `Link` class, to pair a link with either a revolute or a prismatic joint, is used to illustrate how this is accomplished. A function to perform the efficient congruence transformation of the reflected Articulated Body (AB) inertia matrices, $\mathbf{N}_i$, is given as an example since it is shown, in Appendix A, to depend on the type of joint. The first step is to expand the class hierarchy related to the `Link` class which will serve as the superclass for all of the link subclasses which are *specialized* for different joint types. The resulting classes are illustrated in Figure 30. Note that an intermediate class, `MDHLink`, is also defined which contains Modified Denavit-Hartenberg (MDH) parameters and related quantities that are common to both revolute and prismatic joints. The reason that these terms are not included in the `Link` class is because multiple degree of freedom joints cannot be modeled using MDH parameters, but to support polymorphism they will also have to inherit the `Link` class (see `BallnSocketLink` in the next section).

To achieve polymorphism, a uniform interface must be provided in the `Link` superclass for each function that will be specialized according to the joint type in its subclasses. In C++, this is accomplished with the `virtual` keyword. This is used in both the `Link` and `MDHLink` classes for the `Congtx_To_Inboard_Refl()` function. The "`= 0`" at the end of these declarations indicates that these are *pure virtual functions*. This indicates that the definition of this function is deferred until one of the subclasses defines it. Because it is undefined, these classes cannot be used to directly instantiate an object (that is, no objects exist that are just `Link` or `MDHLink` objects). This type of class is called an *abstract* class and can only be used

```
class Link: public RigidBody
{
public:
    Link(FILE *cfg_ptr);

    virtual void CongTx_To_Inboard_Refl(SpatialTensor N,
                                        SpatialTensor Nib) = 0;
};

class MDHLink: public Link
{
protected:
    float aMDH, alphaMDH, dMDH, thetaMDH;    // Modified DH parameters

public:
    MDHLink(FILE *cfg_ptr);

    virtual void CongTx_To_Inboard_Refl(SpatialTensor N,
                                        SpatialTensor Nib) = 0;
};

class RevoluteLink: public MDHLink
{
public:
    RevoluteLink(FILE *cfg_ptr);

    void CongTx_To_Inboard_Refl(SpatialTensor N, SpatialTensor Nib);
};

class PrismaticLink: public MDHLink
{
public:
    PrismaticLink(FILE *cfg_ptr);

    void CongTx_To_Inboard_Refl(SpatialTensor N, SpatialTensor Nib);
};
```

Figure 30: Polymorphism in the link classes.

in inheritance in base classes. In contrast, classes that can be used to instantiate objects are called *concrete* classes such as the `RevoluteLink` and `PrismaticLink` classes.

With that in mind, Figure 31 lists some C++ code that could be used in the `SerialChain` class and illustrates how the virtual functions are used to achieve polymorphism. In this example, an array of two pointers to `Link` objects are used (line 1). Since these are pointers and not `Link` objects themselves nothing is actually instantiated so that the rule against instantiation of an abstract class is not violated. This provides the common name through which the function is called. Lines 4 and 5 are then used to instantiate a `RevoluteLink` and `PrismaticLink` object. The `new` command allocates memory for the specified class, calls the constructor with the specified parameter list, and returns the address of the object. Although the returned values are pointers to `RevoluteLink` and `PrismaticLink` classes and are assigned to a `Link` pointer, the compiler does not complain about a type mismatch because `Link` is a superclass of the others.

With the common name, called `Link`, and the uniform function interface, called `Congtx_To_Inboard_Refl`, polymorphism can now be used. This is illustrated in line 8 of the example where the congruence transformation of `N[2]` is performed across both links in a backward recursion. Regardless of the actual link type, the function call is the same. Internally, however, `PrismaticLink`'s function was executed for `link[1]` and `RevoluteLink`'s function for `link[0]`. This is also called *dynamic binding* where the particular function to be executed is not known until runtime

```
1    Link *link[2];
2    SpatialTensor N[3];
3
4    link[0] = new RevoluteLink(cfg_ptr);
5    link[1] = new PrismaticLink(cfg_ptr);

     ⋮

6    for (i=2; i>0; i--)
7    {
8        link[i-1]->Congtx_To_Inboard_Refl(N[i], N[i-1]);
9    }

     ⋮
```

Figure 31: An example using polymorphism.

(lines 4 and 5). With polymorphism no case statements or conditionals are needed in this code to determine which function to call. Other than the instantiation of the link objects themselves, this portion of the code does not require knowledge about the various types of links that may be defined. This is also advantageous if different link subclasses are added to the software system at a later date because this section of the code would require no modification.

8.3.4 The Hierarchy

The structure implied by the inheritance processes described in the previous three subsections leads to a hierarchy in which the superclasses are placed above the subclasses. In the resulting tree, the more "general" classes appear at the top where characteristics are common to more of the subclasses. As the tree is traversed downward, the classes define more specialized objects. This hierarchy is often referred

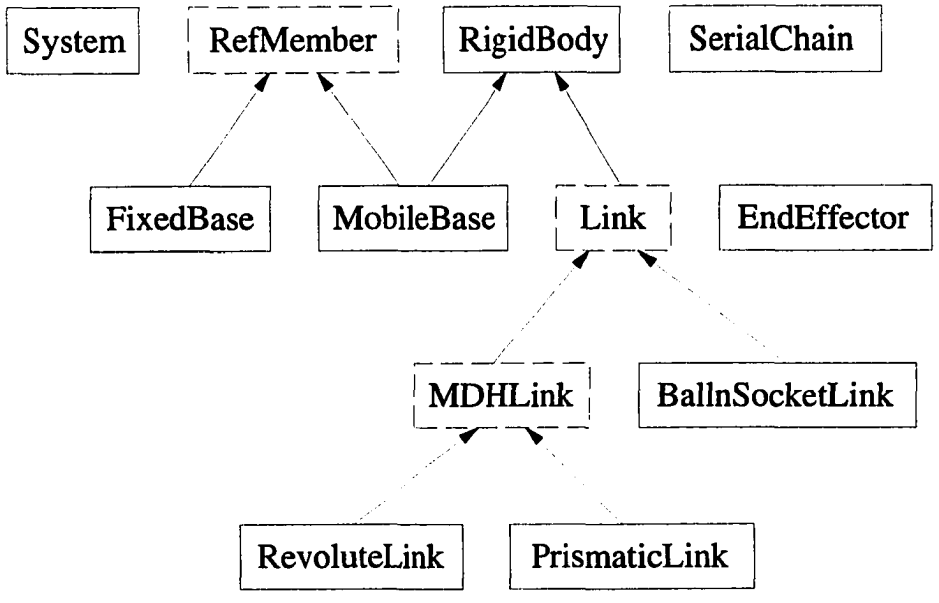


Figure 32: Class hierarchy for **DynaMechs**.

to as the *kind-of* hierarchy[†]. The hierarchy that has been developed for the **DynaMechs** software is shown in Figure 32. Many of the classes have been used in the examples in the previous discussions, and the abstract classes are indicated by dashed boxes.

Reexamining the object hierarchy in Figure 26 in light of this class hierarchy, a number of comments can now be made about the software implementation. The `System` class is a *simple* class (that is, no inheritance and no virtual functions). Its member variables include a pointer to a `RefMember` object and an array of pointers to `SerialChain` objects. The array of pointers to the `SerialChain` objects is dynamically allocated in the constructor when a `System` object is instantiated. This allows

[†]This comes from the statement, "a *child class* is a kind of *parent class*," which should hold true for each parent-child pair in the tree. For example, a `Link` is a kind of `RigidBody`

for any number of serial chains attached to the reference member to be simulated.

The `RefMember` class contains member variables for the position and orientation of the reference member with respect to the inertial coordinate system and provides functions for transformations between this and the reference member's coordinate system. It is an abstract class because it also contains a number of pure virtual functions (not shown in Figure 29) to provide `System` objects with a uniform interface for functions whose implementation depends on whether the reference member is a `FixedBase` or `MobileBase` object. This allows for dynamic binding of the proper code at runtime so that either fixed or mobile base systems can be simulated. One example would be the function to compute the acceleration of the reference member. For the `MobileBase` function the dynamics developed in the previous chapter would be implemented, while the `FixedBase` function would simply return zero acceleration without any computation.

The `SerialChain` class is another simple class. Its member variables include an array of pointers to `Link` objects and a pointer to an `EndEffector` object. The array is also dynamically allocated in the constructor when a `SerialChain` object is instantiated, so that a serial chain with an arbitrary number of links can be simulated by this software. The `EndEffector` class is also a simple class and currently contains code for modeling compliant contacts at the end of each serial chain and computing external tip forces as described in [7]. A pointer is provided in the `SerialChain` class to support future specialization under the `EndEffector` class to support the specific functions of grippers, foot pads, sensors, and even cameras.

The structure of the `Link` class and its subclasses were discussed in some detail above. The only addition to this part of the hierarchy is the `BallnSocketLink` class which is used to efficiently model the kinematics and dynamics of the footpads on the end of each leg of Aquarobot which contain unpowered ball-in-socket joints. This class is an example of how to include new types of links. Its kinematics cannot be modeled using MDH parameters, but rather a position vector and a set of Euler angles is most efficient. The joint axis vector, ϕ_i, is 6×3, hence the joint position, velocity, acceleration, and torque are now represented by 3-vectors (not scalars as in the case of `MDHLinks`). These differences are all hidden in the implementation of the `BallnSocketLink` class. The primary responsibility of this class definition is to provide the definitions of the virtual functions exactly as they appear in the `Link` class.

In summary, the programming problem was decomposed into smaller tasks with the development of the object hierarchy. The class hierarchy developed in this section can be considered orthogonal to the object decomposition, and its development further decomposes the programming tasks. Through techniques such as inheritance, encapsulation, and polymorphism, the overall complexity of the problem is significantly reduced because interactions between code segments are limited and tightly controlled. This also significantly reduces the amount of time taken to maintain and modify such code.

8.4 Using DynaMechs

Now that the object and class hierarchies have been presented, the use of this software package is discussed in this section. The steps for performing the dynamic simulation of a robotic system using this package are initialization of the composite `System` object, execution of the dynamics algorithm, numerical integration, and, if desired, graphical output. Each of these steps is discussed in this section. An example of the graphical output obtained on a Silicon Graphics (SGI) workstation is presented along with performance results on this platform.

8.4.1 Initialization

Before the dynamics algorithm can be implemented, an instance of the `System` class must be created. Just like a variable, it can be given any name, and for this discussion it is called `robot`. When instantiated, the constructor for the `System` class requires, as arguments, a pointer to the file containing all of the necessary kinematic, dynamic and hydrodynamic parameters for the entire robotic system, and a set of variables which will contain the state, accelerations, and inputs of the system. This is accomplished with the code shown in Figure 33.

The rest of the initialization process is hidden from the user. The `System` constructor extracts the information it needs from the file and then uses it to initialize its member variables. Two pieces of data specify the type of reference member (fixed or mobile) and the number of serial chains the system has. These are then used to direct the `System`'s initialization of its component objects, an instance of

```
// state variables.
     EulerAngles     refPose;
     CartesianVector refPos;
     SpatialVector   refVel;
     float ***jointPos, ***jointVel;

// accelerations.
     SpatialVector refAcc;
     float ***jointAcc;

// force/torque inputs
     SpatialVector refForce;
     float ***jointForce;

// a robotic system.
     System robot(cfg_ptr, refPose, refPos, refVel,
                  &jointPos, &jointVel, &jointAcc, &jointForce);
```

Figure 33: Initialization of robotic system and input/output variables.

the `RefMember` class and an array of `SerialChain` instances. This is accomplished by passing the file pointer when the constructor for these objects are called. During the initialization of the `SerialChain` objects the file pointer is also passed on to the individual `Link` constructors and each `EndEffector` constructor. In this manner, the data for the entire robotic system can be contained in a single input data file.

At the same time, the state variables that are passed to the `System` constructor are then passed along to the appropriate constructors of component objects which will set these to the initial conditions specified in the input file. Since the number of serial chains and the number of links in each chain are dynamically allocated at runtime, the corresponding state, acceleration, and input variables (`jointPos`,

`jointVel, jointAcc, jointForce`) are also dynamically allocated during this initialization. The resulting variables appear to the user as three dimensional arrays. For instance, the torque for the fourth joint of the second serial chain of the system would be set as follows:

```
jointForce[1][3][0] = torque;
```

where C++ arrays begin indexing with zero so the first link of the first chain would be referenced with two zero indices. The third index in the example supports multiple degree of freedom joints. For revolute and prismatic joints this is always zero, and for ball-in-socket joints it ranges from 0 to 2. Once they are allocated, the `System` object receives input from and sends output to the user through these variables.

8.4.2 AB Dynamics Algorithm

The next step for the user is to compute the dynamics of the system using the AB algorithm developed in Chapter 7 (Tables 13–15). First, the user must set the inputs to the dynamics computation, `refForce` and `jointForce`. Then, these are passed as arguments to the `AB_Dynamics` member function of the `System` class as follows:

```
robot.AB_Dynamics(refForce, jointForce);
```

This function begins the computation the dynamics for the entire system and the accelerations that are computed will be stored in member variables of the respective objects that make up the system. Although the implementation of the algorithm is completely hidden from the user, this section presents an overview of the imple-

mentation of the AB dynamics algorithm, and to aid in this discussion, a graphical representation of this implementation is given in Figures 34 and 35.

In these figures, each box corresponds to one of the boxes of the object hierarchy. At the top of each, the class and the member function names are given. Pointing to each of these is the object name (which is a member variable of the encompassing box) which is used to reference them. The remainder of the contents of the box describes the computation that is performed in terms of the variable names that were used in Chapter 7. As before, the algorithm is divided into three parts: Forward Kinematics, Backward Dynamics, and Forward Accelerations, and the member functions are named accordingly.

In the Forward Kinematics step, the `robot` object begins by calling the reference member's (`ref`) `AB_Forward_Kinematics()` function to compute velocity, acceleration, and bias terms. When complete, it calls each serial chain's (`chain[k]`) `AB_Forward_Kinematics()` function. Each `chain[k]` object, in turn, computes velocity, acceleration, and $\zeta_{k,0}$ terms, sets up the `for`-loop for the forward recursion along the chain, and calls each of its links' (`link[i]`) `AB_Forward_Kinematics()` functions. At the end of the recursion, the serial chain calls its end effector, `ee`, function to compute the contact force at the end of the chain, ${}^{N}\mathbf{f}_{k,N+1}$, if present.

