SCISPACE
formerly Typeset

Open access • Proceedings Article • DOI:10.1109/ICRA.2012.6224600

A reduced-order recursive algorithm for the computation of the operational-space inertia matrix — Source link

Patrick M. Wensing, Roy Featherstone, David E. Orin

Institutions: Ohio State University, Australian National University

Published on: 14 May 2012 - International Conference on Robotics and Automation

Topics: Freivalds' algorithm, Ramer–Douglas–Peucker algorithm, Average-case complexity, Rabin–Karp algorithm and Computational complexity theory

Related papers:

- Efficient recursive algorithm for the operational space inertia matrix of branching mechanisms
- A unified approach for motion and force control of robot manipulators: The operational space formulation
- Hybrid algorithm for the computation of the matrix polynomial using a fractal number system
- Algorithm for faster computation of non-zero graph based invariants
- Algebraic Graph Theory for Sparse Flexibility Matrices

Share this paper:

View more about this paper here: https://typeset.io/papers/a-reduced-order-recursive-algorithm-for-the-computation-of-3zjtokscud

A Reduced-Order Recursive Algorithm for the Computation of the Operational-Space Inertia Matrix

Patrick Wensing, Roy Featherstone, David E. Orin

***Abstract*— This paper provides a reduced-order algorithm, the Extended-Force-Propagator Algorithm (EFPA), for the computation of operational-space inertia matrices in branched kinematic trees. The algorithm accommodates an operational space of multiple end-effectors, and is the lowest-order algorithm published to date for this computation. The key feature of this algorithm is the explicit calculation and use of matrices that propagate a force across a span of several links in a single operation. This approach allows the algorithm to achieve a computational complexity of $O(N + md + m^2)$ where N is the number of bodies, m is the number of end-effectors, and d is the depth of the system's connectivity tree. A detailed cost comparison is provided to the propagation algorithms of Rodriguez et al. (complexity $O(N + dm^2)$) and to the sparse factorization methods of Featherstone (complexity $O(nd^2 + md^2 + m^2d)$). For the majority of examples considered, our algorithm outperforms the previous best recursive algorithm, and demonstrates efficiency gains over sparse methods for some topologies.**

I. Introduction

Recursive dynamics algorithms for robotic mechanisms have enjoyed a history of success due to their low computational complexity. This class of algorithms has enabled efficient solutions to problems in forward [1], [2], inverse [3], and operational-space dynamics [4]. In this paper, we present the lowest-order algorithm to date for computation of the inverse of the operational-space matrix $\mathbf{\Lambda}^{-1}$, a key component in the operational-space dynamics formulation.

The original operational-space formulation [5] has unlocked a vast body of research over the past decades that enables, in principle, decoupling of task and null-space dynamics through operational-space control. While the operational-space formalism was originally developed to describe the dynamics of a single end-effector, it was extended to accommodate general task spaces that depend on the motion of more than one body in [6].

More recent work has also demonstrated the applicability of the operational-space formalism to the control of constrained and underactuated systems (e.g. [7], [8], [9], [10]). Sentis et al. extended the framework to provide operational control of free floating systems [7]. Park et al. introduced contact constrained operational dynamics for a humanoid in [8] which was later extended to operational control of contact force behavior in [9]. The work of Mistry and Righetti [10] addressed these challenges as well, but through a substantially different approach. Through considering the operational task within the framework of projected inverse dynamics, their results provide simplified control laws in comparison to previous work. This plethora of practical applications motivates the need for efficient operational-space dynamics algorithms to support these controllers.

Many algorithms have been developed to efficiently compute operational-space dynamics quantities. The largest body of work has concentrated on the operational-space inertia matrix, with original algorithms in [11], [12], [13] for single end-effector manipulators. The Force Propagation algorithm of Lilly [11], [14] was independently formulated by Kreutz-Delgado et al. [12] within their spatial operator algebra framework. An alternate approach of Lilly and Orin [13] approximates $\mathbf{\Lambda}$ through direct application of the articulated body inertia recursions of [1].

These approaches have been extended to a more general operational space that may include multiple end-effectors. Chang and Khatib presented a recursive method that extended Lilly's Force Propagation approach to this setting [15]. A very similar method had been developed previously by Rodriguez et al. [4]. An optimized version of the Rodriguez algorithm was provided in [16] and is used as a benchmark for recursive algorithms here.

An alternative line of research has developed methods that exploit sparsity in the structural components of the dynamic equations of motion to compute forward and operational dynamics [16], [17]. The kinematic branching that is found in complex mechanisms leads to specific patterns of zero elements in the system's mass matrix and task Jacobian. The exploitation of this sparsity pattern, referred to as branch-induced sparsity, has led to algorithms that are competitive with, and sometimes superior to, recursive approaches.

The algorithm presented in this paper, called the Extended-Force-Propagator Algorithm (EFPA), calculates $\mathbf{\Lambda}^{-1}$ highly efficiently, and achieves a computational complexity of $O(N + md + m^2)$ where N is the number of bodies, m is the number of end-effectors, and d is the depth of the system's connectivity tree. This beats the previous best reported complexity of $O(N + dm^2)$ achieved by the algorithm of Rodriguez et al. [4]. The key feature of the EFPA, which is responsible for both its high efficiency and low complexity, is the explicit calculation and use of matrices that propagate a force applied at an end-effector directly to an equivalent force at another body in the mechanism that may be several joints away from the end-effector. This approach substantially alters the structure of the computation, so that the EFPA has relatively few

P. Wensing is a PhD student in the Department of Electrical and Computer Engineering, The Ohio State University: wensing.2@osu.edu

R. Featherstone is with the College of Eng. and Computer Science, Australian National University: roy.featherstone@ieee.org

D. E. Orin is a Professor Emeritus in the Department of Electrical and Computer Engineering, The Ohio State University: orin.1@osu.edu

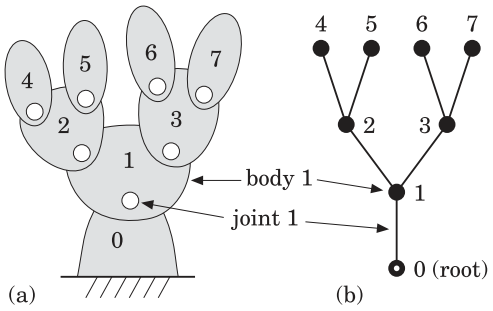

Fig. 1. (a) A kinematic tree and (b) its connectivity graph from [16].

intermediate results in common with previous algorithms.

