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

Table 25: Inverse dynamics algorithm for a single, serial chain with a fixed base.

Given: $\boldsymbol{\omega}_0 = \mathbf{0}$, $\mathbf{a}_0 = [\mathbf{0}^T \; -\mathbf{g}^T]^T$, ${}^i\boldsymbol{R}_{i-1}$, ${}^{i-1}\boldsymbol{p}_i$, $\dot{q}_i$, $\ddot{q}_i$, ${}^N\mathbf{f}_{N+1}$ (external tip force).

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

$$\boldsymbol{\omega}_i = {}^i\boldsymbol{R}_{i-1}\boldsymbol{\omega}_{i-1} + \sigma_i \dot{q}_i \hat{\boldsymbol{z}}_i \tag{B.1}$$

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

$$\mathbf{f}'_i = \mathbf{I}_i\mathbf{a}_i + \begin{bmatrix} \boldsymbol{\omega}_i \times \bar{\boldsymbol{I}}_i\boldsymbol{\omega}_i \\ \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{h}_i) \end{bmatrix} \tag{B.3}$$

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.

---

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, $\boldsymbol{\omega}_i$, 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 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 $\boldsymbol{\omega}_i$ 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:

$$\boldsymbol{u}_i = \begin{bmatrix} \bar{I}_{xz}\Omega_{yx} - \bar{I}_{xy}\Omega_{zx} + \bar{I}_{yz}\left(\Omega_{yy} - \Omega_{zz}\right) + \hat{I}_z\Omega_{yz} - \hat{I}_y\Omega_{zy} \\ \bar{I}_{xy}\Omega_{zy} - \bar{I}_{yz}\Omega_{xy} + \bar{I}_{xz}\left(\Omega_{zz} - \Omega_{xx}\right) + \hat{I}_x\Omega_{zx} - \hat{I}_z\Omega_{xz} \\ \bar{I}_{yz}\Omega_{xz} - \bar{I}_{xz}\Omega_{yz} + \bar{I}_{xy}\left(\Omega_{xx} - \Omega_{yy}\right) + \hat{I}_y\Omega_{xy} - \hat{I}_x\Omega_{yx} \end{bmatrix}, \tag{B.12}$$

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:

$$\boldsymbol{f}_i' = m_i\boldsymbol{a}_i + \dot{\boldsymbol{\omega}}_i \times \boldsymbol{h}_i + \boldsymbol{\omega}_i \times \left[\boldsymbol{\omega}_i \times \boldsymbol{h}_i\right], \tag{B.17}$$

where the last two terms may be combined, much like in the translational acceleration equation, and the resulting equation is given by:

$$\boldsymbol{f}_i' = m_i\boldsymbol{a}_i + \boldsymbol{\Omega}_i\boldsymbol{h}_i. \tag{B.18}$$

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

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:		$94N-105$	$82N-92$

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

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.

---

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.

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, 35A] when joint $i+1$ is revolute ([32M, 32A] 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:

$$\begin{bmatrix} \bar{\boldsymbol{K}}_i^c & \tilde{\boldsymbol{h}}_i^c \\ \left(\tilde{\boldsymbol{h}}_i^c\right)^T & m_i^c \mathbf{1}_3 \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{I}}_i & \tilde{\boldsymbol{h}}_i \\ \tilde{\boldsymbol{h}}_i^T & m_i \mathbf{1}_3 \end{bmatrix} + \begin{bmatrix} {}^{i}\bar{\boldsymbol{K}}_{i+1}^c & {}^{i}\tilde{\boldsymbol{h}}_{i+1}^c \\ \left({}^{i}\tilde{\boldsymbol{h}}_{i+1}^c\right)^T & m_{i+1}^c \mathbf{1}_3 \end{bmatrix}, \tag{B.24}$$

where

$$\bar{\boldsymbol{K}}_i^c = \bar{\boldsymbol{I}}_i + {}^{i}\bar{\boldsymbol{K}}_{i+1}^c, \tag{B.25}$$

$$\boldsymbol{h}_i^c = \boldsymbol{h}_i + {}^{i}\boldsymbol{h}_{i+1}^c, \tag{B.26}$$

Table 28: Computation requirements for the joint space inertia matrix algorithm.

Computation	$\times$	$+$
$\mathbf{K}_i$	$32N - 41$	$44N - 57$
${}^i\mathbf{f}_i$	—	—
$[\mathbf{H}]_{i,i}$	—	—
${}^j\mathbf{f}_i$	$10N^2 - 23N + 16$	$6N^2 - 14N + 11$
$[\mathbf{H}]_{i,j}$	—	—
Total:	$10N^2 + 9N - 25$	$6N^2 + 30N - 46$
$(N = 6)$	(389)	(350)

$$m_i^c = m_i + m_{i+1}^c. \tag{B.27}$$

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, 44A]. 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, 350A]. 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, quan-

tities 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 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.