In the Backward Dynamics step, the `robot` object calls each of its chain's `AB_Backward_Dynamics()` functions. Each `chain[k]` object sets up the `for`-loop for the backward recursion, and calls each link's `AB_Backward_Dynamics()` function. At the end of this recursion, each `chain[k]` object computes ${}^{r}\mathbf{N}_{k,1}$ and ${}^{r}\mathbf{f}^{*}_{k,1}$

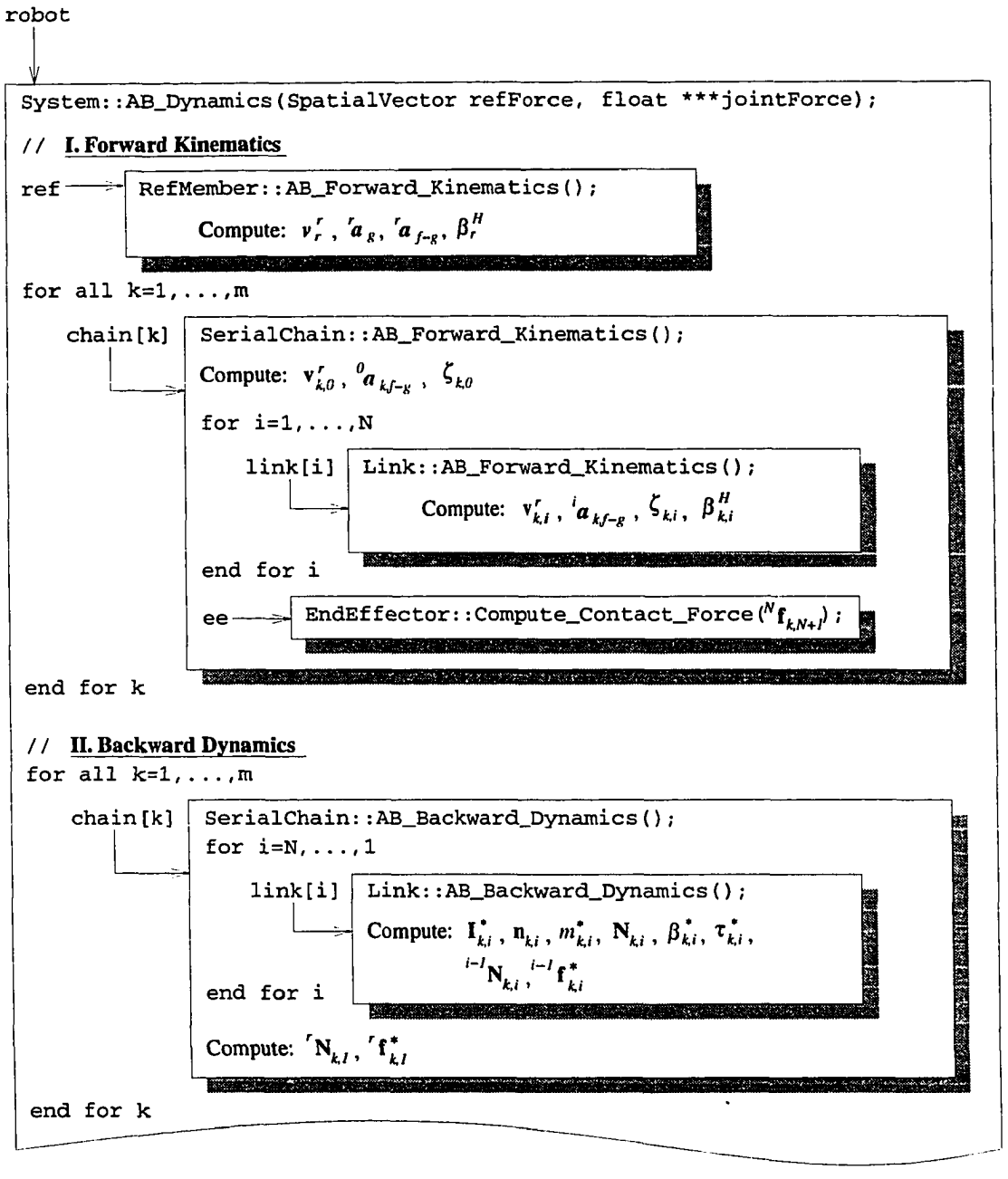


Figure 34: AB algorithm distribution among the **DynaMechs** classes.

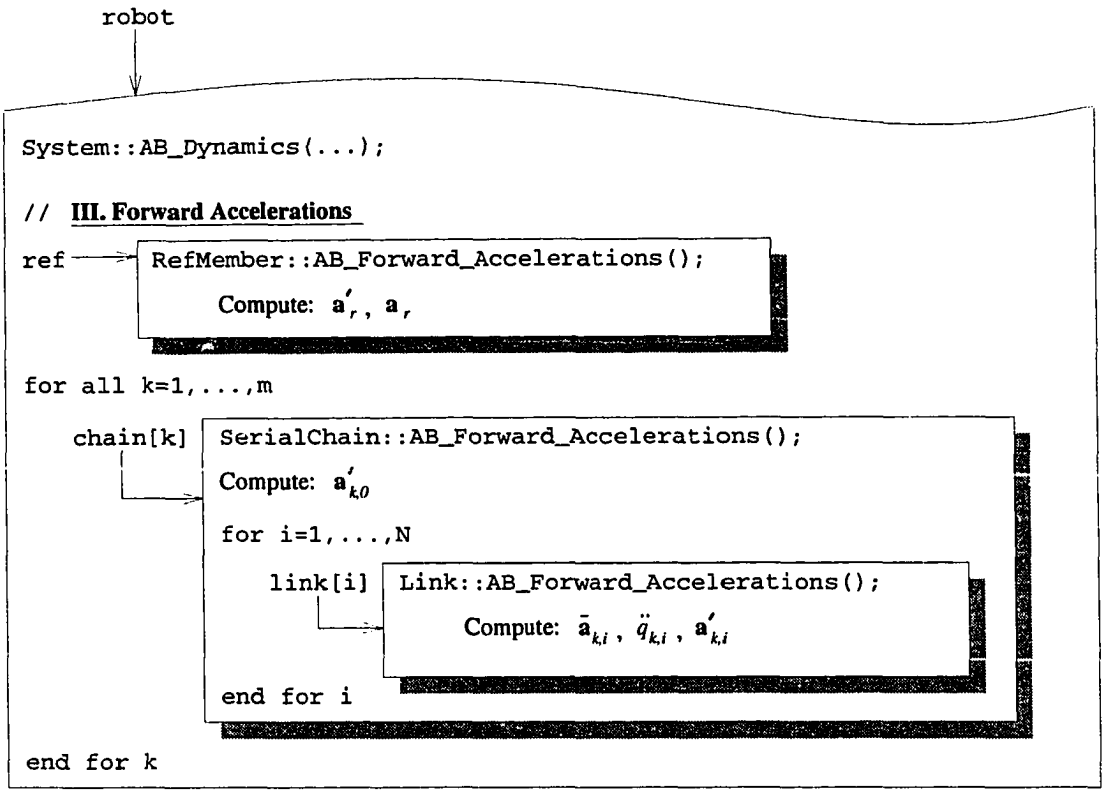


Figure 35: AB algorithm distribution among the **DynaMechs** classes (continued).

in preparation for the final step of the algorithm.

In the final Forward Accelerations step, the `robot` object begins by calling the reference member's `AB_Forward_Accelerations()` function to compute the biased reference member acceleration, $\mathbf{a}'_r$, that is needed by the serial chains, and the actual reference member acceleration, $\mathbf{a}_r$, that is needed for the simulation. Then, it calls each serial chain's (`chain[k]`), `AB_Forward_Accelerations()` function. Each `chain[k]` object, in turn, computes some initial quantities, sets up the `for`-loop for

the forward recursion, and calls the `AB_Forward_Accelerations()` function of each of its links (`link[i]`). At the end of this recursion, the reference member and link accelerations remain stored in the respective objects' member variables.

These figures also indicate how this algorithm can be parallelized because the iterations of the `for all k` loops can be performed simultaneously. By instantiating the `SerialChain` objects on different processors and providing for synchronization and communication between them and the `System` object, spatial parallelism can be implemented like that shown in Chapter 3 for multiple closed chain manipulators. The form the parallelism is also nearly the same except this algorithm requires two stages of serial computation for the reference member forward kinematics and acceleration.

8.4.3 Numerical Integration

The final step of simulation is to numerically integrate the accelerations to update the state of the system (the positions and velocities of the joints and the reference member). At present, only Euler integration is implemented to support real-time evaluation of the resulting system. To perform this step, the user executes another member function of the `System` class as follows:

```
robot.Update_State(idt, refPose, refPos, refVel, refAcc,
                        jointPos, jointVel, jointAcc);
```

The first parameter, `idt`, is the integration stepsize which is provided by the user. To achieve real-time operation, the system clock is monitored and the amount of time since the last call to `Update_State` is provided. The rest of the parameters are output

variables which supply the user with the accelerations that were computed in the last dynamics computation, and the updated state after the numerical integration is performed. This function is implemented in a distributed manner much like the dynamics algorithm explained in the previous section. The `System` function, calls the `RefMember` function to update its state, and then calls each of the `SerialChain` functions. The `SerialChain` functions call each of its `Link` functions to update the joint state.

8.4.4 Silicon Graphics Implementation

Another member function of the `System` class which graphically renders the resulting system is still under development at the Naval Postgraduate School (NPS) [63]. It is invoked as follows:

```
robot.draw();
```

Rendering is distributed among the classes like the `Update_State()` function. The `System` function calls component objects' `draw()` functions, and the `SerialChain` function calls its component objects' `draw()` functions. At each step, the graphics state of the machine is modified using the state of the object to be rendered (reference member position and orientation, MDH parameters of the links, etc.). Thus the position and orientation of each object is defined as needed by the graphics system and it is then rendered. The specific command issued to render the object depends on the particular modeling package used. To date SGI's Inventor [64] and Performer [65] packages, as well as NPS's Graphics Description Language (GDL2) [66] have been

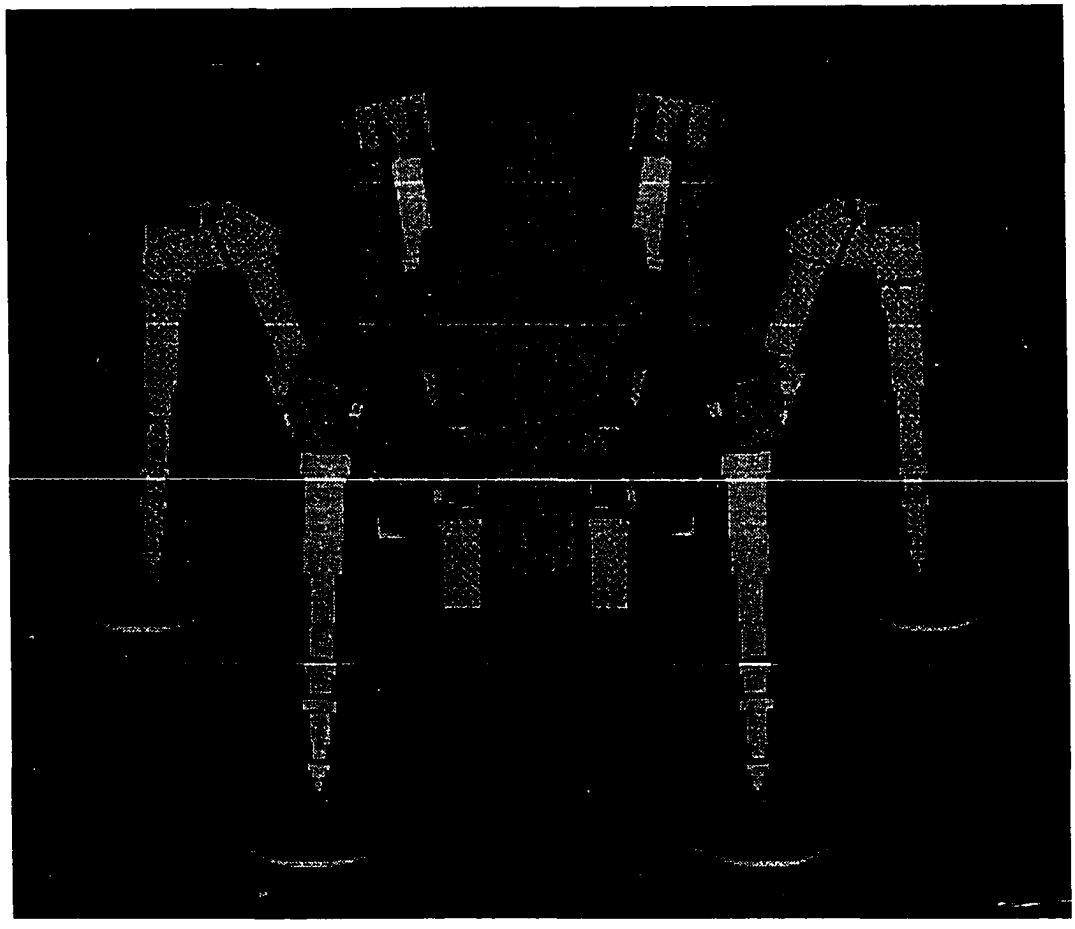


Figure 36: Scene from the *Aquarobot* simulation.

implemented for this purpose. Figure 36 shows the *Aquarobot* which was rendered on an Indigo^2 workstation with Extreme graphics using GDL2. Figure 37 shows *Tiburon* which was rendered using Inventor. It also contains a superimposed "heads up" display of the current joint torques as well as thruster output and the current control mode (not shown).

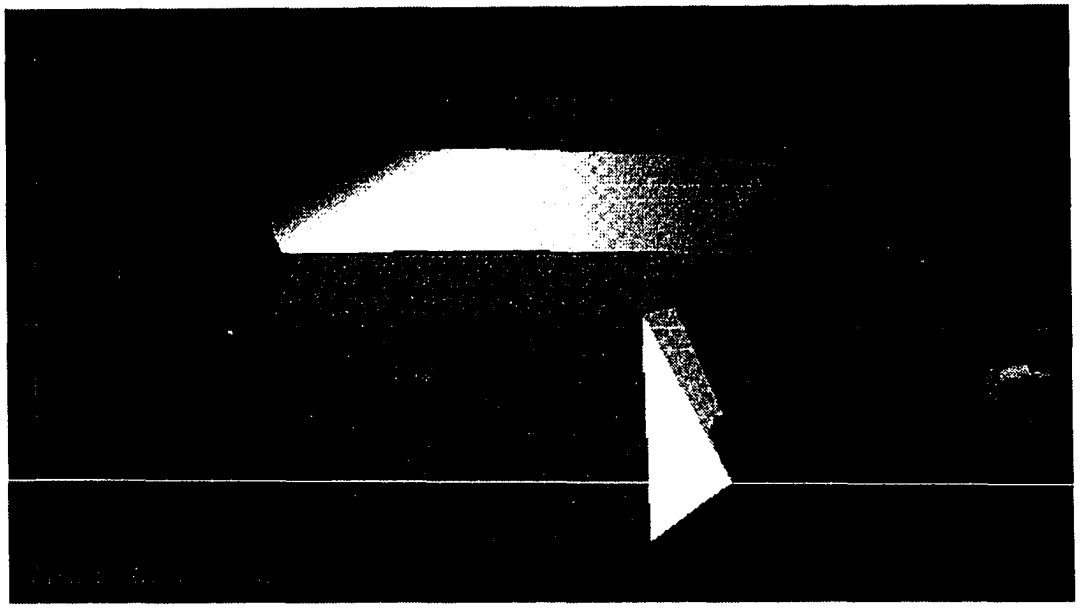


Figure 37: Scene from the *Tiburon* simulation.

The runtime performance of the simulation algorithm has been measured on the SGI system with a 150MHz MIPS R4400 processor. Due to the coarse resolution of the timing functions, runtime averages were taken over a hundred iterations of the dynamics computation. The performance for both systems using single precision arithmetic functions are given in Table 20. The ROV system requires about 0.7 ms on the average to perform one iteration of the simulation which should be adequate for real-time performance during normal operation of the ROV without contact with the environment. One iteration of the *Aquarobot* simulation requires about 2.6 ms. In either case and depending on the stiffness of the compliant contacts with the environment, these may or may not be adequate for real-time performance, and this subject requires further study.

Table 20: Runtime performance of simulations on SGI Indigo^2 workstation.

Robotic System	Stepsize Range (ms)	Stepsize Mean (ms)
Tiburon ($m = 1,\ N = 6$)	[0.68, 0.73]	0.7
Aquarobot ($m = 6,\ N = 4$)	[2.5, 2.8]	2.6

These performance results do not include rendering of the scenes themselves. The amount of time required to render is highly dependent on a number of factors: number and nature of the graphics elements (e.g., filled polygons) to be rendered, the graphics hardware available (e.g., Elan, Extreme, RealityEngine), and the software used to control the rendering process (GDL2, Inventor, Performer). The rendering process is faster with the more powerful graphics hardware. With an SGI RealityEngine, rendering is performed almost completely with the graphics hardware and does not affect the load on the CPU, and thus does not affect the performance of the simulation. As the capabilities of the graphics hardware decrease, significant CPU resources can be required which could significantly affect the performance of the resulting simulation. Multiprocessors and software packages like Performer can be used to distribute the graphics load, and Performer also has a feature to "cull" unseen graphics elements before rendering actually takes place. Both are being investigated at NPS to increase the performance of the resulting simulation system [63].