The rest of this paper will be organized as follows. Section II reviews the conventions and notation that will be used to describe the dynamic properties and topology of a branched kinematic tree. Section III briefly reviews the operational-space dynamic equations of motion. Section IV introduces and derives the EFPA. Section V compares the algorithm's computational performance to the leading recursive and sparse techniques previously mentioned, and Section VI summarizes our work.

II. Kinematic Trees - Conventions and Notation

This section will outline the conventions and notation that will be used to describe the topology and dynamic properties of a rigid-body system. The conventions adopted in [18] will be employed. 6D spatial vector algebra will be used to enable compact notation for algorithm development and efficient spatial vector arithmetic for implementation. Tutorials on spatial vectors and their specific use in dynamics algorithms can be found in [19], [20].

A. Describing Connectivity

A rigid-body system can be modeled as a set of N bodies connected together by a set of joints, each with up to 6 degrees of freedom. The connectivity of such a system can generally be described by a graph, wherein each node represents a body, and each arc represents a joint. In this paper, we restrict our attention to those rigid-body systems which fall into the class of kinematic trees. The connectivity graph of such systems forms a tree, while physically, these systems are free of kinematic loops. An example of a kinematic tree is shown in Fig. 1(a) and its connectivity graph is shown in Fig. 1(b) [16].

Systems which fall into the class of kinematic trees can be modeled as a set of N bodies, a fixed base, and N joints. In the case of a mobile robot, one of the bodies is assigned as a floating base, and a six-degree-of-freedom (DoF) virtual joint is inserted between the fixed and floating base. The fixed base is labeled 0, while the other bodies are labeled 1 through N. Numbering may be done in any order such that the label of body i's parent, denoted $p(i)$, is less than i. The joints, represented by the arcs in the connectivity tree, are then numbered such that joint i connects bodies i and $p(i)$. Given these conventions, we make the following definitions:

$c(i)$: the set of children of body i, defined by $c(i) = \{j \mid p(j) = i\}$

$\kappa(i)$: the set of joints that support body i, defined by $\kappa(i) = \{i\} \cup \kappa(p(i))$ and $\kappa(0) = \emptyset$

Here, a joint is said to support body i precisely when it lies on the path between body i and the fixed base. Additionally, we use the support sets to define the nearest common ancestor for a pair of bodies as follows:

$$\text{ancest}(i, j) = \max(\kappa(i) \cap \kappa(j)). \tag{1}$$

For example, in Fig. 1, $\text{ancest}(2, 6) = 1$ and $\text{ancest}(4, 5) = 2$.

In this work we consider a set of m end-effectors, each of which is rigidly attached to a single body in the tree. We extend the connectivity tree to accommodate the end-effectors, which are numbered from $N + 1$ to $N + m$. As a result, $p(k)$ denotes the body to which end-effector k is rigidly attached. Letting $\kappa(k)$ be similarly defined as before, we define the following sets to describe the end-effectors supported (ES) by each joint in the tree:

$$ES(i) = \{k \mid N < k \text{ and } i \in \kappa(k)\}. \tag{2}$$

B. Spatial Notation

We briefly review some of the basic spatial quantities [18] that will be employed in the algorithm. A coordinate frame will be attached to each body in the tree to describe spatial quantities with respect to a local basis. The general joint notation of Roberson and Schwertassek [21] will be adopted to describe the relationship between connected links. With this notation, the spatial velocity $\mathbf{v}_i$ of link i is related to its parent through the following equation:

$$\mathbf{v}_i = \begin{bmatrix} \boldsymbol{\omega}_i \\ \boldsymbol{v}_i \end{bmatrix} = {}^i\boldsymbol{X}_{p(i)}\mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i\dot{\boldsymbol{q}}_i$$

where $\boldsymbol{\omega}_i$ and $\boldsymbol{v}_i$ are the angular and linear velocities of body i (as referenced to the local coordinate frame). The matrix ${}^i\boldsymbol{X}_{p(i)}$ provides a transformation of spatial motion vectors from $p(i)$ to i coordinates, and the matrix $\boldsymbol{\Phi}_i$ is a full-rank matrix that encodes joint i's free modes of motion. This matrix is dependent on the type of joint, but takes the simplified form $\boldsymbol{\Phi}_i = [0, 0, 1, 0, 0, 0]^T$ for revolute joints following the Denavit-Hartenberg convention.

Similarly, the matrix ${}^i\boldsymbol{X}^T_{p(i)}$ provides a spatial transformation of forces from i coordinates to $p(i)$ coordinates. Body i's rigid-body inertia tensor will be denoted $\boldsymbol{I}_i$, while the articulated body inertia of the subtree rooted at i will be denoted $\boldsymbol{I}^A_i$ [18]. Intuitively, $\boldsymbol{I}^A_i$ is the apparent inertia that would be "felt" by a force acting at the base of the subtree rooted at i if the subsystem were at rest and no joint torques were applied.

III. Operational Space Dynamics

This section briefly introduces the various quantities that are used to describe operational-space dynamics [5]. Given

a rigid-body system, with end-effectors as described previously, we first consider the standard dynamic equations of motion:

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} + \boldsymbol{J}(\boldsymbol{q})^T \boldsymbol{F} \tag{3}$$

where $\boldsymbol{H}$, $\boldsymbol{C}$, and $\boldsymbol{G}$ are the familiar mass matrix, velocity product terms, and gravitational terms respectively. Here $\boldsymbol{F}$ collects end-effector forces and $\boldsymbol{J}$ is a combined end-effector Jacobian that relates joint rates to end-effector velocities as,

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}\,. \tag{4}$$

Premultiplication of (3) by $\boldsymbol{J}\boldsymbol{H}^{-1}$ and incorporation of the time derivative of (4) provides:

$$\ddot{\boldsymbol{x}} = \boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{\tau} + \boldsymbol{\Lambda}^{-1}\boldsymbol{F} + \boldsymbol{\beta}, \tag{5}$$

where

$$\boldsymbol{\beta}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \boldsymbol{J}\boldsymbol{H}^{-1}\left(\boldsymbol{C} + \boldsymbol{G}\right), \text{ and}$$
$$\boldsymbol{\Lambda}^{-1}(\boldsymbol{q}) = \boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{J}^T\,.$$

Efficient computation of $\boldsymbol{\Lambda}^{-1}$ is required for any operational-space control application. The following section provides a recursive algorithm to compute this matrix which is then compared to existing approaches. Zero joint rates, zero joint torques, and no gravity are assumed for the algorithm, as they have no effect on $\boldsymbol{\Lambda}^{-1}$. Methods to efficiently compute $\boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{\tau} + \boldsymbol{\beta}$ can be found in [18], Section 2.5.4.