Table 29: Articulated Body dynamics algorithm for a single serial chain.

I. Forward Kinematics: Given: $\boldsymbol{\omega}_0 = \mathbf{0}$.
for $i = 1, \ldots, N$. Given: ${}^{i}\boldsymbol{R}_{i-1}$, ${}^{i-1}\boldsymbol{p}_i$, $\dot{q}_i$.

$$\boldsymbol{\omega}_i = {}^{i}\boldsymbol{R}_{i-1}\boldsymbol{\omega}_{i-1} + \sigma_i \dot{q}_i \hat{\mathbf{z}} \tag{B.28}$$

$$\boldsymbol{\zeta}_i = \begin{bmatrix} \mathbf{0} \\ {}^{i}\boldsymbol{R}_{i-1}\left[\boldsymbol{\omega}_{i-1} \times (\boldsymbol{\omega}_{i-1} \times {}^{i-1}\boldsymbol{p}_i)\right] \end{bmatrix} + \begin{bmatrix} \sigma_i(\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}_i) \\ \bar{\sigma}_i(2\boldsymbol{\omega}_i \times \dot{q}_i\hat{\mathbf{z}}_i) \end{bmatrix} \tag{B.29}$$

$$\boldsymbol{\beta}_i = -\begin{bmatrix} \boldsymbol{\omega}_i \times \bar{\boldsymbol{I}}_i\boldsymbol{\omega}_i \\ \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{h}_i) \end{bmatrix} \tag{B.30}$$

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.

$$\mathbf{n}_i = \mathbf{I}_i^*\boldsymbol{\phi}_i \tag{B.31}$$

$$m_i^* = \boldsymbol{\phi}_i^T\mathbf{I}_i^*\boldsymbol{\phi}_i \tag{B.32}$$

$$\mathbf{N}_i = \mathbf{I}_i^* - \mathbf{n}_i(m_i^*)^{-1}\mathbf{n}_i^T \tag{B.33}$$

$$\tau_i^* = \tau_i + \boldsymbol{\phi}_i^T\boldsymbol{\beta}_i^* \tag{B.34}$$

$$\mathbf{I}_{i-1}^* = \mathbf{I}_{i-1} + {}^{i}\mathbf{X}_{i-1}^T\mathbf{N}_i{}^{i}\mathbf{X}_{i-1} \tag{B.35}$$

$$\boldsymbol{\beta}_{i-1}^* = \boldsymbol{\beta}_{i-1} + {}^{i}\mathbf{X}_{i-1}^T\left[\boldsymbol{\beta}_i^* - \mathbf{N}_i\boldsymbol{\zeta}_i - \mathbf{n}_i(m_i^*)^{-1}\tau_i^*\right] \tag{B.36}$$

end for i.
III. Forward Accelerations: Given: $\mathbf{a}_0' = [\mathbf{0}^T \quad -{}^{0}\boldsymbol{a}_g^T]^T$.
for $i = 1, \ldots, N$.

$$\bar{\mathbf{a}}_i = {}^{i}\mathbf{X}_{i-1}\mathbf{a}_{i-1}' + \boldsymbol{\zeta}_i \tag{B.37}$$

$$\ddot{q}_i = (m_i^*)^{-1}\tau_i^* - \left[\mathbf{n}_i(m_i^*)^{-1}\right]^T\bar{\mathbf{a}}_i \tag{B.38}$$

$$\mathbf{a}_i' = \bar{\mathbf{a}}_i + \boldsymbol{\phi}_i\ddot{q}_i \tag{B.39}$$

end for i.

Table 30: Computational requirements of the AB dynamics algorithm for a single serial chain with revolute joints.

	$\times$	$+$
$\boldsymbol{\omega}_i$	$8N-12$	$5N-9$
$\boldsymbol{\zeta}_i$	$22N-33$	$10N-18$
$\boldsymbol{\beta}_i$	$27N-19$	$15N-15$
$\mathbf{n}_i$	—	—
$(m_i^*)^{-1}$	$N-1$	—
$\mathbf{n}_i\,(m_i^*)^{-1}$	$5N-5$	—
$\mathbf{N}_i$	$15N-30$	$15N-30$
τ_i^*	—	N
$\mathbf{I}_{i-1}^*$	$70N-80$	$86N-94$
$\boldsymbol{\beta}_{i-1}^*$	$50N-54$	$49N-55$
$\mathbf{a}_i'$	$20N-22$	$17N-21$
$\ddot{q}_i$	$6N-3$	$6N-4$
$\mathbf{a}_i$	—	$N-2$
Total:	$224N-259$	$205N-248$
$(N=6)$	(1085)	(982)

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 $[(224N-259)\text{M}, (205N-248)\text{A}]$. For $N=6$ this corresponds to [1085M, 982A]. 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.