8.5 Summary and Conclusions

In this chapter, the implementation of the simulation algorithm was presented and the resulting software package, called **DynaMechs**[‡], has been described. A primary goal of this endeavor was to investigate object oriented design (OOD) techniques to implement the algorithm in C++ for a general class of robotic systems (star topologies) ranging from single chain systems with fixed bases to multiple chain systems with a mobile base. The first step was to decompose the system into its components and develop a general object hierarchy that is capable of describing the range of robotic systems just described. This forms the foundation for the next step in the OOD process which is the development of the class hierarchy which was used to define the types of objects that could be present in the system (e.g., mobile and fixed reference members, links paired with revolute and prismatic joints).

The two hierarchies provide the "road map" for decomposition and decoupling of the programming task. This process leads to OOD's main advantage over programming languages from previous generations which is the increased ability to manage the complexity in very large programming tasks. Some of the techniques that were employed in this implementation were single and multiple inheritance, abstract classes to support polymorphism, and most importantly, encapsulation (information hiding) techniques. The result is a very efficient simulation algorithm that is capable of simulating different kinds of systems, containing many different kinds of links, and with and without hydrodynamics. With the later addition of

[‡]This package is available via anonymous FTP from `taurus.cs.nps.navy.mil` in the `/pub/dynamechs/` directory.

the `BallnSocketLink` class, it has also proved its enhanced ability to easily extend the package. It has also proven this ability, by the ease with which various graphics packages have been included in the system for rendering the results.

The implementation of the algorithm is very efficient and yet it is still able to handle a wide variety of systems and link types. Runtime performance is very close to or exceeds real-time rates. Work is ongoing at NPS to achieve improved performance with different graphical packages and multiprocessors. For the latter, the parallelism discussed in Part I of this dissertation could be implemented on the multiprocessor SGI systems, and the speedups expected are comparable to those reported in Chapter 3. In addition, simplified dynamics models are also being investigated [67] to increase performance. The "full" dynamics implemented in this package will act as the baseline for measuring the viability of such approaches.

CHAPTER IX

Summary and Conclusions

9.1 Summary

In this dissertation, efficient dynamic simulation of simple closed chain mechanisms on land and under water has been investigated. These systems are characterized by a single reference member supported by a number of serial (actuated) chains. The development of efficient dynamics algorithms is motivated by the need for real-time and even super-real-time computational rates for advanced applications in design, development and operation of these systems. Based on new efficient transformation techniques and single chain dynamics algorithms reported in the appendices, more efficient multiple chain algorithms have been developed in this work. With these results, a new comparison between algorithms based on the Composite Rigid Body (CRB) and Articulated Body (AB) methods for the various systems considered here has been made.

In Part I, typical multiple chain, land-based systems are examined. The first system consists of multiple manipulators cooperating to lift a large load. Hard constraints are set by the grasps on the load and thus a closed chain algorithm results. The most efficient algorithm developed is based on one by Lilly and Orin [24] and

uses the CRB approach to compute some of the required dynamic quantities. The resulting method has a computational complexity of $O(mN^3)$ for an m manipulator system with N degrees of freedom each, and it is more efficient than $O(mN)$ AB-based algorithms for $N \leq 21$.

Parallelism of the resulting algorithm was examined to increase the computational rates obtainable by the simulation. Two forms of parallelism, spatial (structural) and temporal, were investigated and could also be combined for further increases in speedup. The algorithm was implemented on an eight processor Cray Y-MP supercomputer, and the simulation of a four PUMA system was performed. Speedups as high as 3.1 were achieved on four processors using spatial parallelism, and 3.97 on eight processors using combined parallelism. These algorithms break down in the presence of manipulator singularities; therefore, a new algorithm to handle this case was also developed. The increase to the computational requirements is minimal. This also led to the development of a new algorithm for dual arm systems, which requires *less* computation for cases with and without singularities.

Also studied in Part I was the simulation of multilegged vehicles. Although hard constraints at the contacts between the feet and ground have been considered in the past, it is more realistic to model these with compliance. Called the decoupled tree-structure (DTS) approach [7], this approach breaks the closed loops in the system, and open loop algorithms can be applied to the problem. For the resulting system, Freeman and Orin [7] developed an efficient algorithm based on the AB method. In this dissertation, a new algorithm is developed, referred to as the CRB/DTS

method, which extends the CRB algorithm to multiple chains. This algorithm is found to be more efficient than the AB approach for typical legged vehicles with three degree-of-freedom legs.

In Part II, simulation of underwater robotic vehicle (URV) systems is covered. Typical URV systems consist of a mobile base, such as a submarine, and a number of robotic manipulators. Also considered is a multilegged robot for underwater operation. In either case, the DTS approach can be used. Additional development to obtain the equations for hydrodynamics forces and investigation of efficient incorporation of these into existing land-based algorithms was performed. For these systems in an underwater environment, an AB-based algorithm was found to be the most efficient, and the method for incorporating the hydrodynamic effects and deriving the new algorithm have been presented.

As part of the development of a real-time graphical simulation system at the Naval Postgraduate School, an investigation into the implementation of the resulting hydrodynamic simulation algorithm using object oriented design (OOD) techniques was undertaken. The goal was to develop a software package in C++ that is capable of simulating a large class of tree-structured robotic systems having star topologies (when using DTS), and yet maintain the high level of efficiency achieved with the algorithms developed here. Success was achieved through the use of OOD mechanisms including object hierarchies, encapsulation, inheritance, and polymorphism. In addition, the resulting software package is also easily expandable and maintainable.

9.2 Future Work

This dissertation presents the development of computationally efficient algorithms for the simulation of simple closed chain mechanisms on land and under water. In particular, efficient, parallel and robust algorithms for multiple manipulator systems have been developed and implemented; legged vehicle simulation algorithms on land and under water have been derived; and a software package for dynamic simulation of a large class of robotic systems has been created. Although the contributions of this work are broad, considerable work still remains:

1. The simple closed chain mechanisms considered in this work contain serial chains. This may be extended to more general systems which contain chains with internal kinematic loops or the complete class of tree-structured topologies.

2. With DTS approaches, compliant contacts must be modeled. This requires further study to determine appropriate models (rather than the simple spring and damper systems) and the various parameters required for different surfaces. In addition, the effects of stiff surfaces on the numerical stability of the simulation algorithm should be studied.

3. Experimental validation has been performed for the code executed in the simulation of a single chain system with a fixed base. It was verified against the Composite Rigid Body method implemented in C for the simulation of a PUMA manipulator in [68]. Beyond this, there has been no experimental val-

idation of the correctness of the algorithms or software developed for systems with mobile bases. All that has been done is to demonstrate that the *Tiburon* and *Aquarobot* simulations seem qualitatively correct to a number of observers acquainted with dynamics and control of the real vehicles. Although the effort would be very time consuming and expensive, the ultimate validation of a simulation can result only by comparison with a physical system and this should be done.

4. Partial validation is possible with linearization and vibrational mode analysis. This has been done for massless-leg, walking machine models and previously unexpected errors in nonlinear simulations were detected [69]. This work needs to be extended to the more realistic models developed in this dissertation. Such models would also be useful for control system design [69].

5. Computation of fluid dynamics generally requires the solution of the Navier-Stokes equations which are extremely complex. The approach taken in this work is a simplification and computes lumped approximations of the hydrodynamic effects. Although research has been performed with single rigid bodies with simple shapes to determine the accuracy of this approach, experimental validation should be carried out for more complex systems.

6. The development of the **DynaMechs** software package is part of a project at the Naval Postgraduate School (NPS) to develop a real-time graphical simulation system for URV systems to aid in the development of new control

algorithms for an underwater legged vehicle. A number of additional suggestions can be made for the extension of the package to satisfy NPS's goal which include the addition of actuator models (DC and hydraulic). Real-time graphical user interfaces (GUIs) must also be developed for the simulator both for configuration input and operator interfaces which would lead to a number of virtual reality applications especially in the area of physically-based modeling.

7. The parallelism examined in Chapter 3 for multiple manipulator systems, can also be applied to the algorithm developed for this package to increase computational rates. Implementation of this on the new parallel SGI platforms could be undertaken.

8. Additional extensions for the software package include allowing for simulation of closed chain mechanisms with hard contacts such as the multiple manipulator system in Part I. These systems are also characterized by changing topological structures which require additional research to perform modeling of impacts when hard contacts are initiated.

9. To simulate typical space-based robotic systems, an additional `Link` subclass should be developed which models flexibility.

In conclusion, it is believed that the simulation algorithms presented here represent the most efficient ones developed to date for multiple-chain systems. The dissertation also contains a complete and accurate comparison of the two competing approaches, CRB and AB, in developing these algorithms. This dissertation has

also laid the groundwork for the development of a dynamic simulation software tool for multiple-chain robotic systems which, with real-time and super real-time rates, could be used in a number of different applications.

APPENDICES

Appendix A

Efficient Transformations with Modified Denavit-Hartenberg Parameters

A.1 Introduction

Transformations of vectors and matrices account for a significant portion of the operations in any of the dynamics algorithms developed in this dissertation. Hence, efficient implementations of these operations is very important for achieving an overall efficient algorithm. All transformations in this dissertation can be represented as one or more successive planar operations in which rotation and possibly translation occur about and around the same axis. It turns out to be most efficient to perform these transformations in multiple stages of planar operations. The procedure and computational requirements are the same regardless of which axis is used so in many of the presentations only one planar operation is discussed in detail and results for more complex operations are extrapolated from these results. This appendix begins with a presentation of the modified Denavit-Hartenberg (MDH) notation used extensively in this dissertation. Then the efficient transformations of Cartesian and spatial vectors and matrices that results with the use of this notation are developed.

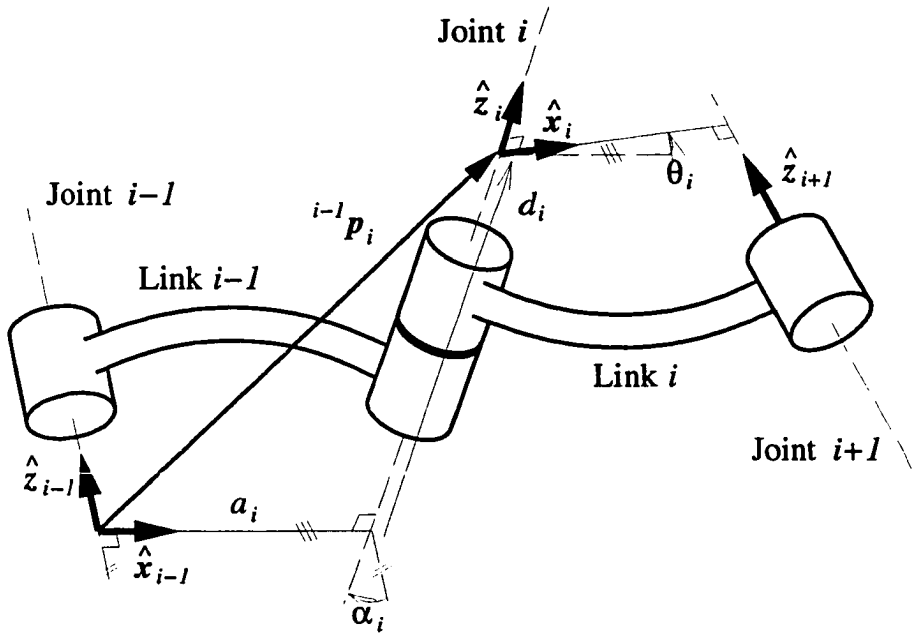


Figure 38: Link coordinate system assignment and MDH parameters.

A.2 Link Coordinate Systems and MDH Parameters

The most efficient dynamics algorithms for articulated mechanisms result when coordinate systems are assigned to each link in the system. For a vast majority of systems, the links in the serial chains are attached to one another with single degree-of-freedom revolute or prismatic joints. In these cases, it is especially efficient to assign these coordinate systems according to a set of rules similar to those first described by Denavit and Hartenberg in [70]. The resulting modified Denavit-Hartenberg (MDH) notation has also been presented by Craig in [26].

Using this notation, the links are numbered, in succession, from link 1, attached

to the base, to the last link at the tip or end-effector, link N. As shown in Figure 38, the joint between links $i-1$ and i is labeled joint i, and the origin of coordinate frame i lies on the axis of this joint's motion. Further, this coordinate frame is fixed to link i at the end "nearest" the base.

The axes of each coordinate system are aligned so that the z-axis, $\hat{\boldsymbol{z}}$ lies along the axis of motion of the joint. This is the axis about which the joint rotates for revolute joints, and along which the joint translates for prismatic joints. The x-axis, $\hat{\boldsymbol{x}}$, lies along the common normal between this joint axis and the next one in the chain as shown in Figure 38. With this arrangement, only four parameters are needed to describe the relative position and orientation between adjacent coordinate systems. To move from link $i-1$'s coordinate system to link i, these parameters are defined as follows:

a_i = the perpendicular distance along $\hat{\boldsymbol{x}}_{i-1}$ from $\hat{\boldsymbol{z}}_{i-1}$ to $\hat{\boldsymbol{z}}_i$,

α_i = the angle about $\hat{\boldsymbol{x}}_{i-1}$ from $\hat{\boldsymbol{z}}_{i-1}$ to $\hat{\boldsymbol{z}}_i$,

d_i = the perpendicular distance along $\hat{\boldsymbol{z}}_i$ from $\hat{\boldsymbol{x}}_{i-1}$ to $\hat{\boldsymbol{x}}_i$, and

θ_i = the angle about $\hat{\boldsymbol{z}}_i$ from $\hat{\boldsymbol{x}}_{i-1}$ to $\hat{\boldsymbol{x}}_i$,

where d_i is variable if joint i is prismatic, or θ_i is variable if it is revolute. Note that when the base of the chain is fixed with respect to an inertial frame, it is convenient to place the origin of coordinate frame 0 coincident with coordinate system 1 with its z-axis also along joint 1. As a result, the only non-zero parameter is the joint variable, θ_1 or d_1.

These parameters are used to define the rotational transformation matrix and

position vector from one coordinate frame to the next. The rotational transformation from coordinate frame $i-1$ to frame i is defined as follows:

$$ {}^{i}\boldsymbol{R}_{i-1} = \begin{bmatrix} c\theta_i & s\theta_i c\alpha_i & s\theta_i s\alpha_i \\ -s\theta_i & c\theta_i c\alpha_i & c\theta_i s\alpha_i \\ 0 & -s\alpha_i & c\alpha_i \end{bmatrix}, \tag{A.1} $$

where $s\alpha_i$, $c\alpha_i$, $s\theta_i$, and $c\theta_i$ denote $\sin\alpha_i$, $\cos\alpha_i$, $\sin\theta_i$, and $\cos\theta_i$, respectively. This can also be expressed as two planar rotations:

$$ {}^{i}\boldsymbol{R}_{i-1} = \boldsymbol{R}(z,\theta_i)\,\boldsymbol{R}(x,\alpha_i), \tag{A.2} $$

$$ = \begin{bmatrix} c\theta_i & s\theta_i & 0 \\ -s\theta_i & c\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\alpha_i & s\alpha_i \\ 0 & -s\alpha_i & c\alpha_i \end{bmatrix}. \tag{A.3} $$

The 3×1 position vector from the origin of frame $i-1$ to the origin of frame i (with components expressed in frame $i-1$) shown in Figure 38 is defined as follows:

$$ {}^{i-1}\boldsymbol{p}_i = \begin{bmatrix} a_i \\ -d_i\, s\alpha_i \\ d_i\, c\alpha_i \end{bmatrix}, \tag{A.4} $$

which is constant when joint i is revolute.