IV. The Extended-Force-Propagator Algorithm

Recursive algorithms for articulated structures have been shown to be very efficient, for example, in the calculation of forward dynamics with the articulated body algorithm (ABA) [18]. The ABA proceeds with three recursions. The first, outward from base-to-tips, calculates joint velocities and associated velocity dependent terms. The second recursion, inward from tips-to-base, then computes articulated body inertias and bias forces. The final recursion, once again outward, calculates body accelerations. In this paper, we extend the ABA to isolate the effects of the end-effector forces that are needed in operational-space dynamics.

The EFPA stems from the final two recursions of the articulated body algorithm. Its main feature is the recursive calculation and re-use of the extended force propagator matrices ${}^k\boldsymbol{\mathcal{X}}_i^T$, in which body i may be arbitrarily far away from end-effector k. While the standard spatial force transform ${}^k\boldsymbol{X}_i^T$ provides a transformation of spatial forces from k to i as if the bodies are locked at the joints, the matrix ${}^k\boldsymbol{\mathcal{X}}_i^T$ provides an *articulated* transformation as if the joints are free to move. As a result, ${}^k\boldsymbol{\mathcal{X}}_i^T\boldsymbol{f}_k$ describes the force that is felt at i due to a force $\boldsymbol{f}_k$ at k.

Analogous to the relationship between ${}^k\boldsymbol{X}_i^T$ and ${}^k\boldsymbol{X}_i$, ${}^k\boldsymbol{\mathcal{X}}_i$ is an extended acceleration propagator that transforms an acceleration at body i to body k given that the system is free at the joints. As such, ${}^k\boldsymbol{\mathcal{X}}_i$ and ${}^k\boldsymbol{\mathcal{X}}_i^T$ can be referred to as *articulated transforms* as in [14] to convey their operation.

The concept of an articulated transform is not new. For example, ${}^k\boldsymbol{\mathcal{X}}_i^T$ appears in [15] as ${}^i_k\boldsymbol{L}^*$ and in [4] as $\psi(i,k)$. However, the EFPA is the first algorithm to explicitly compute and use these matrices for efficiency gains.

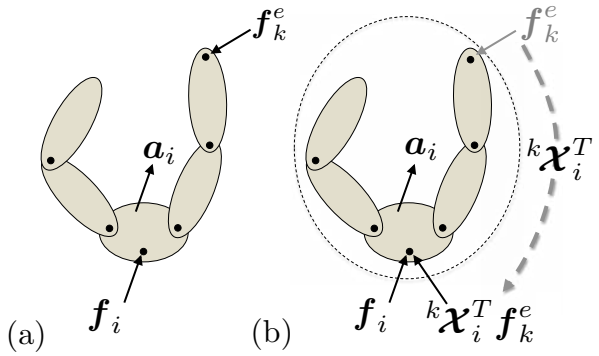

Fig. 2. (a) Spatial quantities used to describe the dynamics of body i in the subtree rooted at i. (b) Recursive relationships obtained by the inward recursion of the EFPA. An articulated inertia is used to describe the effective inertia of all outlined bodies. The force propagator ${}^k\boldsymbol{\mathcal{X}}_i^T$ transforms the dynamic effect of the end-effector force $\boldsymbol{f}_k^e$ through bodies k to i to describe the force effect on the i-th body.

A. Inward Recursion

The first pass of the EFPA proceeds from tip to base. It seeks to propagate the following equation:

$$\boldsymbol{f}_i = \boldsymbol{I}_i^A\boldsymbol{a}_i - \sum_{k\in ES(i)} {}^k\boldsymbol{\mathcal{X}}_i^T\boldsymbol{f}_k^e\,, \tag{6}$$

which provides a force-acceleration relationship for body i of the articulated subtree rooted at joint i. Here $\boldsymbol{f}_k^e$ represents a force which acts at end-effector k. The articulated transforms ${}^k\boldsymbol{\mathcal{X}}_i^T$ are extended force propagators as described previously and illustrated in Fig. 2. This inward recursion is more complex in comparison to previous algorithms [4], [15], but enables computational savings in the outward recursion. Initialization of (6) at the end-effector bodies seeds this recursion with ${}^k\boldsymbol{\mathcal{X}}_{p(k)} = {}^k\boldsymbol{X}_{p(k)}$.

The assumptions of zero joint torques and zero velocities provide the following relationships at each joint:

$$\boldsymbol{\Phi}_i^T\boldsymbol{f}_i = \boldsymbol{0}, \tag{7}$$

$$\boldsymbol{a}_i = {}^i\boldsymbol{X}_{p(i)}\,\boldsymbol{a}_{p(i)} + \boldsymbol{\Phi}_i\,\ddot{\boldsymbol{q}}_i, \text{ and} \tag{8}$$

$$\boldsymbol{f}_{p(i)} = \boldsymbol{I}_{p(i)}\,\boldsymbol{a}_{p(i)} + \sum_{j\in c(p(i))} {}^j\boldsymbol{X}_{p(i)}^T\,\boldsymbol{f}_j\,. \tag{9}$$

