

# Recursive Rigid-Body Dynamics Algorithms for Systems with Kinematic Loops

Matthew Chignoli<sup>1</sup>, Nicholas Adrian<sup>2</sup>, Sangbae Kim<sup>1</sup>, Patrick M. Wensing<sup>2</sup>

**Abstract**—We propose a novel approach for generalizing the following rigid-body dynamics algorithms: Recursive Newton-Euler Algorithm, Articulated-Body Algorithm, and Extended-Force-Propagator Algorithm. The classic versions of these recursive algorithms require systems to have an open chain structure. Dealing with closed-chains has, conventionally, required different algorithms. In this paper, we demonstrate that the classic recursive algorithms can be modified to work for closed-chain mechanisms. The critical insight of our generalized algorithms is the clustering of bodies involved in local loop constraints. Clustering bodies enables loop constraints to be resolved locally, i.e., only when that group of bodies is encountered during a forward or backward pass. This local treatment avoids the need for large-scale matrix factorization. We provide self-contained derivations of the algorithms using familiar, physically meaningful concepts. Overall, our approach provides a foundation for simulating robotic systems with traditionally difficult-to-simulate designs, such as geared motors, differential drives, and four-bar mechanisms. The performance of our library of algorithms is validated numerically in C++ on various modern legged robots: the MIT Mini Cheetah, the MIT Humanoid, the UIUC Tello Humanoid, and a modified version of the JVRC-1 Humanoid. Our algorithms are shown to outperform state-of-the-art algorithms for computing constrained rigid-body dynamics.

**Index Terms**—Dynamics, Recursive Algorithms, Legged Robots, Humanoid Robots

## I. INTRODUCTION

In recent years, the field of legged robots has been dominated by a trend of complex limb designs that aim to expand the robot’s range of motion, minimize the inertia of the limbs, and smoothly handle impacts with the environment. Designs with sub-mechanisms such as parallel belt transmissions [1] (Fig. 1a), differential drives [2], [3] (Fig. 1b), and linkages [4], [5] have emerged as popular options to meet these design criteria. Unfortunately, the dynamics algorithms used to simulate these robotic systems have not kept pace with these recent hardware advancements. Fast, accurate algorithms are crucially important for both model-based control and “model-free” reinforcement learning, which greatly depend on the quality and speed of the underlying simulator. While the Articulated-Body Algorithm (ABA) [6] has traditionally served the role of a fast, accurate algorithm for simulating open-chain robotic systems, it is not suitable for systems with actuation sub-mechanisms like those mentioned above due to

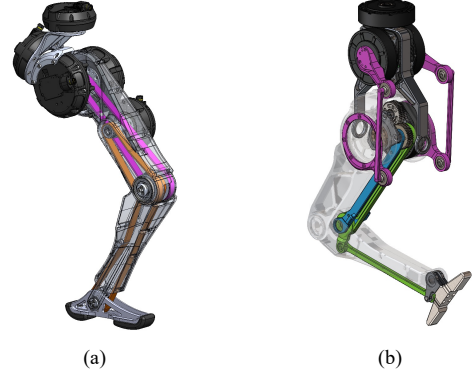


Fig. 1. (a) A parallel belt transmission that actuates the knee (pink) and ankle (orange) of the MIT Humanoid’s leg [1]. (b) A pair of differential drives that actuate the hip (pink) as well as the knee and ankle (blue and green) of the Tello Humanoid’s leg [2].

the presence of kinematic loops. To that end, we present a suite of rigid-body dynamics algorithms that is capable of accounting for the constraints imposed by these kinematic loops while maintaining the favorable recursive structure of state-of-the-art algorithms used for dynamic simulation and optimal control such as the ABA for forward dynamics, Recursive Newton-Euler Algorithm (RNEA) for inverse dynamics, and Extended-Force-Propagator Algorithm (EFPA) for computing the inverse Operational-Space Inertia Matrix (OSIM).