These two quantities are also used to define the 6×6 spatial transformation matrix, ${}^{i}\mathbf{X}_{i-1}$, which is used to transform spatial vectors between adjacent coordinate systems. The spatial transformation between joints $i-1$ and i is expressed as follows:

$$ {}^{i}\mathbf{X}_{i-1} = \begin{bmatrix} {}^{i}\boldsymbol{R}_{i-1} & \mathbf{0} \\ {}^{i}\boldsymbol{R}_{i-1}\,{}^{i-1}\tilde{\boldsymbol{p}}_i^{\,T} & {}^{i}\boldsymbol{R}_{i-1} \end{bmatrix}, \tag{A.5} $$

where the tilde above the position vector signifies that its components should be combined in a skew symmetric matrix such that $\tilde{\boldsymbol{b}}\boldsymbol{a} = \boldsymbol{b} \times \boldsymbol{a}$. This matrix can also be expressed as successive planar screw transformations about the x and z axes as

follows:

$$
\begin{aligned}
{}^{i}\mathbf{X}_{i-1} &= \mathbf{X}(z, d_i, \theta_i)\, \mathbf{X}(x, a_i, \alpha_i), && \text{(A.6)} \\
&= \begin{bmatrix} \boldsymbol{R}(z,\theta_i) & \mathbf{0} \\ \boldsymbol{R}(z,\theta_i)\, \tilde{\boldsymbol{p}}_z^T & \boldsymbol{R}(z,\theta_i) \end{bmatrix} \begin{bmatrix} \boldsymbol{R}(x,\alpha_i) & \mathbf{0} \\ \boldsymbol{R}(x,\alpha_i)\, \tilde{\boldsymbol{p}}_x^T & \boldsymbol{R}(x,\alpha_i) \end{bmatrix}, && \text{(A.7)}
\end{aligned}
$$

where $\boldsymbol{p}_x = [a_i\ 0\ 0]^T$ and $\boldsymbol{p}_z = [0\ 0\ d_i]^T$.

A.3 Vector Transformations

In this section, transformations of Cartesian and spatial vectors are presented. Examples using only the z-axis planar transformation are given which were also reported by Featherstone ([27], p. 146). The computational cost of successive planar transformations on the result is the same so that the cost of transforming vectors using the MDH parameters from the previous section can be obtained from this result directly.

A.3.1 Cartesian Vectors

The rotation of a vector about the z-axis through an angle θ is accomplished by the following multiplication:

$$
\begin{aligned}
\boldsymbol{R}(z,\theta)\, \boldsymbol{p} &= \boldsymbol{p}' && \text{(A.8)} \\
\begin{bmatrix} \mathrm{c}\theta & \mathrm{s}\theta & 0 \\ -\mathrm{s}\theta & \mathrm{c}\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} &= \begin{bmatrix} \mathrm{c}\theta p_x + \mathrm{s}\theta p_y \\ \mathrm{c}\theta p_y - \mathrm{s}\theta p_x \\ p_z \end{bmatrix}. && \text{(A.9)}
\end{aligned}
$$

The computational cost of this step is only [4M, 2A], which is the same if the rotation is carried out about either of the other two axes. The rotation of a vector from link $(i-1)$ to link i's coordinate system using MDH parameters requires a

rotation of α_i about the x-axis and then θ_i about the z-axis as follows:

$$\boldsymbol{p}_i \;=\; \boldsymbol{R}(z,\theta_i)\;\boldsymbol{R}(x,\alpha_i)\;\boldsymbol{p}_{i-1}. \tag{A.10}$$

Each planar rotation requires [4M, 2A] for a total cost of [8M, 4A]. This is much more efficient than computing ${}^{i}\boldsymbol{R}_{i-1}$ from Eq. (A.1), requiring [4M, 0A], and performing the matrix-vector multiply requiring [9M, 6A]. It should also be noted that rotations in the opposite direction, which require ${}^{i}\boldsymbol{R}_{i-1}^{T}$, require [8M, 4A] as well.

A.3.2 Spatial Vectors

Successive planar screw transformation operations, which require rotation and translation about the same axis, can be used to transform spatial vectors from one coordinate system to another. For example rotating a spatial velocity vector, $\mathbf{v}$, about the z-axis through an angle θ and translating along it through a distance, d, can be accomplished with the following matrix operation:

$$\mathbf{X}(z,d,\theta)\;\mathbf{v} \;=\; \mathbf{v}', \tag{A.11}$$

$$\begin{bmatrix} \mathrm{c}\theta & \mathrm{s}\theta & 0 & 0 & 0 & 0 \\ -\mathrm{s}\theta & \mathrm{c}\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ -\mathrm{s}\theta d & \mathrm{c}\theta d & 0 & \mathrm{c}\theta & \mathrm{s}\theta & 0 \\ -\mathrm{c}\theta d & -\mathrm{s}\theta d & 0 & -\mathrm{s}\theta & \mathrm{c}\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \mathrm{c}\theta\omega_x + \mathrm{s}\theta\omega_y \\ \mathrm{c}\theta\omega_y - \mathrm{s}\theta\omega_x \\ \omega_z \\ \mathrm{c}\theta\,(v_x + \omega_y d) + \mathrm{s}\theta\,(v_y - \omega_x d) \\ \mathrm{c}\theta\,(v_y - \omega_x d) - \mathrm{s}\theta\,(v_x + \omega_y d) \\ v_z \end{bmatrix}. \tag{A.12}$$

This operation requires [10M, 6A]. The transformation of a spatial vector between links using all of the MDH parameters involves an x-planar screw as well:

$$\mathbf{v}_i \;=\; \mathbf{X}(z,d,\theta)\;\mathbf{X}(x,a,\alpha)\;\mathbf{v}_{i-1}. \tag{A.13}$$

Each screw transformation requires [10M, 6A] for a total of [20M, 12A]. Spatial transformations of forces in the other direction requires the transpose of these matrices. Nevertheless, the computational cost remains the same for this operation as well.

A.4 Matrix Congruence Transformations

In order to transform inertia matrices from one coordinate system to the next, a congruence transformation operation is required. In this section, transformations for symmetric and general 3×3 matrices are examined in detail. The results represent the most efficient that have been reported in the robotics literature. These operations form the basis for the more complex transformations involving symmetric 6×6 matrices such as the operational space inertias used in Part I and Articulated Body (AB) inertias used in Part II of this dissertation. Finally, the transformation of specialized 6×6 matrices such as the reflected AB inertia and the spatial inertia, which has the same form as a CRB inertia matrix, are covered.

A.4.1 Symmetric 3×3 Matrices

First, the efficient procedure for performing the congruence transformation of a symmetric 3×3 matrix, $\bar{\boldsymbol{K}}_i$, from link i to link $(i-1)$'s coordinate system is presented. The transformation matrix is ${}^{i}\boldsymbol{R}_{i-1}$, and has already been shown to consist of two planar rotations. Therefore, this computation can be divided into two steps: a z-axis planar congruence transform through the angle θ_i and an x-axis planar congruence transform through the angle α_i.

The first transformation results in the following matrix:

$$\bar{K}^z = \boldsymbol{R}^T(z,\theta)\,\bar{\boldsymbol{K}}_i\boldsymbol{R}(z,\theta) \tag{A.14}$$

$$= \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{12} & k_{22} & k_{23} \\ k_{13} & k_{23} & k_{33} \end{bmatrix} \begin{bmatrix} c\theta & s\theta & 0 \\ -s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.15}$$

$$= \begin{bmatrix} k_{11}+a & b & k_{13}c\theta - k_{23}s\theta \\ b & k_{22}-a & k_{13}s\theta + k_{23}c\theta \\ k_{13}c\theta - k_{23}s\theta & k_{13}s\theta + k_{23}c\theta & k_{33} \end{bmatrix}, \tag{A.16}$$

where

$$a = (k_{22}-k_{11})s^2\theta - k_{12}(2s\theta c\theta) \tag{A.17}$$

$$b = k_{12}(c^2\theta - s^2\theta) - (k_{22}-k_{11})(s\theta c\theta). \tag{A.18}$$

The number of operations required to compute this matrix are given in the following table:

Computation	×	+
$s^2\theta = \sin\theta\sin\theta$	1	0
$c^2\theta - s^2\theta = 1 - s^2\theta - s^2\theta$	0	2
$s\theta c\theta = \sin\theta\cos\theta$	1	0
$2s\theta c\theta = s\theta c\theta + s\theta c\theta$	0	1
$k_{22} - k_{11}$	0	1
$(k_{22}-k_{11})(s\theta c\theta)$, $(k_{22}-k_{11})s^2\theta$	2	0
$k_{12}(c^2\theta - s^2\theta)$, $k_{12}(2s\theta c\theta)$	2	0
a, b	0	2
Elements $k^z_{33} = k_{33}$, $k^z_{12} = b$	0	0
Elements k^z_{11}, k^z_{22}	0	2
Elements k^z_{13}, k^z_{23}	4	2

Note that the resulting matrix is also symmetric so only the upper triangular elements need to be computed resulting in [10M, 10A].

This result is used in the x-planar congruence transform to complete the opera-

tion as follows:

$$^{i-1}\bar{K}_i = \boldsymbol{R}^T(x,\alpha)\,\bar{K}^z\boldsymbol{R}(x,\alpha). \tag{A.19}$$

The procedure for breaking down this computation into the fewest number of operations is the same as that found for the z-planar transformation. Note, however, that the link twist angle, α, is constant and the trigonometry functions involving this angle are constant and can be computed off-line. Therefore, this step requires [8M, 7A], and a total of [18M, 17A] is needed for the complete congruence transformation.

A.4.2 General 3×3 Matrices

To find an efficient method for transforming a general 3×3 matrix, a procedure similar to the one used above is followed. The z-planar transformation results in the following matrix:

$$\bar{N}^z = \boldsymbol{R}^T(z,\theta)\,\bar{N}_i\boldsymbol{R}(z,\theta) \tag{A.20}$$

$$= \begin{bmatrix} \mathrm{c}\theta & -\mathrm{s}\theta & 0 \\ \mathrm{s}\theta & \mathrm{c}\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} \begin{bmatrix} \mathrm{c}\theta & \mathrm{s}\theta & 0 \\ -\mathrm{s}\theta & \mathrm{c}\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.21}$$

$$= \begin{bmatrix} n_{11}+a & n_{12}-b & n_{13}\mathrm{c}\theta - n_{23}\mathrm{s}\theta \\ n_{21}-b & n_{22}-a & n_{13}\mathrm{s}\theta + n_{23}\mathrm{c}\theta \\ n_{31}\mathrm{c}\theta - n_{32}\mathrm{s}\theta & n_{31}\mathrm{s}\theta + n_{32}\mathrm{c}\theta & n_{33} \end{bmatrix}, \tag{A.22}$$

where

$$a = (n_{22}-n_{11})\mathrm{s}^2\theta - (n_{12}+n_{21})\mathrm{s}\theta\mathrm{c}\theta \tag{A.23}$$

$$b = (n_{22}-n_{11})\mathrm{s}\theta\mathrm{c}\theta + (n_{12}+n_{21})\mathrm{s}^2\theta. \tag{A.24}$$

The number of operations required to compute this matrix are given in the following table:

Computation	×	+
$s^2\theta, s\theta c\theta$	2	0
$(n_{22}-n_{11})s^2\theta,\ (n_{22}-n_{11})(s\theta c\theta)$	2	1
$(n_{12}+n_{21})s^2\theta,\ (n_{12}+n_{21})(s\theta c\theta)$	2	1
$a,\ b$	0	2
Elements $n^z_{11},\ n^z_{12},\ n^z_{21},\ n^z_{22}$	0	4
Elements $n^z_{13},\ n^z_{31},\ n^z_{32},\ n^z_{23}$	8	4
Element $n^z_{33} = n_{33}$	0	0

The computational cost of this step is [14M, 12A]. To complete the transformation, the x-planar transformation proceeds in a similar fashion but only requires [12M, 12A] because α is constant when using MDH parameters and the trigonometric computations can be computed off-line.

A.4.3 Symmetric 6×6 Matrices

This section presents the details required to efficiently transform both operational space inertia matrices that are used in Chapter 2 and AB inertia matrices that are used in Chapter 7. Efficient computation of spatial congruence transformations requires the same approach as before by computing planar screw transformations in succession. Therefore the congruence transformation is written as follows:

$$\mathbf{X}^T\mathbf{\Lambda}\mathbf{X} \;=\; \mathbf{X}^T(x,a,\alpha)\,\mathbf{X}^T(z,d,\theta)\,\mathbf{\Lambda}\,\mathbf{X}(z,d,\theta)\,\mathbf{X}(x,a,\alpha). \tag{A.25}$$

Each planar congruence transform is performed separately using the z planar screw transformation first, and then performing the x planar screw transformation on this result.

First the z planar screw transformation step is written in terms of its Cartesian

Table 21: Computational cost of a 6×6 planar congruence transformation.

Step	Computation	$\times$	$+$
(A)	$s^2\theta$, $c^2\theta$ - $s^2\theta$, $s\theta c\theta$, $2s\theta c\theta$	2	3
(B)	$\boldsymbol{R}^T\boldsymbol{\Lambda}_{11}\boldsymbol{R}$	8	7
(C)	$[\mathbf{X}^T\boldsymbol{\Lambda}\mathbf{X}]_{22} = \boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R}$	8	7
(D)	$\boldsymbol{R}^T\boldsymbol{\Lambda}_{12}\boldsymbol{R}$	12	12
(E)	$\tilde{\boldsymbol{p}}(\boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R})$	5	0
(F)	$(\tilde{\boldsymbol{p}}\boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R})\tilde{\boldsymbol{p}}^T$	3	0
(G)	$(\boldsymbol{R}^T\boldsymbol{\Lambda}_{12}\boldsymbol{R})\tilde{\boldsymbol{p}}^T$	6	0
(H)	$[\mathbf{X}^T\boldsymbol{\Lambda}\mathbf{X}]_{11}$ = (B)+(F)+(G)+(G)^T	0	11
(I)	$[\mathbf{X}^T\boldsymbol{\Lambda}\mathbf{X}]_{12} = [\mathbf{X}^T\boldsymbol{\Lambda}\mathbf{X}]_{21}^T$ = (D)+(E)	0	6
	Total:	44	46

components as follows:

$$\boldsymbol{\Lambda}^z = \mathbf{X}^T(z,d,\theta)\,\boldsymbol{\Lambda}\,\mathbf{X}(z,d,\theta) \tag{A.26}$$

$$= \begin{bmatrix} \boldsymbol{R}^T & \tilde{\boldsymbol{p}}\boldsymbol{R}^T \\ \mathbf{0} & \boldsymbol{R}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \mathbf{0} \\ \boldsymbol{R}\tilde{\boldsymbol{p}}^T & \boldsymbol{R} \end{bmatrix} \tag{A.27}$$

$$= \left[\begin{array}{l|l} \boldsymbol{R}^T\boldsymbol{\Lambda}_{11}\boldsymbol{R} + \tilde{\boldsymbol{p}}\boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R}\tilde{\boldsymbol{p}}^T + & \boldsymbol{R}^T\boldsymbol{\Lambda}_{12}\boldsymbol{R} + \tilde{\boldsymbol{p}}\boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R} \\ \quad \boldsymbol{R}^T\boldsymbol{\Lambda}_{12}\boldsymbol{R}\tilde{\boldsymbol{p}}^T + \left(\boldsymbol{R}^T\boldsymbol{\Lambda}_{12}\boldsymbol{R}\tilde{\boldsymbol{p}}^T\right)^T & \\ \hline \left(\boldsymbol{R}^T\boldsymbol{\Lambda}_{12}\boldsymbol{R} + \tilde{\boldsymbol{p}}\boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R}\right)^T & \boldsymbol{R}^T\boldsymbol{\Lambda}_{22}\boldsymbol{R} \end{array}\right], \tag{A.28}$$

where $\boldsymbol{R}$ and $\boldsymbol{p}$ are $\boldsymbol{R}(z,d,\theta)$ and $\boldsymbol{p}_z$, respectively, and defined earlier in this appendix. Nine steps are identified in this computation and listed in Table 21.