Similar to the ABA derivation [22], relationships (6)-(8) can be combined to relate the acceleration of the i-th subtree to the predecessor acceleration and end-effector forces. This relationship can then be combined with (9) to provide:

$$\boldsymbol{f}_{p(i)} = \left(\boldsymbol{I}_{p(i)} + \sum_{j\in c(p(i))} {}^j\boldsymbol{X}_{p(i)}^T\,\boldsymbol{L}_j^T\,\boldsymbol{I}_j^A\,{}^j\boldsymbol{X}_{p(i)}\right)\boldsymbol{a}_{p(i)} - \sum_{j\in c(p(i))}\sum_{k\in ES(j)} {}^j\boldsymbol{X}_{p(i)}^T\,\boldsymbol{L}_j^T\,{}^k\boldsymbol{\mathcal{X}}_j^T\,\boldsymbol{f}_k^e, \tag{10}$$

where each $\boldsymbol{L}_i^T$ is a force propagator across the i-th joint and takes the form:

$$\boldsymbol{L}_i^T = \boldsymbol{1}_{6\times 6} - \boldsymbol{I}_i^A\boldsymbol{K}_i$$

where $\boldsymbol{K}_i = \boldsymbol{\Phi}_i \left(\boldsymbol{\Phi}_i^T \boldsymbol{I}_i^A \boldsymbol{\Phi}_i\right)^{-1} \boldsymbol{\Phi}_i^T$. While the quantity in parenthesis in (10) provides the familiar recursive relationship for the articulated body inertia, the remaining portion provides the following recursive relationships for each ${}^k\boldsymbol{\mathcal{X}}_i$:

$$ {}^k\boldsymbol{\mathcal{X}}_{p(i)} := {}^k\boldsymbol{\mathcal{X}}_i \, \boldsymbol{L}_i \, {}^i\boldsymbol{X}_{p(i)}. \tag{11} $$

We note that the product $\boldsymbol{L}_i \, {}^i\boldsymbol{X}_{p(i)}$ used here is an articulated acceleration transform (an acceleration propagator) across a single joint. This recursion is required no more than d times for each end-effector. This leads to a computational complexity of $O(md)$ to calculate all required articulated transforms. The articulated inertia computation has complexity $O(N)$, providing a overall complexity of $O(N+md)$ for this pass of the algorithm.

B. Outward Recursion

The final goal of the outward recursion is to produce the combined force-acceleration relationship $\boldsymbol{a} = \boldsymbol{\Lambda}^{-1}\boldsymbol{f}$ for the end-effectors. This relationship is arrived at recursively, from base to tips, through the solution for accelerations at intermediate links along the tree. This is accomplished through propagation of the following equation:

$$ \boldsymbol{a}_i = \sum_{k=N+1}^{N+m} \boldsymbol{\Lambda}_{ik}^{-1} \, \boldsymbol{f}_k^e \ , \tag{12} $$

where each $\boldsymbol{\Lambda}_{ik}^{-1}$ describes the acceleration at link i caused by end-effector force $\boldsymbol{f}_k^e$. We note that $\boldsymbol{\Lambda}_{ik}^{-1}$ is not a block of $\boldsymbol{\Lambda}^{-1}$ when i refers to a body. The assumption that the fixed base is stationary seeds this recursion with:

$$ \boldsymbol{\Lambda}_{0k}^{-1} = \boldsymbol{0}_{6\times 6}. $$

To propagate (12) to body i from its predecessor, the combination of (12) at $p(i)$ along with (6)-(8) leads to:

$$ \begin{aligned} \boldsymbol{a}_i = & \sum_{k \notin ES(i)} \boldsymbol{L}_i \, {}^i\boldsymbol{X}_{p(i)} \, \boldsymbol{\Lambda}_{p(i)\,k}^{-1} \, \boldsymbol{f}_k^e \\ & + \sum_{k \in ES(i)} \left(\boldsymbol{L}_i \, {}^i\boldsymbol{X}_{p(i)} \, \boldsymbol{\Lambda}_{p(i)\,k}^{-1} + \boldsymbol{K}_i \, {}^k\boldsymbol{\mathcal{X}}_i^T \right) \boldsymbol{f}_k^e \, . \end{aligned} $$

That is, the recursive relationship between $\boldsymbol{\Lambda}_{i\,k}^{-1}$ and $\boldsymbol{\Lambda}_{p(i)\,k}^{-1}$ is dependent on whether or not joint i supports end-effector k. For each $k \in ES(i)$ we have the recursion:

$$ \boldsymbol{\Lambda}_{i\,k}^{-1} := \boldsymbol{L}_i \, {}^i\boldsymbol{X}_{p(i)} \, \boldsymbol{\Lambda}_{p(i)\,k}^{-1} + \boldsymbol{K}_i \, {}^k\boldsymbol{\mathcal{X}}_i^T \ , \tag{13} $$

while

$$ \boldsymbol{\Lambda}_{i\,k}^{-1} := \boldsymbol{L}_i \, {}^i\boldsymbol{X}_{p(i)} \, \boldsymbol{\Lambda}_{p(i)\,k}^{-1} $$

for each $k \notin ES(i)$. Thus, if i does not support end-effector k_2, the acceleration influence of $\boldsymbol{f}_{k_2}^e$ propagates from $p(i)$ to i according to an articulated acceleration transform (as if the joint i were free to move). As a result, for any other end-effector k_1 in the subtree at i we have the following:

$$ \boldsymbol{\Lambda}_{k_1\,k_2}^{-1} := {}^{k_1}\boldsymbol{\mathcal{X}}_{p(i)} \, \boldsymbol{\Lambda}_{p(i)\,k_2}^{-1}. \tag{14} $$

This simplification is illustrated in Fig. 3 for a basic example. We note that this simplification is first possible when i is the child of ancest(k_1, k_2). In the example figure, an acceleration propagator may be applied to calculate $\boldsymbol{\Lambda}_{k_1\,k_2}^{-1}$ recursively through links 1, 2, and 3. Yet, computational savings can occur when $\boldsymbol{\Lambda}_{k_1\,k_2}^{-1}$ is computed directly at the common ancestor of k_1 and k_2 through the use of ${}^{k_1}\boldsymbol{\mathcal{X}}_1$. With this insight we define the following set:

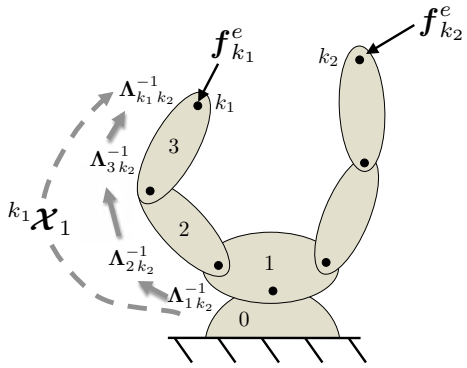

Fig. 3. Simplification to the outward recursion that is provided by the acceleration propagator ${}^{k_1}\boldsymbol{\mathcal{X}}_1$. It's transpose, the force propagator ${}^1\boldsymbol{\mathcal{X}}_{k_1}^T$ was calculated during the inward recursion. As a result, three recursive steps for the calculation of $\boldsymbol{\Lambda}_{k_1\,k_2}^{-1}$ are able to be replaced with one matrix multiplication.

$$ \begin{aligned} GCA(i) = \{(k_1, k_2) \mid & \ k_1 < k_2 \\ & \text{and} \ \ i = \text{ancest}(k_1, k_2)\} \end{aligned} \tag{15} $$

which contains all end-effector pairs that have a greatest common ancestor at i. The EFPA, uses this set to determine the links at which the cross-terms of $\boldsymbol{\Lambda}^{-1}$ may be computed.

Through the use of these simplifications, the recursion listed in (13) is required not more than d times for each end-effector. This ultimately leads to the computation of each $\boldsymbol{\Lambda}_{kk}^{-1}$ and has overall complexity of $O(md)$. These quantities represent the diagonal blocks of $\boldsymbol{\Lambda}^{-1}$. Each of the cross terms in $\boldsymbol{\Lambda}^{-1}$ can then be calculated via an appropriate version of (14), resulting in a cross-term calculation complexity of $O(m^2)$. Thus, the overall complexity of this recursion is $O(md + m^2)$. Including both passes of the algorithm, a complexity of $O(N + md + m^2)$ is achieved.

The full EFP algorithm is listed in Table I. The second and third loops provide implementations of the inward and outward recursions respectively. We note that due to the symmetry of $\boldsymbol{\Lambda}^{-1}$, only the upper triangle of $\boldsymbol{\Lambda}^{-1}$ is computed by the algorithm. The final loop provides the necessary transformation for each $\boldsymbol{\Lambda}_{k\,k}^{-1}$ into end-effector coordinates. This additional transformation loop is not required for off-diagonal terms, as the acceleration propagators ${}^k\boldsymbol{\mathcal{X}}_i$ transform accelerations directly to the end-effector frames.

V. Algorithm Comparison

This section will provide a comparison of the computational performance of the EFPA to the algorithms of Rodriguez et al. [4] and Featherstone [16]. We first briefly describe the operation of these algorithms.

Initialize: $\boldsymbol{\Lambda}_{0k}^{-1} = \mathbf{0}_{6\times 6}, \quad \boldsymbol{I}_i^A = \boldsymbol{I}_i$

for $k = N+1$ **to** $N+m$ **do**
  ${}^k\boldsymbol{\mathcal{X}}_{p(k)}^T = {}^k\boldsymbol{X}_{p(k)}^T$
end for k

for $i = N$ **to** 1 **do**
  $\boldsymbol{K}_i = \boldsymbol{\Phi}_i \left(\boldsymbol{\Phi}_i^T \boldsymbol{I}_i^A \boldsymbol{\Phi}_i\right)^{-1} \boldsymbol{\Phi}_i^T$
  $\boldsymbol{L}_i = \mathbf{1}_{6\times 6} - \boldsymbol{K}_i \boldsymbol{I}_i^A$
  if $p(i) \neq 0$ **then**
    $\boldsymbol{I}_{p(i)}^A = \boldsymbol{I}_{p(i)}^A + {}^i\boldsymbol{X}_{p(i)}^T \boldsymbol{L}_i^T \boldsymbol{I}_i^A {}^i\boldsymbol{X}_{p(i)}$
    for all $k \in ES(i)$ **do**
      ${}^k\boldsymbol{\mathcal{X}}_{p(i)} = {}^k\boldsymbol{\mathcal{X}}_i \boldsymbol{L}_i {}^i\boldsymbol{X}_{p(i)}$
    end for k
  end
end for i

for $i = 1$ **to** N **do**
  for all $k \in ES(i)$ **do**
    $\boldsymbol{\Lambda}_{ik}^{-1} = \boldsymbol{L}_i {}^i\boldsymbol{X}_{p(i)} \boldsymbol{\Lambda}_{p(i)\,k}^{-1} + \boldsymbol{K}_i {}^k\boldsymbol{\mathcal{X}}_i^T$
  end for k
  for all $(k_1, k_2) \in GCA(i)$ **do**
    $\boldsymbol{\Lambda}_{k_1 k_2}^{-1} = {}^{k_1}\boldsymbol{\mathcal{X}}_i \boldsymbol{\Lambda}_{i\,k_2}^{-1}$
  end for k
end for i

for $k = N+1$ **to** $N+m$ **do**
    $\boldsymbol{\Lambda}_{kk}^{-1} = {}^k\boldsymbol{X}_{p(k)} \boldsymbol{\Lambda}_{p(k)\,k}^{-1}$
end for i

$$\boldsymbol{\Lambda}^{-1} = \begin{bmatrix} \boldsymbol{\Lambda}_{N+1,N+1}^{-1} & \cdots & \boldsymbol{\Lambda}_{N+1,N+m}^{-1} \\ \vdots & \ddots & \vdots \\ \boldsymbol{\Lambda}_{N+1,N+m}^{-T} & \cdots & \boldsymbol{\Lambda}_{N+m,N+m}^{-1} \end{bmatrix}$$

TABLE I
EXTENDED-FORCE-PROPAGATOR ALGORITHM

A. RJK Algorithm

The algorithm of Rodriguez, Jain and Kreutz-Delgado (RJKA) is described in [4] and a dramatically optimized version can be found in [16]. The main differences between the RJKA and EFPA can be summarized as follows.