### A. Related Work

The RNEA and ABA were developed in the 1970’s by Orin [7] and Vereshchagin [8], respectively. These algorithms represented a significant improvement relative to the algorithms of their time because they were the first to demonstrate linear complexity in the number of joints of the robot. Vereshchagin’s algorithm, however, did not experience widespread adoption until a decade later, when Featherstone independently developed an operationally identical recursive forward dynamics algorithm [6]. Vereshchagin developed his algorithm by applying Dynamic Programming (DP) [9] to the optimization problem that arises from the Gauss Principle of Least Constraint (GPLC) [10]. The derivation can be intuitive for many roboticists because it requires reasoning about the GPLC, which considers how the system would move in the presence versus the absence of constraint forces. A direct connection is also possible between the optimal Gauss cost to go for this optimization problem and Featherstone’s “articulated-body inertias” for the system. A key advantage

<sup>1</sup> are with the Department of Mechanical Engineering, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA {chignoli, sangbae}@mit.edu

<sup>2</sup> are with the Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN, USA {nadrian, pwensing}@nd.edu

of Featherstone’s version of the algorithm, on the other hand, is that it has been developed in the context of Spatial Vector Algebra (SVA) [11], which enables concise descriptions of complex dynamic relationships through the use of physically interpretable quantities. Furthermore, SVA provides a general set of operations that have been employed to develop other recursive algorithms for open chains [12]–[15]. Many of the same recursive algorithms have alternatively been developed using Spatial Operator Algebra (SOA) [16], [17], which, despite its similar name, is different from SVA and relies upon sparse matrix factorization identities that can be more difficult to comprehend and extend.

Recently, efforts have been devoted to developing efficient recursive algorithms for computing the inverse OSIM. The inverse OSIM plays a crucial role in contact-rich dynamic simulation, so computing it efficiently can lead to significantly faster simulation capabilities. The lowest complexity methods are the EFPA [13] and the PV-OSIMr [18]. The efficiency of the EFPA results from the computation and use of matrices that propagate a force across several links in a single operation. The PV-OSIMr, the lowest complexity method, is made more efficient than the EFPA by carefully computing (and combining) these propagators between branching points in the mechanism. In practice, though, many simulators opt to use sparsity exploiting matrix factorization [19] to compute the inverse OSIM because, despite its worse theoretical complexity, it has been empirically shown to be competitive to previous recursive methods [20].

All of the algorithms discussed so far are only applicable to open chains. Efforts have also been made to generalize rigid-body dynamics algorithms to capture the effects of loop constraints. Recognizing the importance of including the complete effects of gear ratios and the gyroscopic effects of the spinning motors, researchers working with high gear-ratio manipulators developed minimally modified versions of classic algorithms such as the RNEA [21], [22], Coriolis matrix factorization algorithm [23], and ABA [24], [25] so that they could handle this particular class of sub-mechanisms. The subsequent development of General Assembly Algorithms such as the Divide-and-Conquer Algorithm [26], [27] and the Assembly-Disassembly Algorithm [28] used the concepts of multi-handle articulated bodies and inverse articulated inertia to compute the forward dynamics of systems with a larger class of loop constraints. While these algorithms excel when parallel computing is available, they are not optimized for serial performance in the way that ABA and the forward dynamics algorithm proposed in this work are. Using a GPLC-based approach similar to the one proposed in this work, [29] develops a family of constrained dynamics algorithms. Their method is specialized to handle endpoint constraints, which are different than the “internal” loop constraints such as those resulting from actuation sub-mechanisms as considered herein. Efficient non-recursive algorithms for constrained forward dynamics of systems with arbitrary closure constraints were also developed. These algorithms use techniques such as sparse matrix factorization [19] to efficiently solve the large systems of linear equations arising from the equations of motion and closure constraints. A recursive counterpart to

these methods was proposed by Bradl et al., which considers Lagrange multiplier propagation/elimination [30] to address loop constraints sequentially rather than for the system as a whole.

A popular alternative approach to using constrained dynamics algorithms, especially in dynamic simulation for reinforcement learning, is to approximate or ignore these effects. MuJoCo, for example, adds a diagonal “armature matrix” to the joint-space inertia matrix that allows them to approximate the effects of geared motors [20] while still using efficient recursive dynamics algorithms. Techniques such as domain randomization [31] can make the learned policies robust to such modeling errors, but typically at the cost of sample complexity and the optimality of the policy.