Note that the numbers reported in this table rely heavily on the previous results of this section for transformation of 3×3 matrices. The trigonometric functions needed for 3×3 symmetric and general matrices overlap; therefore, they are listed separately in Step (A). The remainder of the computation for the symmetric and general 3×3 congruence transformations are listed in Steps (B), (C), and (D).

The multiplication of a 3×3 matrix by $\tilde{\boldsymbol{p}}$ in Steps (E), (F), and (G) are equivalent to performing three cross products with a vector whose third element is the only non-zero element. In Step (E), the matrix is symmetric so only [5M, 0A] are required and the third row of the result is zero. The result of Step (F) is symmetric and has only four non-zero (three in the upper triangular portion) elements which requires [3M, 0A] to compute. Unlike Step (E), Step (G) begins with a general matrix so that [6M, 0A] are required to produce the result which also has a zero row.

Step (H) involves the addition of four 3×3 matrices whose result is symmetric so that only six elements need to be determined. Since the matrices in Steps (F) and (G) have zero elements, the amount of computation for this step is only [0M, 11A]. The same happens in Step (I) with the result of Step (E). The result is a general 3×3 matrix requiring [0M, 6A].

To complete the 6×6 congruence transformation when using MDH parameters, the steps in Table 21 are repeated for the x-planar screw. Since α is always constant, the computations for Step (A) may be completed off-line. The rest of the steps have the same computational cost as listed in the table for a total of [42M, 43A], and the complete transformation requires a total of [86M, 89A]. In Chapter 2, a full set of constant MDH parameters is used to transform the operational space inertia matrix, $\boldsymbol{\Lambda}_k$ to the reference member's coordinate system. In this case both θ and α are constant so the computational cost of this step is [84M, 86A]. In Chapter 7, a constant z planar screw transformation is needed to transform the reflected AB inertia matrix, ${}^0\mathbf{N}_{k,1}$, from a chain's base coordinate system to the reference member's coordinate

system. This results in a computational cost of [42M, 43A].

A.4.4 Reflected AB Inertia Matrices

In the algorithm developed in Part II, the congruence transformation of the reflected inertia is another computationally expensive step. This step is very similar to the transformation of a symmetric 6×6 matrix that was described in the previous section. The structure of reflected AB inertia matrices contain extra zeros which lead to a more efficient implementation than in the previous section. If joint i is revolute, where $\phi_i = [0\ 0\ 1\ 0\ 0\ 0]^T$, the third row and column of the ith reflected inertia matrix, $\mathbf{N}_i$, are zero as follows:

$$\mathbf{N}_i = \left[\begin{array}{c|c} \mathbf{N}_{11} & \mathbf{N}_{12} \\ \hline \mathbf{N}_{21} & \mathbf{N}_{22} \end{array}\right] \tag{A.29}$$

$$= \left[\begin{array}{ccc|ccc} n_{11} & n_{12} & 0 & n_{14} & n_{15} & n_{16} \\ n_{12} & n_{22} & 0 & n_{24} & n_{25} & n_{26} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline n_{14} & n_{24} & 0 & n_{44} & n_{45} & n_{46} \\ n_{15} & n_{25} & 0 & n_{45} & n_{55} & n_{56} \\ n_{16} & n_{26} & 0 & n_{46} & n_{56} & n_{66} \end{array}\right]. \tag{A.30}$$

For a prismatic joint, $\phi_i = [0\ 0\ 0\ 0\ 0\ 1]^T$ and so the sixth column and row are zero:

$$\mathbf{N}_i = \left[\begin{array}{ccc|ccc} n_{11} & n_{12} & n_{13} & n_{14} & n_{15} & 0 \\ n_{12} & n_{22} & n_{23} & n_{24} & n_{25} & 0 \\ n_{13} & n_{23} & n_{33} & n_{34} & n_{35} & 0 \\ \hline n_{14} & n_{24} & n_{34} & n_{44} & n_{45} & 0 \\ n_{15} & n_{25} & n_{35} & n_{45} & n_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]. \tag{A.31}$$

Due to the difference in the structure for revolute and prismatic joints, both cases are covered in this section.

The z-planar congruence transform is performed first, and the same nine steps that were identified in Table 21 of the previous section apply to this computation.

Table 22: Computational cost for a z-planar congruence transformation of a reflected AB inertia.

		Revolute		Prismatic	
Step	Computation	×	+	×	+
(A)	$s^2\theta$, $c^2\theta$ - $s^2\theta$, $s\theta c\theta$, $2s\theta c\theta$	2	3	2	3
(B)	$\boldsymbol{R}^T\mathbf{N}_{11}\boldsymbol{R}$	4	5	8	7
(C)	$[\mathbf{X}^T\mathbf{N}\mathbf{X}]_{22} = \boldsymbol{R}^T\mathbf{N}_{22}\boldsymbol{R}$	8	7	4	5
(D)	$\boldsymbol{R}^T\mathbf{N}_{12}\boldsymbol{R}$	8	10	8	10
(E)	$\tilde{\boldsymbol{p}}(\boldsymbol{R}^T\mathbf{N}_{22}\boldsymbol{R})$	5	0	3	0
(F)	$(\tilde{\boldsymbol{p}}\boldsymbol{R}^T\mathbf{N}_{22}\boldsymbol{R})\tilde{\boldsymbol{p}}^T$	3	0	3	0
(G)	$(\boldsymbol{R}^T\mathbf{N}_{12}\boldsymbol{R})\tilde{\boldsymbol{p}}^T$	4	0	6	0
(H)	$[\mathbf{X}^T\mathbf{N}\mathbf{X}]_{11}$ = (B)+(F)+(G)+$(G)^T$	0	9	0	11
(I)	$[\mathbf{X}^T\mathbf{N}\mathbf{X}]_{12} = [\mathbf{X}^T\mathbf{N}\mathbf{X}]_{21}^T$ = (D)+(E)	0	6	0	4
	Total:	34	40	34	40

These are listed in Table 22 for both types of joints. Because of the extra zeros, Steps (B), (D), (G), and (H) require less computation than the previous section for a revolute joint. For a prismatic joint, Steps (C), (D), (E), and (I) require less computation. The computation requires [34M, 40A] for either type of joint, and more importantly, the zero rows and columns that are shown in Eqs. (A.30) and (A.31) are present in the final result of this step as well.

Because of these zero rows, the computations for the x-planar congruence transformation are listed in Table 23. In this case, the computation for Step (A) can be accomplished off-line as before because α is a constant. Steps (B) and (D) require less computation in the case of a revolute joint, and Steps (C) and (D) require less computation in the case of a prismatic joint. The final result in either case is a full symmetric 6×6 matrix, and the total computational cost is [70M, 71A].

Table 23: Computational cost for a x-planar congruence transformation of a reflected AB inertia.

		Revolute		Prismatic	
Step	Computation	×	+	×	+
(A)	$s^2\alpha$, $c^2\alpha$ - $s^2\alpha$, $sc\alpha$, $2sc\alpha$	0	0	0	0
(B)	$\boldsymbol{R}^T\mathbf{N}_{11}\boldsymbol{R}$	4	1	8	7
(C)	$[\mathbf{X}^T\mathbf{N}\mathbf{X}]_{22} = \boldsymbol{R}^T\mathbf{N}_{22}\boldsymbol{R}$	8	7	4	1
(D)	$\boldsymbol{R}^T\mathbf{N}_{12}\boldsymbol{R}$	10	6	10	6
(E)	$\tilde{\boldsymbol{p}}(\boldsymbol{R}^T\mathbf{N}_{22}\boldsymbol{R})$	5	0	5	0
(F)	$(\tilde{\boldsymbol{p}}\boldsymbol{R}^T\mathbf{N}_{22}\boldsymbol{R})\tilde{\boldsymbol{p}}^T$	3	0	3	0
(G)	$(\boldsymbol{R}^T\mathbf{N}_{12}\boldsymbol{R})\tilde{\boldsymbol{p}}^T$	6	0	6	0
(H)	$[\mathbf{X}^T\mathbf{N}\mathbf{X}]_{11}$ = (B)+(F)+(G)+(G)^T	0	11	0	11
(I)	$[\mathbf{X}^T\mathbf{N}\mathbf{X}]_{12} = [\mathbf{X}^T\mathbf{N}\mathbf{X}]_{21}^T$ = (D)+(E)	0	6	0	6
	Total:	36	31	36	31

A.4.5 Spatial Inertia Matrices

The spatial inertia matrix, which was described in Chapter 1, has a very specific structure which is the same as the Composite Rigid Body (CRB) inertias, $\mathbf{K}_i$, that are used in the computation of the joint space inertia matrices in Chapters 2 and 5. The structure is repeated here for reference:

$$\mathbf{K}_i = \begin{bmatrix} \bar{\boldsymbol{K}}_i^c & \tilde{\boldsymbol{h}}_i^c \\ \left(\tilde{\boldsymbol{h}}_i^c\right)^T & m_i^c\mathbf{1}_3 \end{bmatrix}, \tag{A.32}$$

where $\bar{\boldsymbol{K}}_i^c$ is the moment of inertia (3×3, symmetric matrix), $\boldsymbol{h}_i^c$ is the first moment of mass (3-vector), and m_i^c is the mass (scalar) of the ith composite rigid body. Note that only ten elements are needed to completely specify the corresponding spatial inertia matrix, and this will be used to significantly reduce the cost of a congruence transformation involving it.

First the z planar screw transformation step is written in terms of its Cartesian components as follows:

$$\mathbf{K}^z = \mathbf{X}^T(z,d,\theta)\,\mathbf{K}\,\mathbf{X}(z,d,\theta), \tag{A.33}$$

$$\begin{bmatrix} \bar{\boldsymbol{K}}^z & \tilde{\boldsymbol{h}}^z \\ \left(\tilde{\boldsymbol{h}}^z\right)^T & m^z\mathbf{1}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}^T & \tilde{\boldsymbol{p}}\boldsymbol{R}^T \\ \mathbf{0} & \boldsymbol{R}^T \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{K}}^c & \tilde{\boldsymbol{h}}^c \\ \left(\tilde{\boldsymbol{h}}^c\right)^T & m^c\mathbf{1}_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{R} & \mathbf{0} \\ \boldsymbol{R}\tilde{\boldsymbol{p}}^T & \boldsymbol{R} \end{bmatrix}, \tag{A.34}$$

$$= \left[\begin{array}{l|l} \boldsymbol{R}^T\bar{\boldsymbol{K}}^c\boldsymbol{R} + \tilde{\boldsymbol{p}}\boldsymbol{R}^T m^c\mathbf{1}_3\boldsymbol{R}\tilde{\boldsymbol{p}}^T + & \boldsymbol{R}^T\tilde{\boldsymbol{h}}^c\boldsymbol{R} + \tilde{\boldsymbol{p}}\boldsymbol{R}^T m^c\mathbf{1}_3\boldsymbol{R} \\ \quad \boldsymbol{R}^T\tilde{\boldsymbol{h}}^c\boldsymbol{R}\tilde{\boldsymbol{p}}^T + \left(\boldsymbol{R}^T\tilde{\boldsymbol{h}}^c\boldsymbol{R}\tilde{\boldsymbol{p}}^T\right)^T & \\ \hline \left(\boldsymbol{R}^T\tilde{\boldsymbol{h}}^c\boldsymbol{R} + \tilde{\boldsymbol{p}}\boldsymbol{R}^T m^c\mathbf{1}_3\boldsymbol{R}\right)^T & \boldsymbol{R}^T m^c\mathbf{1}_3\boldsymbol{R} \end{array}\right], \tag{A.35}$$

where $\boldsymbol{R}$ and $\boldsymbol{p}$ are $\boldsymbol{R}(z,d,\theta)$ and $\boldsymbol{p}_z$, respectively, and defined earlier in this appendix. The (1,1), (1,2), and (2,2) blocks correspond to the moment of inertia, first moment of mass, and mass of the composite rigid body expressed at a new coordinate system and can likewise be completely specified with ten quantities. Each of these blocks is examined individually.

The (2,2) block corresponds to the mass of the composite rigid body and can be simplified to a scalar as follows:

$$m^z = m^c, \tag{A.36}$$

which requires no computation.

The (1,2) block corresponds to the first moment of mass for the composite rigid body. It can be simplified as follows:

$$\tilde{\boldsymbol{h}}^z = \boldsymbol{R}^T(z,\theta)\tilde{\boldsymbol{h}}^c\boldsymbol{R}(z,\theta) + m^c\tilde{\boldsymbol{p}}_z, \tag{A.37}$$

This equation can be expressed in vector form as follows:

$$\boldsymbol{h}^z = \boldsymbol{R}^T(z,\theta)\boldsymbol{h}^c + m^c\boldsymbol{p}_z, \tag{A.38}$$

where the following equivalence theorem was applied to the transformation of a skew symmetric matrix:

$$\tilde{a} = \boldsymbol{R}^T \tilde{b} \boldsymbol{R} \;\rightarrow\; \boldsymbol{a} = \boldsymbol{R}^T \boldsymbol{b}. \tag{A.39}$$

The quantity, $m^c \, \boldsymbol{p}_z$, has one non-zero element which requires [1M, 0A] to compute, but for revolute joints, it is constant and can be computed off-line. From previous results, [4M, 2A] are required to compute the planar rotation of a Cartesian vector, $\boldsymbol{R}^T(z,\theta)\boldsymbol{h}^c$. The result is obtained by adding these together which only requires [0M, 1A]. For revolute joints, the total cost is [4M, 3A] and for prismatic joints it is [5M, 3A].

The (1,1) block corresponds to the composite rigid body's moment of inertia, and can be simplified to the following:

$$\begin{aligned} \bar{\boldsymbol{K}}^z \;=\; & \boldsymbol{R}^T(z,\theta)\bar{\boldsymbol{K}}^c\boldsymbol{R}(z,\theta) + \boldsymbol{R}^T(z,\theta)\tilde{\boldsymbol{h}}^c\boldsymbol{R}(z,\theta)\tilde{\boldsymbol{p}}_z^T + \\ & \left[\boldsymbol{R}^T(z,\theta)\tilde{\boldsymbol{h}}^c\boldsymbol{R}(z,\theta)\tilde{\boldsymbol{p}}_z^T\right]^T + m^c\tilde{\boldsymbol{p}}_z\tilde{\boldsymbol{p}}_z^T. \end{aligned} \tag{A.40}$$

This can be further simplified to the following equation:

$$\begin{aligned} \bar{\boldsymbol{K}}^z \;=\; & \boldsymbol{R}^T(z,\theta)\bar{\boldsymbol{K}}^c\boldsymbol{R}(z,\theta) + \left[\boldsymbol{R}^T(\tilde{z},\theta)\boldsymbol{h}^c\right]\tilde{\boldsymbol{p}}_z^T + \\ & \left\{\left[\boldsymbol{R}^T(\tilde{z},\theta)\boldsymbol{h}^c\right]\tilde{\boldsymbol{p}}_z^T\right\}^T + m^c\tilde{\boldsymbol{p}}_z\tilde{\boldsymbol{p}}_z^T, \end{aligned} \tag{A.41}$$

where $\left[\boldsymbol{R}^T(\tilde{z},\theta)\boldsymbol{h}^c\right]$ is the screw symmetric matrix representation of the vector, $\boldsymbol{R}^T(z,\theta)\boldsymbol{h}^c$, which was computed as part of the previous step.