- Only the EFPA calculates extended force propagators.
- The RJKA calculates $\boldsymbol{\Lambda}_{kk}^{-1}$ via a recursion that calculates $\boldsymbol{\Lambda}_{ii}^{-1}$ from $\boldsymbol{\Lambda}_{p(i)p(i)}^{-1}$, whereas the EFPA instead calculates $\boldsymbol{\Lambda}_{ik}^{-1}$ from $\boldsymbol{\Lambda}_{p(i)k}^{-1}$.
- The RJKA calculates $\boldsymbol{\Lambda}_{k_1 k_2}^{-1}$ recursively from a common ancestor using local articulated transforms only, whereas the EFPA calculates these quantities in a single step using (14).

B. Sparse Factorization Algorithm (SFA)

Featherstone's sparse factorization algorithm (SFA) [16] is based on the exploitation of branch-induced sparsity in the system mass matrix $\boldsymbol{H}$ and the rows of the task Jacobian $\boldsymbol{J}$. The algorithm first uses the factorization approach of [17] to express the mass matrix as $\boldsymbol{H} = \boldsymbol{L}^T \boldsymbol{L}$, where $\boldsymbol{L}$ is a lower-triangular matrix that enjoys the same sparsity pattern as the lower triangle of $\boldsymbol{H}$. The class of lower-triangular matrices that possesses this branch-induced sparsity pattern is shown to be a group, which implies the same sparsity pattern in $\boldsymbol{L}^{-1}$, and thus in the rows of $\boldsymbol{Y} = \boldsymbol{J}\boldsymbol{L}^{-1}$. This definition provides:

$$\boldsymbol{\Lambda}^{-1} = \boldsymbol{J}\boldsymbol{H}^{-1}\boldsymbol{J}^T = \boldsymbol{Y}\boldsymbol{Y}^T .$$

The properties of branch-induced sparsity are then exploited to greatly accelerate the computation of $\boldsymbol{Y}$ and $\boldsymbol{\Lambda}^{-1}$. The final algorithm to compute $\boldsymbol{\Lambda}^{-1}$ can be shown to have computational complexity $O(nd^2 + md^2 + m^2 d)$ where n represents the total number of degrees of freedom in the system.

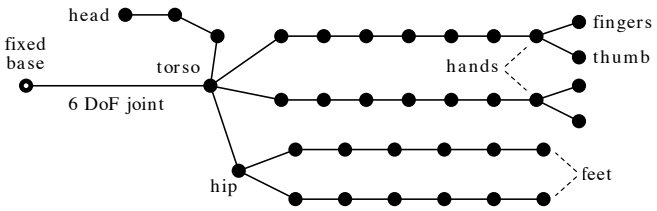

Fig. 4. Connectivity graph of the ASIMO Next-Generation robot (modified from [16]).

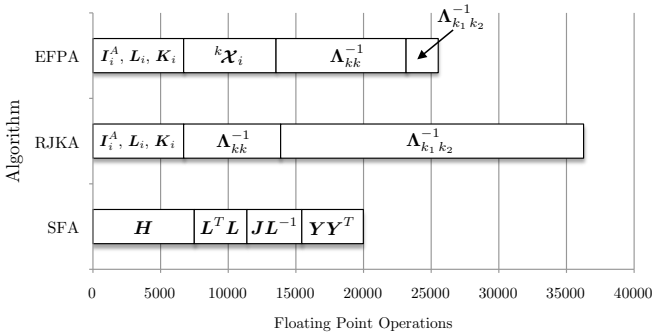

Fig. 5. Cost comparison breakdown for the Hands and Feet operational-space example for Fig. 4. Although the EFPA incurs additional cost on the inward recursion to calculate each ${}^k\boldsymbol{\mathcal{X}}_i$, this enables savings to calculate the off-diagonal terms of $\boldsymbol{\Lambda}_{k_1 k_2}^{-1}$. Still, the sparsity from this topology provides advantage to the SFA.

C. Computational Examples

To understand the comparative performance of the algorithms, this section presents the floating point operation counts (flops) for the calculation of $\boldsymbol{\Lambda}^{-1}$ in a number of examples. We mainly consider the same examples as those in [16] for the ASIMO Next-Generation humanoid robot and derived mechanisms. This floating base humanoid consists of $N = 35$ bodies, with connectivity shown in Fig. 4. All joints aside from the floating base are modeled as revolutes.

We first consider an operational space consisting of the position and orientation of the hands and feet (4 end-effectors) for this system. Figure 5 provides a breakdown of the cost to calculate $\boldsymbol{\Lambda}^{-1}$ for each algorithm. Further details on the computational costs for the RJK and EFP algorithms are provided in the Appendix.

This example highlights the advantages afforded by the computation of the extended-force-propagators over previous recursive algorithms. In comparison to the RJKA, the EFPA incurs additional cost during the inward recursion to obtain

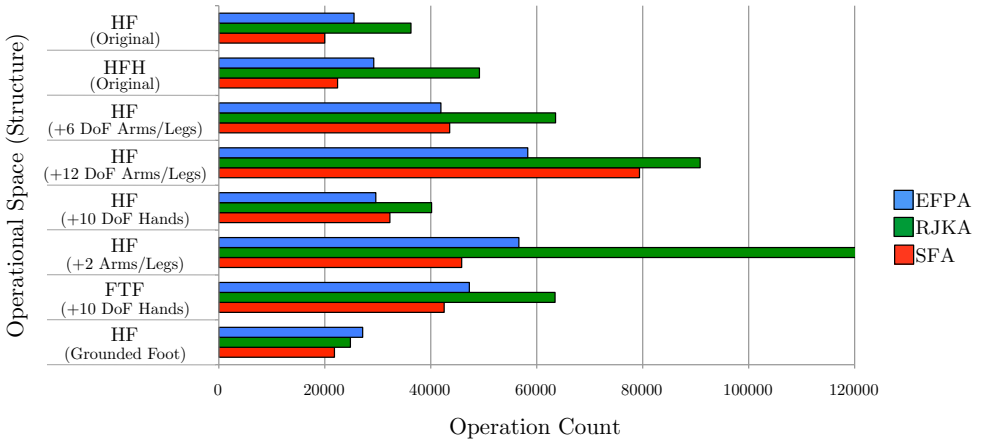

Fig. 6. Cost comparison for operational-space algorithms. Operational spaces are abbreviated as: HF = Hands and Feet; HFH = Hands, Feet, and Head; FTF = Forefingers, Thumbs, and Feet. This figure shows the benefits of the EFPA for systems with less sparsity in the mass matrix.