A critical development in constrained forward dynamics was manipulating systems such that, even in the presence of kinematic loops, they would be amenable to the conventional recursive algorithms for open chains. The first implementation of this idea was called Recursive Coordinate Reduction (RCR) [32] and involved introducing “phantom bodies” into the system to sever the closed loops artificially. Jain’s development of Local Constraint Embedding (LCE) [33] achieved the same effect without introducing phantom bodies. Instead, LCE involves grouping bodies involved in a kinematic loop into an “aggregate link” and then applying conventional algorithms to the open chain that consists of both conventional and aggregate links. Until recently, however, RCR and LCE have been largely ignored by roboticists. Recent use cases for LCE have focused on applications related to parallel kinematic mechanisms [34], with an emphasis on how hybrid numerical and analytical approaches to dealing with the constraints can lead to improved computational efficiency and accuracy [35], [36].

## B. Contribution

This paper aims to present a library of rigid-body dynamics algorithms that (i) lays the foundation for a new dynamics engine capable of accurate and efficient simulation of robots with actuation sub-mechanisms and (ii) can be used in model-based control applications to reduce modeling errors without significantly increasing solve time. To that end, this work offers the following contributions:

1) *Development of Spatial Vector Algebra for clusters of bodies:* We generalize conventional dynamics algorithms for open chains to systems with kinematic loops by clustering the groups of bodies involved in the loops. While our proposed treatment of kinematic loops is functionally similar to Jain’s LCE [33], our alternative SVA-based development of “body clusters” offers physical interpretability that lends to its integration into other algorithms (e.g., RNEA and EFPA), as detailed for the first time herein.

2) *Physically-grounded derivation of a generalized ABA for systems with loop constraints:* We synthesize the concept of SVA for clusters with Vereshchagin’s DP-based derivation of the ABA to develop a recursive algorithm for computing the forward dynamics of a rigid-body system with loop constraints, the Cluster-Based Articulated-Body Algorithm (C-ABA). The final recursive computations are the same as

the ones in [33]. Still, the derivation is decidedly different, emphasizing generalizing the concept of an articulated body to the case of body clusters, which should increase accessibility for roboticists.

3) *Generalized versions of the RNEA and EFPA for systems with loop constraints*: In addition to the C-ABA, we use the technique of clustering bodies to develop novel extensions of the RNEA and EFPA for systems with loop constraints: the Cluster-Based Recursive Newton-Euler Algorithm (C-RNEA) and the Cluster-Based Extended-Force-Propagator Algorithm (C-EFPA). The derivations of all three algorithms are unified under a common notation and shared use of cluster-based SVA quantities that are given a physical meaning herein.

We provide an open-source C++ implementation of the library of algorithms at [https://github.com/ROAM-Lab-ND/generalized\\_rbda](https://github.com/ROAM-Lab-ND/generalized_rbda). To demonstrate the capabilities of our library, we benchmark our methods against state-of-the-art algorithms applied to a variety of modern legged robots with actuation mechanisms: MIT Mini Cheetah [37], MIT Humanoid [1], and UIUC Tello Humanoid [2], and a modified version of the JVRC-1 [38]. Across all platforms, the robots include actuation mechanisms that are traditionally not handled via purely recursive methods.

The rest of this paper will be organized as follows. Section II provides a summary of existing rigid-body algorithms used for dynamic simulation, with a background on SVA provided in Section III. Section IV explains how bodies in kinematic loops are “clustered” to make arbitrary rigid-body systems amenable to recursive algorithms. Section V develops the generalized version of SVA for clusters, with an emphasis on how it differs compared to SVA for individual bodies. Sections VI, VII, and VIII derive the C-RNEA, C-ABA, and C-EFPA, respectively. Section IX compares the performance of our proposed algorithms against state-of-the-art algorithms. Section X discusses the broader impacts of the work, and Section XI summarizes the work.

## II. BACKGROUND

This section first describes a methodology for modeling systems with such loop constraints, following the conventions in [12]. The remainder of the section summarizes existing rigid-body dynamics algorithms that are valid for such models. A summary of the notation used throughout the rest of the paper is provided in Table I.