The computational cost of $\boldsymbol{R}^T(z,\theta)\bar{\boldsymbol{K}}^c\boldsymbol{R}(z,\theta)$, the planar congruence transformation of a symmetric 3×3 matrix, is [10M, 10A] when the joint is revolute and

Table 24: Computation requirements for congruence transformation of a spatial (or CRB) inertia matrix.

	Revolute		Prismatic	
Computation	×	+	×	+
m^z	—	—	—	—
$\boldsymbol{h}^z$	4	3	5	3
$\bar{\boldsymbol{K}}^z$	13	18	12	15
m^x	—	—	—	—
$\boldsymbol{h}^x$	4	3	4	3
$\bar{\boldsymbol{K}}^x$	11	14	11	14
Total:	32	39	32	36

θ is variable, or [8M, 7A] when the joint is prismatic. The multiplication of the two skew symmetric matrices, $\left[\boldsymbol{R}^T(\tilde{z},\theta)\boldsymbol{h}^c\right]\tilde{\boldsymbol{p}}_z^T$ only requires [4M, 0A] because only one element of $\tilde{\boldsymbol{p}}_z$ is nonzero. The quantity, $m^c\tilde{\boldsymbol{p}}_z\tilde{\boldsymbol{p}}_z^T$, is constant when the joint is revolute and can be computed off-line; otherwise, it requires [1M, 0A]. Adding these terms together to obtain the final result only requires [0M, 8A] because the intermediate results contain many zero elements. The total computational cost is [13M, 18A] for the case of a revolute joint, and [12M, 15A] for a prismatic joint.

The same steps must be carried out for the x-planar congruence transform as well. For this step, α and $\boldsymbol{p}_x$ are both constant which results in the same amount of computation for either type of joint, and which is less than either of the results for the z-planar congruence transform. The cost for all of the steps is listed in Table 24. For a revolute joint, the transformation of the spatial inertia matrix costs [32M, 39A] and for a prismatic joint it only requires [32M, 36A]

A.5 Summary and Conclusions

In this appendix, the transformation operations used extensively in the multibody dynamics algorithms have been examined. Parameters to specify these transformations are obtained with the use of modified Denavit-Hartenberg (MDH) notation which is presented first. Use of these parameters are shown in this appendix to yield particularly efficient transformation functions that are used extensively throughout this dissertation. The key to efficient transformations is to decompose them into a series of planar transformations. With this approach, the transformation matrices, $\boldsymbol{R}$ and $\mathbf{X}$, and position vector, $\boldsymbol{p}$, are always represented by a set of scalar parameters and are never explicitly computed. This reduces the amount of computation as well as computer storage requirements. In addition, the set of successive planar operations on the vectors and matrices require fewer operations than the corresponding matrix-vector and matrix-matrix operations if $\boldsymbol{R}$ and $\mathbf{X}$ were computed first.

Appendix B

Efficient Single, Open Chain Robotics Algorithms

B.1 Introduction

In Part I, algorithms to compute the dynamics of multiple manipulator systems grasping a common object and multilegged vehicles are presented. Two components of both algorithms are the same and account for a majority of the computation. They are the inverse dynamics and joint space inertia matrix computations for single, open chain mechanisms. Efficient recursive algorithms for these computations first appeared over a decade ago, and were used in the Composite Rigid Body (CRB) approach to simulating robotic mechanisms [14]. Since that time, significant improvements have been made to these algorithms [16, 29]. The algorithms in Part II, are based on the Articulated Body (AB) algorithm first developed by Featherstone [15], and significant improvements to the efficiency of this algorithm were reported by Brandl, Johanni, and Otter [17]. Algorithms for the inverse dynamics, joint space inertia matrix, and AB dynamics of a single serial chain with a fixed base are presented in this appendix using spatial notation wherever appropriate. Computational efficiency is comparable to, and sometimes better than the best results reported in the literature.

B.2 Inverse Dynamics

The inverse dynamics computation is used to determine joint forces and torques that are required to bring about a desired motion that is specified by joint positions, velocities and accelerations. For a single serial chain with revolute or prismatic joints and a fixed base, the algorithm is listed in Table 25 using spatial notation wherever possible. In this form, it is used extensively in dynamic control algorithms to compute a feedforward term. In a slightly modified form, with accelerations set to zero, it is also used to compute the bias terms needed in the Composite Rigid Body simulation algorithm for a single chain [14]. In this form, it is used in the computation of open-chain accelerations for the multiple manipulator simulation in Chapter 2, and in an extended form, this computation is used to compute the analogous bias terms for a multilegged vehicle in Chapter 5. This section outlines the procedure for computing the inverse dynamics algorithm in Table 25 very efficiently, and presents the computational requirements for this and the corresponding computation for the bias terms.

The algorithm consists of two recursions. The first, from the base of the chain to the end-effector, is often referred to as the Forward Kinematics recursion and computes the angular velocity, ω_i, and spatial acceleration, $\mathbf{a}_i$, of each link, as well as the gravity- and velocity-dependent bias force, $\mathbf{f}'_i$, exerted on each chain. The force due to gravity is included in this term by accelerating the base "upward" by the gravitational acceleration and its force on each link is efficiently included in the acceleration vector of the bias force equation. The second recursion is called

Table 25: Inverse dynamics algorithm for a single, serial chain with a fixed base.

Given: $\omega_0 = \mathbf{0}$, $\mathbf{a}_0 = [\mathbf{0}^T - \mathbf{g}^T]^T$, ${}^iR_{i-1}$, ${}^{i-1}p_i$, $\dot{q}_i$, $\ddot{q}_i$, ${}^N\mathbf{f}_{N+1}$ (external tip force).

for $i = 1, \ldots, N$.

$$\omega_i = {}^iR_{i-1}\omega_{i-1} + \sigma_i\dot{q}_i\hat{\mathbf{z}}_i \tag{B.1}$$

$$\mathbf{a}_i = {}^i\mathbf{X}_{i-1}\mathbf{a}_{i-1} + \ddot{q}_i\boldsymbol{\phi}_i + \begin{bmatrix} \mathbf{0} \\ {}^iR_{i-1}[\omega_{i-1} \times (\omega_{i-1} \times {}^{i-1}p_i)] \end{bmatrix} + \begin{bmatrix} \sigma_i(\omega_i \times \dot{q}_i\hat{\mathbf{z}}_i) \\ \bar{\sigma}_i(2\omega_i \times \dot{q}_i\hat{\mathbf{z}}_i) \end{bmatrix} \tag{B.2}$$

$$\mathbf{f}_i' = \mathbf{I}_i\mathbf{a}_i + \begin{bmatrix} \omega_i \times \bar{I}_i\omega_i \\ \omega_i \times (\omega_i \times h_i) \end{bmatrix} \tag{B.3}$$

end for i.

for $i = N, \ldots, 1$.

$$\mathbf{f}_i = \mathbf{f}_i' + {}^{i+1}\mathbf{X}_i^T\mathbf{f}_{i+1} \tag{B.4}$$

$$\tau_i = \boldsymbol{\phi}_i^T\mathbf{f}_i \tag{B.5}$$

end for i.

---

the Backward Dynamics recursion which computes the force exerted on each link to achieve the specified motion and projects this onto the joint motion axis, $\boldsymbol{\phi}_i$, to determine the desired joint torque or force, τ_i.

Using the MDH parameters and the results presented in Appendix A for efficient transformations of vectors, the rotation of a Cartesian vector, as in the computation of ω_i costs [8M, 4A]. In addition, the transformation of a spatial vector in the computation of the spatial accelerations, $\mathbf{a}_i$, costs [20M, 12A] with the procedure presented in Appendix A. Closer examination of the computations for the spatial

accelerations, $\mathbf{a}_i$, results in a different, more efficient procedure. To begin, the spatial acceleration can be written as two equations for angular and translational acceleration as follows:

$$\dot{\boldsymbol{\omega}}_i = {}^i\boldsymbol{R}_{i-1}\dot{\boldsymbol{\omega}}_{i-1} + \sigma_i\ddot{q}_i\hat{\boldsymbol{z}}_i + \sigma_i\left(\boldsymbol{\omega}_i \times \dot{q}_i\hat{\boldsymbol{z}}_i\right), \tag{B.6}$$

$$\begin{aligned}\boldsymbol{a}_i = {}& {}^i\boldsymbol{R}_{i-1}\boldsymbol{a}_{i-1} + \bar{\sigma}_i\ddot{q}_i\hat{\boldsymbol{z}}_i + {}^i\boldsymbol{R}_{i-1}\left(\dot{\boldsymbol{\omega}}_{i-1} \times {}^{i-1}\boldsymbol{p}_i\right) \\ & + {}^i\boldsymbol{R}_{i-1}\left[\boldsymbol{\omega}_{i-1} \times \left(\boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{p}_i\right)\right] + \bar{\sigma}_i\left(2\boldsymbol{\omega}_i \times \dot{q}_i\hat{\boldsymbol{z}}_i\right).\end{aligned} \tag{B.7}$$

Balafoutis, Patel, and Misra [29] showed that the latter can be made much more efficient by combining the terms multiplied by the rotation matrix ${}^i\boldsymbol{R}_{i-1}$ and then the terms performing the cross-product with ${}^{i-1}\boldsymbol{p}_i$. The resulting equation is given as follows:

$$\boldsymbol{a}_i = {}^i\boldsymbol{R}_{i-1}\left(\boldsymbol{a}_{i-1} + \boldsymbol{\Omega}_{i-1}{}^{i-1}\boldsymbol{p}_i\right) + \bar{\sigma}_i\ddot{q}_i\hat{\boldsymbol{z}}_i + \bar{\sigma}_i\left(2\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}\right), \tag{B.8}$$

where $\boldsymbol{\Omega}_i$ is defined as follows:

$$\boldsymbol{\Omega}_i = \tilde{\dot{\boldsymbol{\omega}}}_i + \tilde{\boldsymbol{\omega}}_i\tilde{\boldsymbol{\omega}}_i. \tag{B.9}$$

Therefore Eqs. (B.6), (B.8), and (B.9) replace Eq. (B.2). This reduces the amount of computation for the spatial acceleration from [42M, 27A] to [33M, 28A] for a link with a revolute joint.

Using $\boldsymbol{\Omega}_i$, a substantial reduction in the amount of computation for the bias force in Eq. (B.3) can also be obtained. This equation is also divided into two Cartesian equations for moment and linear force. The moment is computed as follows:

$$\boldsymbol{n}_i' = \bar{\boldsymbol{I}}_i\dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times \bar{\boldsymbol{I}}_i\boldsymbol{\omega}_i + \boldsymbol{h}_i \times \boldsymbol{a}_i. \tag{B.10}$$

He and Goldenberg [71] showed that the first two terms can be substituted with a much more efficient expression which is a function of $\boldsymbol{\Omega}_i$ and $\bar{\boldsymbol{I}}_i$. This results in the following equation for the moment:

$$\boldsymbol{n}_i' = \boldsymbol{u}_i + \boldsymbol{h}_i \times \boldsymbol{a}_i, \tag{B.11}$$

where $\boldsymbol{u}_i$ is defined as:

$$\boldsymbol{u}_i = \begin{bmatrix} \bar{I}_{xz}\Omega_{yx} - \bar{I}_{xy}\Omega_{zx} + \bar{I}_{yz}(\Omega_{yy} - \Omega_{zz}) + \hat{I}_z\Omega_{yz} - \hat{I}_y\Omega_{zy} \\ \bar{I}_{xy}\Omega_{zy} - \bar{I}_{yz}\Omega_{xy} + \bar{I}_{xz}(\Omega_{zz} - \Omega_{xx}) + \hat{I}_x\Omega_{zx} - \hat{I}_z\Omega_{xz} \\ \bar{I}_{yz}\Omega_{xz} - \bar{I}_{xz}\Omega_{yz} + \bar{I}_{xy}(\Omega_{xx} - \Omega_{yy}) + \hat{I}_y\Omega_{xy} - \hat{I}_x\Omega_{yx} \end{bmatrix}, \tag{B.12}$$

and the elements of the 3×3 matrices, $\boldsymbol{\Omega}_i$ and $\bar{\boldsymbol{I}}_i$, have been specified in the following way:

$$\boldsymbol{A}_i = \begin{bmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{bmatrix}, \tag{B.13}$$

The $\hat{I}$ terms are constant functions of the elements of $\bar{\mathbf{I}}_i$ and can be obtained with the following off-line computation:

$$\hat{I}_x = \bar{I}_{xx} - \frac{1}{2}\left(\bar{I}_{xx} + \bar{I}_{yy} + \bar{I}_{zz}\right) \tag{B.14}$$

$$\hat{I}_y = \bar{I}_{yy} - \frac{1}{2}\left(\bar{I}_{xx} + \bar{I}_{yy} + \bar{I}_{zz}\right) \tag{B.15}$$

$$\hat{I}_z = \bar{I}_{zz} - \frac{1}{2}\left(\bar{I}_{xx} + \bar{I}_{yy} + \bar{I}_{zz}\right). \tag{B.16}$$

Finally, the linear force component of Eq. (B.3) is written as follows:

$$\boldsymbol{f}_i' = m_i\boldsymbol{a}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{h}_i + \boldsymbol{\omega}_i \times [\boldsymbol{\omega}_i \times \boldsymbol{h}_i], \tag{B.17}$$

where the last two terms may be combined, much like in the translational acceleration equation, and the resulting equation is given by:

$$\boldsymbol{f}_i' = m_i\boldsymbol{a}_i + \boldsymbol{\Omega}_i\boldsymbol{h}_i. \tag{B.18}$$

Table 26: Computation requirements for the inverse dynamics algorithm.

Computation		×	+
$\boldsymbol{\omega}_i$		$8N-12$	$5N-9$
$\mathbf{a}_i$:	$\dot{\boldsymbol{\omega}}_i$	$10N-14$	$7N-11$
	$\boldsymbol{a}_i$	$17N-14$	$13N-14$
	$\boldsymbol{\Omega}_i$	$6N-5$	$9N-9$
$\mathbf{f}'_i$:	$\boldsymbol{u}_i$	$15N-14$	$15N-14$
	$\boldsymbol{n}'_i$	$6N-4$	$6N-4$
	$\boldsymbol{f}'_i$	$12N-12$	$9N-9$
$\mathbf{f}_i$		$20N-30$	$18N-22$
τ_i		—	—
Total:		$94N-105$	$82N-92$

Noting that $\boldsymbol{\Omega}_i$ has already been computed with the acceleration terms, the cost of computing Eqs. (B.11), (B.12), and (B.18) is [33M, 30A] which is more efficient than the straightforward implementation of Eq. (B.3) which would cost [51M, 39A].

The computation of the equations of the Backward Dynamics recursion are comparatively straightforward. The computation of the spatial force involves a spatial transformation, as presented in the previous appendix, and a spatial vector addition. The projection of a vector onto the joint motion axis requires no computation.

The computational requirements for each of the resulting equations has been broken down in Table 26 for a serial chain with revolute joints having two or more links. With a fixed base, a significant savings can be obtained in the first steps of the Forward Kinematics recursion by taking into account the fact that $\boldsymbol{\omega}_0$ and $\dot{\boldsymbol{\omega}}_0$ are both zero and as a result, $\boldsymbol{\omega}_1$ and $\dot{\boldsymbol{\omega}}_1$ also contain two zero elements. This

reduces the amount of computation that must be done in the first two iterations of this recursion. In the Backward Dynamics recursion, it is also assumed that the tip force is given with respect to link N's coordinate system so that the first spatial transformation in Eq. (B.4) is not necessary. Finally, only the third element of $\mathbf{f}_1$ and, hence, $\mathbf{f}_1'$, is needed in the last iteration of this recursion.