each ${}^k\boldsymbol{\mathcal{X}}_i$. Yet, this enables significant savings in the computation of the cross-terms of $\boldsymbol{\Lambda}^{-1}$, which is by far the most expensive step in the RJKA. The sparsity of the system mass matrix still allows the SFA to obtain $\boldsymbol{\Lambda}^{-1}$ with 78 percent of the flops when compared to the EFPA. Approximately 67 percent of the elements of $\boldsymbol{J}$ and 56 percent of the elements of $\boldsymbol{H}$ are zero in this case.

The next series of examples show the benefits of the reduced order EFPA for higher DoF systems that lack a high-degree of sparsity in their mass matrix. Figure 6 shows the computational costs for a series of alterations to the ASIMO Next-Generation mechanism. We consider additional DoFs in the appendages, additional appendages, and the modification to 12-DoF four-fingered hands. The EFPA provides advantages over the other recursive algorithm (RJKA) in every case. This is largely the result of RJKA's high calculation cost for the cross-terms of $\boldsymbol{\Lambda}^{-1}$.

The modifications of additional DoFs in the arms, legs, or hands adversely effect the sparsity of the system's mass matrix and thus provide advantages for the EFPA over the SFA. The addition of extra DoFs in any of the appendages lengthens the unbranched chains in the mechanism's connectivity tree. These unbranched chains lead to fully dense blocks in the system mass matrix, reducing the benefits of sparse techniques. Thus, the SFA exhibits a lower relative efficiency in these cases.

The EFPA is outperformed when 2 arms and 2 legs are added, an unexpected result for the low-order algorithm. This is mainly a result of the high-degree of sparsity in the mass matrix for this mechanism. The mass matrix has 73 percent zero elements, providing advantages to the SFA. Despite this sparsity increase, the cost ratio to the EFPA is nearly the same as in the base example. This property is not shared by the RJKA, whose cost ratio to the SFA increases over 50 percent in comparison to the base example.

The fore-fingers, thumbs, feet (FTF) operational-space example does highlight a potential improvement for the EFPA. Considering the fore-finger and thumb end-effectors for the left hand, these end effectors have a long, common, support chain (from the base to the left hand). As a result, force propagators for these end-effectors have much common structure. This fact is not currently exploited, and provides opportunity for algorithmic improvements in similar cases.

The final example is another case that would benefit from this type of improvement. In this example, the humanoid is balanced on one foot, which is treated as a fixed-base, with the hands and free foot as the operational space. Here, the links between the torso and grounded foot provide a common support chain that could be exploited in the computation of many of the extended-force-propagators. This current drawback of our algorithm, repeated propagation calculations over common support chains, is not shared with the RJKA, providing advantage to it in this and similar examples.

VI. Conclusions and Future Work

This paper has detailed the Extended-Force-Propagator Algorithm, the lowest-order algorithm to date for the computation of $\boldsymbol{\Lambda}^{-1}$. The algorithm is shown to provide computational efficiency benefits over the optimized RJK algorithm [4], [16], the previous benchmark for recursive $\boldsymbol{\Lambda}^{-1}$ algorithms. The recursive approach is able to maintain efficient computation for systems that lack sparsity, providing benefits over the sparse techniques of Featherstone for some topologies. These computational benefits have been enabled by the use of extended-force-propagators, which provide articulated transformations of spatial quantities over spans of links. This represents the first time these quantities have been explicitly computed and used in recursive dynamics algorithms.

Aside from the potential algorithmic improvements noted previously, future work will include the extension of the extended-force-propagator approach to calculate other quantities of interest in operation-space dynamics, such as the dynamically consistent Jacobian pseudo-inverse $\overline{\boldsymbol{J}}$. The sparse matrix approach currently has a large computational advantage over the EFPA if $\overline{\boldsymbol{J}}$ is required in addition to $\boldsymbol{\Lambda}^{-1}$, as the sparse factors of $\boldsymbol{\Lambda}^{-1}$ and $\boldsymbol{H}$ can be used to accelerate computation of $\overline{\boldsymbol{J}}$. Preliminary work also shows promise for the extension of our approach to the recursive computation of

closed-chain operational dynamics, a realm where sparsity-based approaches have yet to show their applicability.

VII. Acknowledgments

This work was supported in part by fellowships from the The Ohio State University and the NSF to Patrick Wensing, and by Grant No. CNS-0960061 from the NSF with a subaward to The Ohio State University.

Appendix

The computational costs of each of the algorithms follow mainly from the results presented in [16] (Table 4, Appendix B). Note that this table uses the notation of [4]. The correspondence to notation in this paper is established below. The correspondence in the last line is only approximate, as

[16], Table 4	EFPA
$\boldsymbol{D}_i^{-1}$	$(\boldsymbol{\Phi}_i^T \boldsymbol{I}_i^A \boldsymbol{\Phi}_i)^{-1}$
$\overline{\boldsymbol{\tau}}_i$	$\boldsymbol{L}_i^T$
$\boldsymbol{\phi}_i$	${}^i\boldsymbol{X}_{p(i)}^T$
$\boldsymbol{P}_i$	$\boldsymbol{I}_i^A$
$\boldsymbol{\Omega}_{ij}$	$\boldsymbol{\Lambda}_{ij}^{-1}$

the RJKA computes $\boldsymbol{\Lambda}^{-1}$ from $\boldsymbol{\Omega}$ in a post-processing step to accommodate general end-effector coordinates.

Each of the spatial operations in the algorithms employ various efficiency tricks that are enabled by each recursive step taking place in local coordinates. For the RJKA, we modify the assumptions in [16] as follows:

- As noted in [23], the floating-base coordinate system can be located such that it is related by a pure rotation about a fixed-axis to a privileged child. This provides the following reduced costs for the privileged child:

Calculation	Cost	Flops
${}^i\boldsymbol{X}_{p(i)}^T\, \boldsymbol{I}_i^A\, \boldsymbol{L}_i\, {}^i\boldsymbol{X}_{p(i)}$	22m + 25a	47
$\boldsymbol{I}_{p(i)}^A + (\ldots)$	15 a	15
${}^i\boldsymbol{X}_{p(i)}\, \boldsymbol{\Lambda}_{p(i)k}^{-1}$	48 m + 24 a	72
${}^i\boldsymbol{X}_{p(i)}\, \boldsymbol{\Lambda}_{p(i)p(i)}^{-1}\, {}^i\boldsymbol{X}_{p(i)}^T$	63 m + 54 a	117

Cost modifications for transformations that require ${}^i\boldsymbol{X}_{p(i)}$ where i is the privileged child of the floating-base.

- The floating-base joint is not modeled as revolute. This requires additional computation for $\boldsymbol{K}_1 = \left(\boldsymbol{I}_1^A\right)^{-1}$ which is carried out through an LDL^T factor and invert. This amounts to a cost of 231 flops.
- We allow general end-effector coordinates, which requires post processing of the matrix $\boldsymbol{\Omega}$ found in [16] to produce $\boldsymbol{\Lambda}^{-1}$. This requires a series of spatial transforms with cost 137m + 137a to compute each $\boldsymbol{\Lambda}_{kk}^{-1}$ (a spatial congruence in this case) and cost 288m + 216a to compute each $\boldsymbol{\Lambda}_{k_1k_2}^{-1}$.

We employ the same efficiency tricks for the implementation of the EFPA. Substantial cost is incurred by the application of ${}^k\boldsymbol{\mathcal{X}}_i$ across multiple links. That is, multiplications:

$$\boldsymbol{\Lambda}_{k_1k_2}^{-1} = {}^{k_1}\boldsymbol{\mathcal{X}}_i\, \boldsymbol{\Lambda}_{i\,k_2}^{-1} \quad \text{and} \quad \boldsymbol{\Lambda}_{1\,k}^{-1} = \boldsymbol{K}_1\, {}^k\boldsymbol{\mathcal{X}}_1^T$$

require dense matrix multiplications with cost 216m + 180a in general. (This cost is reduced by 36m+36a for the second operation if k is supported by the privileged child, since ${}^k\boldsymbol{\mathcal{X}}_1^T$ has a zero row in this case.) Despite this high cost, application of ${}^k\boldsymbol{\mathcal{X}}_i$ is still cheaper than repeated application of local transforms, even for relatively short chains.

References

[1] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *Int. J. of Rob. Research*, vol. 2, no. 1, pp. 13–30, 1983.
[2] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 104, no. 3, pp. 205–211, 1982.
[3] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, no. 2, pp. 69–76, 1980.
[4] G. Rodriguez, A. Jain, and K. Kreutz-Delgado, "Spatial operator algebra for multibody system dynamics," *Journal of the Astronautical Sciences*, vol. 40, pp. 27–50, Jan-Mar 1992.
[5] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal of Robotics and Automation*, vol. 3, pp. 43–53, February 1987.
[6] J. Russakow, O. Khatib, and S. Rock, "Extended operational space formulation for serial-to-parallel chain (branching) manipulators," in *Proc. of the IEEE Int. Conf. on Rob. and Auto.*, pp. 1056–1061, 1995.
[7] L. Sentis and O. Khatib, "Control of free-floating humanoid robots through task prioritization," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1718–1723, April 2005.
[8] J. Park and O. Khatib, "Contact consistent control framework for humanoid robots," in *Proc. of the IEEE Int. Conference on Robotics and Automation*, pp. 1963–1969, May 2006.
[9] L. Sentis, J. Park, and O. Khatib, "Compliant control of multicontact and center-of-mass behaviors in humanoid robots," *IEEE Transactions on Robotics*, vol. 26, pp. 483–501, June 2010.
[10] M. Mistry and L. Righetti, "Operational space control of constrained and underactuated systems," in *Proceedings of Robotics: Science and Systems*, (Los Angeles, CA, USA), June 2011.
[11] K. W. Lilly, *Efficient Dynamic Simulation of Robotic Mechanisms*. PhD thesis, The Ohio State University, 1989.
[12] K. Kreutz-Delgado, A. Jain, and G. Rodriguez, "Recursive formulation of operational space control," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 1750–1753, Apr. 1991.
[13] K. Lilly and D. Orin, "Efficient O(N) recursive computation of the operational space inertia matrix," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 5, pp. 1384–1391, 1993.
[14] K. W. Lilly, *Efficient Dynamic Simulation of Robotic Mechanisms*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.
[15] K.-S. Chang and O. Khatib, "Efficient recursive algorithm for the operational space inertia matrix of branching mechanisms," *Advanced Robotics*, vol. 14, no. 8, pp. 703–715, 2001.
[16] R. Featherstone, "Exploiting sparsity in operational-space dynamics," *Int. J. of Robotics Research*, vol. 29, no. 10, pp. 1353–1368, 2010.
[17] R. Featherstone, "Efficient factorization of the joint-space inertia matrix for branched kinematic trees," *International Journal of Robotics Research*, vol. 24, no. 6, pp. 487–500, 2005.
[18] R. Featherstone and D. Orin, "Chapter 2: Dynamics," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 35–65, New York: Springer, 2008.
[19] R. Featherstone, "A beginner's guide to 6-d vectors (part 1)," *IEEE Robotics Automation Magazine*, vol. 17, pp. 83–94, Sept. 2010.
[20] R. Featherstone, "A beginner's guide to 6-d vectors (part 2)," *IEEE Robotics Automation Magazine*, vol. 17, pp. 88–99, Dec. 2010.
[21] R. E. Roberson and R. Schwertassek, *Dynamics of Multibody Systems*. Berlin/Heidelberg/New York: Springer-Verlag, 1988.
[22] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2008.
[23] S. McMillan, D. Orin, and R. McGhee, "Efficient dynamic simulation of an underwater vehicle with a robotic manipulator," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 25, pp. 1194–1206, Aug. 1995.