For the inverse dynamics algorithm [$(94N - 105)$M, $(82N - 92)$A] are required. This is comparable to the most efficient result reported by Balafoutis, Patel, and Misra in [29] with a computational requirement of [$(93N - 69)$M, $(81N - 66)$A]. For the bias computation in the CRB method, the same algorithm is used with all of the joint accelerations set to zero. This leads to one fewer addition in the computation of all the $\dot{\boldsymbol{\omega}}_i$'s and an additional zero element in $\dot{\boldsymbol{\omega}}_1$. The resulting computational complexity for this case is [$(94N - 114)$M, $(81N - 105)$A].

B.3 Joint-Space Inertia Matrix

One of the most concise expressions of algorithms for computing the joint space inertia matrix for a serial chain is given by Lilly and Orin in [25] using spatial notation. The most efficient of their algorithms is called the Spatial CRB Method (Method IV) and is shown in Table 27. An alternate implementation of this algorithm was also developed by Balafoutis and Patel [16] that used the concepts of generalized and augmented bodies to obtain significantly more efficient results. In this section, the results of Appendix A are used to develop an implementation that is equally efficient and much more straightforward. Table 28 summarizes the computational cost of the algorithm in Table 27 for a serial chain with revolute joints.

Table 27: Joint-space inertia matrix algorithm for a single, serial chain.

Given: $\mathbf{K}_{N+1} = \mathbf{0}$ and ${}^{i+1}\mathbf{X}_i$ for all $i = 1, \ldots, N$.

for $i = N, \ldots, 1$.

$$\mathbf{K}_i = \mathbf{I}_i + {}^{i+1}\mathbf{X}_i^T \mathbf{K}_{i+1} {}^{i+1}\mathbf{X}_i \tag{B.19}$$

$${}^{i}\mathbf{f}_i = \mathbf{K}_i \boldsymbol{\phi}_i \tag{B.20}$$

$$[\mathbf{H}]_{i,i} = \boldsymbol{\phi}_i^T {}^{i}\mathbf{f}_i \tag{B.21}$$

for $j = i-1, \ldots, 1$.

$${}^{j}\mathbf{f}_i = {}^{j+1}\mathbf{X}_j^T {}^{j+1}\mathbf{f}_i \tag{B.22}$$

$$[\mathbf{H}]_{i,j} = \boldsymbol{\phi}_j^T {}^{j}\mathbf{f}_i \tag{B.23}$$

end for j.

end for i.

The most expensive step in this algorithm is the computation of $\mathbf{K}_i$ in Eq. (B.19), which is the CRB inertia of links i through N with the joints frozen in the position specified by the state input. In the previous appendix, the congruence transformation in this equation was shown to require [32M, 39A] when joint $i+1$ is revolute (less when it is prismatic). In this procedure, the moment of inertia matrix, first moment of mass vector, and scalar mass were computed individually rather than in a form specified by the structure of the spatial inertia matrix. The sum in Eq. (B.19) is most efficiently accomplished in the same manner. First, this equation is partitioned into its 3×3 Cartesian matrix equations as follows:

$$\begin{bmatrix} \bar{\boldsymbol{K}}_i^c & \tilde{\boldsymbol{h}}_i^c \\ \left(\tilde{\boldsymbol{h}}_i^c\right)^T & m_i^c \mathbf{1}_3 \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{I}}_i & \tilde{\boldsymbol{h}}_i \\ \tilde{\boldsymbol{h}}_i^T & m_i \mathbf{1}_3 \end{bmatrix} + \begin{bmatrix} {}^{i}\bar{\boldsymbol{K}}_{i+1}^c & {}^{i}\tilde{\boldsymbol{h}}_{i+1}^c \\ \left({}^{i}\tilde{\boldsymbol{h}}_{i+1}^c\right)^T & m_{i+1}^c \mathbf{1}_3 \end{bmatrix}, \tag{B.24}$$

Table 28: Computation requirements for the joint space inertia matrix algorithm.

Computation	×	+
$\mathbf{K}_i$	$32N - 41$	$48N - 61$
${}^i\mathbf{f}_i$	—	—
$[\mathbf{H}]_{i,i}$	—	—
${}^j\mathbf{f}_i$	$10N^2 - 23N + 16$	$6N^2 - 14N + 11$
$[\mathbf{H}]_{i,j}$	—	—
Total:	$10N^2 + 9N - 25$	$6N^2 + 34N - 50$
$(N = 6)$	(389)	(370)

where

$$\bar{\boldsymbol{K}}_i^c = \bar{\boldsymbol{I}}_i + {}^i\bar{\boldsymbol{K}}_{i+1}^c, \tag{B.25}$$

$$\boldsymbol{h}_i^c = \boldsymbol{h}_i + {}^i\boldsymbol{h}_{i+1}^c, \tag{B.26}$$

$$m_i^c = m_i + m_{i+1}^c. \tag{B.27}$$

These three equations specify the mass properties (moment of inertia, first moment of mass, and mass, respectively) of the composite rigid body consisting of links i through N with the intervening joints locked, and they are expressed with respect to link i's coordinate system. The first is symmetric and only requires [0M, 6A], the second is a Cartesian vector which requires [0M, 3A], and the third is constant and can be computed off-line. For a revolute joint, this step requires a total of [32M, 48A]. Some additional savings is achieved for this step by taking into account that $\mathbf{K}_{N+1}$ is zero, and only the (3,3) element of $\mathbf{K}_1$ needs to be computed.

Since the spatial joint motion axis, $\boldsymbol{\phi}_i$, contains zeros and a one, no computation is required to compute ${}^i\mathbf{f}_i$, $[\mathbf{H}]_{i,i}$, and $[\mathbf{H}]_{i,j}$. The only other step in this algorithm

that requires any arithmetic operations is the spatial transformation of ${}^{j+1}\mathbf{f}_i$. This step is executed $N(N-1)/2$ times and the computational cost of this spatial transform is [20M, 12A]. Additional savings is achieved in two ways for this step. First, only the third element of this vector is needed when $j = 1$ which requires only [10M, 7A] to compute, and occurs $N-1$ times during the last iteration of the j loop. Second, the last element of ${}^{i}\mathbf{f}_i$ is zero when link i is revolute and the cost of a spatial transformation in this instance is [17M, 9A]. This occurs $N-2$ times, during the first iteration every time the j loop is entered and j does not equal 1.

For a serial chain containing six revolute joints the cost of computing the joint space inertia matrix is [389M, 370A]. This is much more efficient than the results reported by Lilly and Orin [25] which are [555M, 470A]. They are comparable to Balafoutis and Patel's [16] which require [420M, 339A]. Their results were obtained with a much more complex decomposition of Eq. (B.19) which mixed steps involved in transforming the CRB inertia with the subsequent addition. By doing this, quantities related to augmented and generalized links could be identified and the former could be computed off-line in the case of revolute links. The method presented here is much more modular, because the transformation can be computed separately from the sum, and comparatively little computation is dependent on the joint type.

B.4 Articulated Body Dynamics

The AB dynamics algorithm for a single serial chain as derived in Chapter 7 contains three $O(N)$ recursions. The first is the Forward Kinematics recursion which computes the velocities and velocity-dependent terms, $\boldsymbol{\beta}_i$ and $\boldsymbol{\zeta}_i$, of each link from

the base of the chain to the end-effector. Then, the articulated-body inertias, $\mathbf{I}_i^*$, and bias forces, $\boldsymbol{\beta}_i^*$, are computed in the Backward Dynamics recursion from the end-effector to the base. The final Forward Accelerations recursion computes each joint and link acceleration, in that order, from the base to the end-effector. The resulting algorithm is listed in Table 29.

In this algorithm, the equations for $\mathbf{n}_i$, m_i^*, $\mathbf{N}_i$, and τ_i^*, that were defined in Chapter 7 represent intermediate computations that are also used elsewhere in the algorithm. Wherever possible, these terms, along with $(m_i^*)^{-1}$ and $\mathbf{n}_i\,(m_i^*)^{-1}$, have been substituted into the equations to minimize the amount of computation in the final algorithm.

The computational requirements of this algorithm are listed in Table 30 for a chain containing revolute joints. The requirements for the Forward Kinematics recursion are listed in the top section of the table. Computation of the links' angular velocities is the same as in the inverse dynamics algorithm. Because the base of the chain is fixed, $\boldsymbol{\omega}_i$ is zero and [12M, 9A] can be eliminated in the first two iterations of this equation by accounting for the zero elements. Computational requirements for $\boldsymbol{\zeta}_i$ and $\boldsymbol{\beta}_i$ are straightforward and can also be reduced by taking these zero elements into account.

A number of steps can be taken to reduce the amount of computation in the Backward Dynamics recursion. First, $\mathbf{n}_N$, m_N^*, and $\mathbf{N}_N$ are constant and can be computed off-line along with $(m_N^*)^{-1}$ and $\mathbf{n}_N(m_N^*)^{-1}$. For revolute joints, subsequent iterations to compute $\mathbf{n}_i$ simplifies to selecting the third column of $\mathbf{I}_i^*$, $(m_i^*)^{-1}$ reduces

Table 29: Articulated Body dynamics algorithm for a single serial chain.

I. Forward Kinematics: Given: $\omega_0 = \mathbf{0}$.
for $i = 1, \ldots, N$. Given: ${}^{i}\boldsymbol{R}_{i-1}$, ${}^{i-1}\boldsymbol{p}_i$, $\dot{q}_i$.

$$\omega_i = {}^{i}\boldsymbol{R}_{i-1}\omega_{i-1} + \sigma_i \dot{q}_i \hat{\mathbf{z}} \tag{B.28}$$

$$\zeta_i = \begin{bmatrix} \mathbf{0} \\ {}^{i}\boldsymbol{R}_{i-1}\left[\omega_{i-1} \times (\omega_{i-1} \times {}^{i-1}\boldsymbol{p}_i)\right] \end{bmatrix} + \begin{bmatrix} \sigma_i(\omega_i \times \dot{q}_i\hat{\mathbf{z}}_i) \\ \bar{\sigma}_i(2\omega_i \times \dot{q}_i\hat{\mathbf{z}}_i) \end{bmatrix} \tag{B.29}$$

$$\boldsymbol{\beta}_i = -\begin{bmatrix} \omega_i \times \bar{\boldsymbol{I}}_i\omega_i \\ \omega_i \times (\omega_i \times \boldsymbol{h}_i) \end{bmatrix} \tag{B.30}$$

end for i.

II. Backward Dynamics: Given: $\mathbf{I}_N^* = \mathbf{I}_N$, $\boldsymbol{\beta}_N^* = \boldsymbol{\beta}_N - {}^{N+1}\mathbf{X}_N^T\mathbf{f}_{N+1}$ (tip force).
for $i = N, \ldots, 1$. Given: τ_i.

$$\mathbf{n}_i = \mathbf{I}_i^*\boldsymbol{\phi}_i \tag{B.31}$$

$$m_i^* = \boldsymbol{\phi}_i^T\mathbf{I}_i^*\boldsymbol{\phi}_i \tag{B.32}$$

$$\mathbf{N}_i = \mathbf{I}_i^* - \mathbf{n}_i(m_i^*)^{-1}\mathbf{n}_i^T \tag{B.33}$$

$$\tau_i^* = \tau_i + \boldsymbol{\phi}_i^T\boldsymbol{\beta}_i^* \tag{B.34}$$

$$\mathbf{I}_{i-1}^* = \mathbf{I}_{i-1} + {}^{i}\mathbf{X}_{i-1}^T\mathbf{N}_i{}^{i}\mathbf{X}_{i-1} \tag{B.35}$$

$$\boldsymbol{\beta}_{i-1}^* = \boldsymbol{\beta}_{i-1} + {}^{i}\mathbf{X}_{i-1}^T\left[\boldsymbol{\beta}_i^* - \mathbf{N}_i\zeta_i - \mathbf{n}_i(m_i^*)^{-1}\tau_i^*\right] \tag{B.36}$$

end for i.

III. Forward Accelerations: Given: $\mathbf{a}_0' = [0^T \quad -{}^{0}\boldsymbol{a}_g^T]^T$.
for $i = 1, \ldots, N$.

$$\bar{\mathbf{a}}_i = {}^{i}\mathbf{X}_{i-1}\mathbf{a}_{i-1}' + \zeta_i \tag{B.37}$$

$$\ddot{q}_i = (m_i^*)^{-1}\tau_i^* - \left[\mathbf{n}_i(m_i^*)^{-1}\right]^T\bar{\mathbf{a}}_i \tag{B.38}$$

$$\mathbf{a}_i' = \bar{\mathbf{a}}_i + \boldsymbol{\phi}_i\ddot{q}_i \tag{B.39}$$

end for i.

Table 30: Computational requirements of the AB dynamics algorithm for a single serial chain with revolute joints.

	×	+
$\boldsymbol{\omega}_i$	$8N-12$	$5N-9$
$\boldsymbol{\zeta}_i$	$22N-33$	$10N-18$
$\boldsymbol{\beta}_i$	$27N-19$	$15N-15$
$\mathbf{n}_i$	—	—
$(m_i^*)^{-1}$	$N-1$	—
$\mathbf{n}_i\,(m_i^*)^{-1}$	$5N-5$	—
$\mathbf{N}_i$	$15N-30$	$15N-30$
τ_i^*	—	N
$\mathbf{I}_{i-1}^*$	$70N-80$	$86N-96$
$\boldsymbol{\beta}_{i-1}^*$	$50N-54$	$49N-49$
$\mathbf{a}_i'$	$20N-18$	$17N-19$
$\ddot{q}_i$	$6N-2$	$6N-3$
$\mathbf{a}_i$	—	$N-2$
Total:	$224N-254$	$205N-241$
$(N=6)$	(1090)	(989)

to a scalar reciprocal, and the third element of $\mathbf{n}_i(m_i^*)^{-1}$ is one. The matrix, $\mathbf{N}_i$, can be thought of as the ith AB inertia after it has been reflected across joint i. Because this joint is free to move, it contains a zero row and column:

$$\mathbf{N}_i = \begin{bmatrix} n_{11} & n_{12} & 0 & n_{14} & n_{15} & n_{16} \\ n_{12} & n_{22} & 0 & n_{24} & n_{25} & n_{26} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ n_{14} & n_{24} & 0 & n_{44} & n_{45} & n_{46} \\ n_{15} & n_{25} & 0 & n_{45} & n_{55} & n_{56} \\ n_{16} & n_{26} & 0 & n_{46} & n_{56} & n_{66} \end{bmatrix}. \tag{B.40}$$

This reflected inertia matrix is also symmetric (note the indices), and only 15 of its elements must be computed which requires [1M, 1A] each. The most costly step in this recursion is the spatial congruence transformation of $\mathbf{N}_i$ in the computation of

the AB inertia, $\mathbf{I}^*_{i-1}$. A careful analysis in Appendix A shows that this is computed most efficiently with [70M, 71A]. To complete the computation of $\mathbf{I}^*_{i-1}$, the addition this result with a spatial rigid body inertia must be performed which requires another 15 additions for a total of [70M, 86A]. Note that [10M, 10A] can be saved in the congruence transformation of $\mathbf{N}_N$ because n_{46} and n_{56} are also zero. Additional savings in this recursion also result because $\mathbf{N}_1$, $\mathbf{I}^*_0$, and $\boldsymbol{\beta}^*_0$ are not needed.

As with the inverse dynamics algorithm, gravitational effects are most efficiently incorporated by biasing the fixed base's acceleration by the negative of gravity, $-{}^0\boldsymbol{a}_g$. Since the base's coordinate system is fixed within the inertia frame, this term can be computed off-line. The resulting spatial acceleration, $\mathbf{a}_0$, has three zero elements which reduces the amount of computation required in the first few iterations of the Forward Accelerations recursion. The computational requirements for the rest of this recursion is straightforward. However, note that the third element of $\boldsymbol{\zeta}_i$ is always zero for revolute joints and the computation of $\mathbf{a}_N$ is not required.

The resulting computational cost of the algorithm for a single serial chain with N revolute links is $[(224N-254)\text{M}, (205N-247)\text{A}]$. For $N = 6$ this corresponds to [1090M, 989A]. Previously, the most efficient algorithm was developed by Brandl, Johanni, and Otter [17] which has a computational requirement of $[(250N-222)\text{M}, (220N-198)\text{A}]$ in general, and [1278M, 1122A] for $N = 6$. In comparison, the improved algorithm presented in this section requires about 15% less computation. Most of this improvement is directly attributable to the new efficient transformation of the reflected AB inertia developed in Appendix A.

REFERENCES

[1] M. Iwasaki, J. Akizono, H. Takahashi, T. Umetani, T. Nemoto, O. Asakura, and K. Asayama, "Development on Aquatic Walking Robot for Underwater Inspection," *Report of the Port and Harbour Research Institute*, vol. 26, pp. 393–422, December 1987.

[2] *Proceedings of the 20th Annual Technical Symposium and Exhibition of the Association for Unmanned Vehicle Systems.* Washington, D.C.: Association for Unmanned Vehicle Systems, June 1993.

[3] M. J. Zyda, R. B. McGhee, S. Kwak, D. B. Nordman, R. C. Rogers, and D. Marco, "Three-Dimensional Visualization of Mission Planning and Control for the NPS Autonomous Underwater Vehicle," *IEEE Journal of Oceanic Engineering*, vol. 15, no. 3, pp. 217–221, 1990.

[4] S.-Y. Oh and D. E. Orin, "Dynamic Computer Simulation of Multiple Closed-Chain Robotic Mechanisms," in *IEEE Int. Conf. on Robotics and Automation*, (San Francisco, CA), pp. 15–20, April 1986.

[5] L. Shih, A. A. Frank, and B. Ravani, "Dynamic Simulation of Legged Machines Using a Compliant Joint Model," *Int. Journal of Robotics Research,* The MIT Press, vol. 6, pp. 33–46, Winter 1987.

[6] K. W. Lilly and D. E. Orin, "Efficient Dynamic Simulation for Multiple Chain Robotic Mechanisms," in *Proc. 3rd Annual Conf. on Aerospace Computational Control* (D. E. Bernard and G. K. Man, ed.), vol. 1, (Pasadena, CA), pp. 73–87, December 1989.

[7] P. S. Freeman and D. E. Orin, "Efficient Dynamic Simulation of a Quadruped Using a Decoupled Tree-Structured Approach," *Int. Journal of Robotics Research,* The MIT Press, vol. 10, pp. 619–627, December 1991.

[8] D. P. Brutzman, Y. Kanayama, and M. J. Zyda, "Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles," in *Proc. of IEEE Symposium on Autonomous Underwater Vehicle Technology,* (Washington, D.C.), pp. 3–10, June 1992.

[9] L. Conway, R. Volz, and M. Walker, "Tele-Autonomous Systems: Methods and Architectures for Intermingling Autonomous and Telerobotic Technology," in *Proc. of 1987 IEEE Int. Conf. on Robotics and Automation*, (Raleigh, NC), pp. 1121–1130, 1987.

[10] J. Funda and R. P. Paul, "Teleprogramming: Overcoming Communication Delays in Remote Manipulation," in *Proc. of 1st IARP Workshop on Mobile Robots for Subsea Environments*, (Monterey, CA), pp. 155–162, October 1990.

[11] D. E. Orin, "Dynamical Modelling of Coordinated Multiple Robot Systems," in *Report of Workshop on Coordinated Multiple Robot Manipulators* (A. J. Koivo and G. A. Bekey, eds.), (San Diego, CA), Jan. 1987.

[12] R. B. McGhee, D. E. Orin, D. R. Pugh, and M. R. Patterson, "A Hierarchically-Structured System for Computer Control of a Hexapod Walking Machine," in *Proc. of Symposium on Theory and Practice of Robots and Manipulators* (A. Morecki *et al.*, ed.), (London), pp. 375–381, Hermes Publishing Co., 1985.

[13] D. E. Orin and S. Y. Oh, "Control of Force Distribution in Robotic Mechanisms Containing Closed Kinematic Loops," *Transactions of the ASME*, vol. 102, pp. 134–141, 1981.

[14] M. W. Walker and D. E. Orin, "Efficient Dynamic Computer Simulation of Robotic Mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, pp. 205–211, September 1982.

[15] R. Featherstone, "The Calculation of Robot Dynamics Using Articulated-Body Inertias," *The Int. Journal of Robotics Research*, The MIT Press, vol. 2, pp. 13–30, Spring 1983.

[16] C. A. Balafoutis and R. V. Patel, "Efficient Computation of Manipulator Inertia Matrices and the Direct Dynamics Problem," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 1313–1321, Sept./Oct. 1989.

[17] H. Brandl, R. Johanni, and M. Otter, "A Very Efficient Algorithm for the Simulation of Robots and Similar Multibody Systems Without Inversion of the Mass Matrix," in *Proc. of IFAC/IFIP/IMACS Int. Symp. on Theory of Robots*, (Vienna, Austria), December 1986.

[18] M. Amin-Javaheri and D. E. Orin, "Systolic Architectures for the Manipulator Inertia Matrix," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 18, pp. 939–951, November/December 1988.

[19] C. S. G. Lee and P. R. Chang, "Efficient Parallel Algorithms For Robot Forward Dynamics Computation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, pp. 238–251, March/April 1988.

[20] P. Sadayappan, Y. L. C. Ling, K. W. Olson, and D. E. Orin, "A Restructurable VLSI Robotics Vector Processor Architecture for Real-Time Control," *IEEE Trans. on Robotics and Automation*, vol. 5, pp. 583–599, October 1989.

[21] A. Fijany and A. K. Bejczy, "Techniques for Parallel Computation of Mechanical Manipulator Dynamics. Part II: Forward Dynamics," *Control and Dynamic Systems*, vol. 40, pp. 357–410, 1991.

[22] C. L. Chen, C. S. G. Lee, and E. S. H. Hou, "Efficient Scheduling Algorithms for Robot Inverse Dynamics Computation on a Multiprocessor System," in *IEEE Int. Conf. on Robotics and Automation*, (Philadelphia, PA), pp. 1146–1151, 1988.

[23] J. Y. S. Luh and C. S. Lin, "Scheduling of Parallel Computation for a Computer-Controlled Mechanical Manipulator," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC–12, pp. 214–234, March/April 1982.

[24] K. W. Lilly and D. E. Orin, "Efficient Dynamic Simulation of Multiple Chain Robotic Mechanisms," To appear in *ASME Journal of Dynamic Systems, Measurement, and Control*, June 1994.

[25] K. W. Lilly and D. E. Orin, "Alternate Formulations for the Manipulator Inertia Matrix," *International Journal of Robotics Research,* The MIT Press, vol. 10, pp. 64–74, February 1991.

[26] J. J. Craig, *Introduction to Robotics: Mechanics and Control.* Reading, MA: Addison-Wesley, 1986.

[27] R. Featherstone, *Robot Dynamics Algorithms.* Boston: Kluwer Academic Publishers, 1987.

[28] J. Y. S. Luh, M. W. Walker, and R. P. Paul, "On-Line Computational Scheme for Mechanical Manipulators," *IEEE Transactions on Automatic Control*, vol. AC-35, no. 3, pp. 468–474, 1980.

[29] C. A. Balafoutis, R. V. Patel, and P. Misra, "Efficient Modeling and Computation of Manipulator Dynamics Using Orthogonal Cartesian Tensors," *IEEE Trans. of Robotics and Automation*, vol. 4, pp. 665–676, December 1988.

[30] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "A Spatial Operator Algebra for Manipulator Modeling and Control," *The Int. Journal of Robotics Research,* The MIT Press, vol. 10, pp. 371–381, August 1991.

[31] G. Rodriguez, "Recursive Forward Dynamics for Multiple Robot Arms Moving a Common Task Object," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 510–521, August 1989.

[32] S. H. Murphy, J. T. Wen, and G. N. Saridis, "Simulation of Cooperating Robot Manipulators on a Mobile Platform," *IEEE Trans. on Robotics and Automation*, vol. 7, pp. 468–478, August 1991.

[33] K. W. Lilly and D. E. Orin, "Efficient Dynamic Simulation of a Single Closed-Chain Manipulator," in *IEEE Int. Conf. on Robotics and Automation*, (Sacramento, California), pp. 210–215, April 1991.

[34] O. Khatib, "A Unified Approach for Motion and Force Control of Robot Manipulators: The Operational Space Formulation," *IEEE Trans. of Robotics and Automation*, vol. RA–3, pp. 43–53, February 1987.

[35] K. W. Lilly, *Efficient Dynamic Simulation of Robotic Mechanisms.* Norwell, MA: Kluwer Academic Publishers, 1993.

[36] S. McMillan, D. E. Orin, and P. Sadayappan, "Towards Super-Real-Time Simulation of Robotic Mechanisms Using a Parallel Integration Method," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, pp. 384–391, March/April 1992.

[37] K. W. Lilly and D. E. Orin, "Efficient $O(N)$ Recursive Computation of the Operational Space Inertia Matrix," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pp. 1384 – 1391, September/October 1993.

[38] M. Amin-Javaheri and D. E. Orin, "Parallel Algorithms for Computation of the Manipulator Inertia Matrix," *The International Journal of Robotics Research,* The MIT Press, vol. 10, pp. 162–170, April 1991.

[39] C. S. G. Lee and P. R. Chang, "Efficient Parallel Algorithms For Robot Inverse Dynamics Computation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC–16, pp. 532–542, July/August 1986.

[40] T. Kailath, *Linear Systems.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1980.

[41] L. G. Birta and O. Abou-Rabia, "Parallel Block Predictor-Corrector Methods for ODE's," *IEEE Trans. on Computers*, vol. C-36, pp. 299–311, March 1987.

[42] Y. F. Zheng, "Dynamic and Kinetic Behavior of Two Coordinating Robots in Assembly," *Mathl. Comput. Modelling*, vol. 12, no. 1, pp. 77-88, 1989.

[43] J. P. Keener, *Principles of Applied Mathematics.* Redwood City, CA: Addison Wesley, 1988.

[44] W. Khalil and J. F. Kleinfinger, "A New Geometric Notation for Open and Closed-Loop Robots," in *IEEE Int. Conf. on Robotics and Automation*, (San Francisco, CA), pp. 1174–1179, 1986.

[45] A. Fijany and A. Bejczy, "A Class of Parallel Algorithms for Computation of the Manipulator Inertia Matrix," *IEEE Trans. on Robotics and Automation*, vol. 5, pp. 600–615, October 1989.

[46] J. B. Newman and B. H. Robison, "Development of a Dedicated ROV for Ocean Science," *Marine Technology Society Journal*, vol. 26, pp. 46–53, Winter 1992.

[47] Anon., "Titan II High Resolution Bilateral Force Feedback Remote Manipulator System," Technical Report, Schilling Development, Inc., Davis, CA, April 1991.

[48] D. R. Yoerger and J.-J. E. Slotine, "Robust Trajectory Control of Underwater Vehicles," *IEEE Transactions of Oceanic Engineering*, vol. OE-10, pp. 462–470, October 1985.

[49] J. Yuh, "Modeling and Control of Underwater Robotic Vehicles," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, pp. 1475–1483, November/December 1990.

[50] K. Ioi and K. Itoh, "Modelling and Simulation of an Underwater Manipulator," *Advanced Robotics*, vol. 4, no. 4, pp. 303–317, 1990.

[51] A. K. Ramadorai and T. J. Tarn, "On Modeling and Adaptive Control of Underwater Robots," in *Proc. IARP 4th Workshop on Underwater Robots*, (Capri, Italy), pp. 47–58, June 1992.

[52] I. H. Shames, *Mechanics of Fluids.* New York, NY: McGraw-Hill, 1962.

[53] J. N. Newman, *Marine Hydrodynamics.* Cambridge, MA: The MIT Press, 1977.

[54] "Controllability," in *Principles of Naval Architecture* (E. Lewis, ed.), vol. 3, (New York, NY), Society of Naval Architects and Marine Engineers, 1987.

[55] J. L. Synge and B. A. Griffith, *Principles of Mechanics.* New York, NY: McGraw Hill, 1949.

[56] T. Sarpkaya and M. Isaacson, *Mechanics of Wave Forces on Offshore Structures.* New York, NY: Van Nostrand Reinhold Co., 1981.

[57] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing.* New York, NY: Cambridge University Press, 1988.

[58] S. K. Chakrabarti, W. A. Tam, and A. L. Wolbert, "Wave Forces on a Randomly Oriented Tube," in *Proc. of the O.T.C.*, no. 2190, (Houston, TX), 1975.

[59] O. M. Griffin and S. E. Ramberg, "Some Recent Studies of Vortex Shedding with Application to Marine Tubulars and Risers," *Transactions of the ASME*, vol. 104, pp. 2–13, March 1982.

[60] J. M. Cooke, M. J. Zyda, D. R. Pratt, and R. B. McGhee, "NPSNET: Flight Simulation Dynamic Modeling Using Quaternions," *Presence*, vol. 1, pp. 404–420, Fall 1992.

[61] G. Booch, *Object Oriented Design with Applications.* Redwood City, CA: Benjamin/Cummings Publishing Co., Inc., 1986.

[62] A. Register and M. Hopper, "Programming with Objects," *IEEE Potentials*, vol. 13, pp. 34–36, April 1994.

[63] J. R. Goetz, "Graphical Simulation of Articulated Rigid Body System Kinematics with Collision Detection," Master's thesis, Naval Postgraduate School, Monterey, CA 93943, March 1994.

[64] J. Wernecke, *IRIS Inventor Programming Guide.* Silicon Graphics, Inc., Mountain View, CA, 1992.

[65] P. McLendon, *IRIS Performer Programming Guide.* Silicon Graphics, Inc., Mountain View, CA, 1992.

[66] K. P. Wilson, M. J. Zyda, and D. R. Pratt, "NPSGDL: An Object Oriented Graphics Description Language for Virtual World Application Support," in *Proc. 3rd Eurographics Workshop on Object Oriented Graphics*, (Champrey, Switzerland), October 28–30 1992.

[67] K. J. R. W. Kristiansen, "A Computer Simulation of Vehicle and Actuator Dynamics for a Hexapod Walking Robot," Master's thesis, Naval Postgraduate School, Monterey, CA 93943, March 1994.

[68] S. McMillan, "Real-Time Simulation of a Robotic Manipulator Using a Parallel Integration Method," Master's thesis, The Ohio State University, Columbus, OH 43210, Spring 1990.

[69] R. B. McGhee and A. L. Pai, "An Approach to Computer Control for Legged Vehicles," *Journal of Terramechanics*, vol. 11, no. 1, pp. 9–27, 1974.

[70] J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *ASME Journal of Applied Mathematics*, pp. 215–221, June 1955.

[71] X. He and A. A. Goldenberg, "An Algorithm for Efficient Computation of Dynamics of Robotic Manipulators," in *Proc. Fourth Int. Conf. on Advanced Robotics*, (Columbus, OH), pp. 175–188, 1989.