### A. Modeling

A popular approach to modeling rigid-body systems is to describe the system in terms of its component parts via a *system model* [12]. A system model consists solely of bodies and the kinematic relationships between them (i.e., joints) and can be expressed as a Rigid-Body Connectivity Graph (RB-CG). The RB-CG for a system has the following properties [12]:

- Its nodes represent bodies.
- Its arcs represent joints.
- Exactly one node represents a fixed base.
- The graph is undirected.

TABLE I  
NOTATION

Symbol	Definition
$\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$	Spanning tree joints positions, velocities, and accelerations
$\tau$	Spanning tree generalized forces
$\phi(\cdot), \mathbf{K}, \mathbf{k}$	Implicit loop constraint, Jacobian, and bias
$\gamma(\cdot), \mathbf{G}, \mathbf{g}$	Explicit loop constraint, Jacobian, and bias
$\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}$	Independent joint positions, velocities, and accelerations
$\tau_{\text{ind}}$	Independent generalized forces
$\mathbf{H}$	Joint-space inertia matrix
$\mathbf{c}$	Joint-space bias force
$\Omega$	Inverse operational-space inertia matrix
$\mathbf{f}_e$	Spatial force acting at end-effector $e$
<b>Body / Cluster</b>	
$\mathcal{G} / \mathcal{G}_C$	Rigid-Body / Cluster connectivity graph
$N_B / N_C$	Number of nodes in $\mathcal{G} / \mathcal{G}_C$
$\mathbf{q}_i, \dot{\mathbf{q}}_i, \ddot{\mathbf{q}}_i / \mathbf{q}_k, \dot{\mathbf{q}}_k, \ddot{\mathbf{q}}_k$	Portion of $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ associated with joint $i$ / cluster joint $k$
$\tau_i / \tau_{j_k}$	Portion of $\tau$ associated with joint $i$ / cluster joint $k$
$\mathbf{y}_i, \dot{\mathbf{y}}_i, \ddot{\mathbf{y}}_i / \mathbf{y}_k, \dot{\mathbf{y}}_k, \ddot{\mathbf{y}}_k$	Portion of $\mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}$ associated with joint $i$ / cluster joint $k$
$\tau_k$	Portion of $\tau_{\text{ind}}$ associated with cluster joint $k$
$\mathbf{G}_k, \mathbf{g}_k$	Explicit constraint Jacobian, bias associated with cluster joint $k$
$\mathbf{v}_i / \mathbf{v}_k$	Spatial velocity of body $i$ / cluster $k$
$\mathbf{a}_i / \mathbf{a}_k$	Spatial acceleration of body $i$ / cluster $k$
$\mathbf{f}_i / \mathbf{f}_k$	Spatial force of body $i$ / cluster $k$
$\mathbf{I}_i / \mathbf{I}_k$	Spatial inertia of body $i$ / cluster $k$
$\mathbf{S}_i / \mathbf{S}_k$	Motion Subspace Matrix of joint $i$ / cluster joint $k$
$\mathbf{T}_i^a / \mathbf{T}_k^a$	Active Force Subspace Matrix of joint $i$ / cluster joint $k$
$\mathbf{T}_i^c / \mathbf{T}_k^c$	Constraint Force Subspace Matrix of joint $i$ / cluster joint $k$
$\mathbf{f}_i^j / \mathbf{f}_k^j$	Force exchanged across joint $i$ / cluster joint $k$
${}^j\mathbf{X}_i / {}^j\mathbf{X}_k$	Spatial motion transform between bodies $i$ and $j$ / clusters $k$ and $\ell$
${}^j\mathbf{X}_i^* / {}^j\mathbf{X}_k^*$	Spatial force transform between bodies $i$ and $j$ / clusters $k$ and $\ell$
$\Lambda^{(k)}\mathbf{X}_{\lambda(k)}$	Spatial motion transform within cluster $k$
$\mathbf{I}_i^A / \mathbf{I}_k^A$	Articulated inertia for body $i$ / cluster $k$
$\mathbf{p}_i^A / \mathbf{p}_k^A$	Articulated bias force for body $i$ / cluster $k$
$V_k$	Gauss cost to go for cluster $k$
${}^e\mathbf{X}_i / {}^e\mathbf{X}_k$	Extended force propagator from body $i$ / cluster $k$ to end-effector $e$

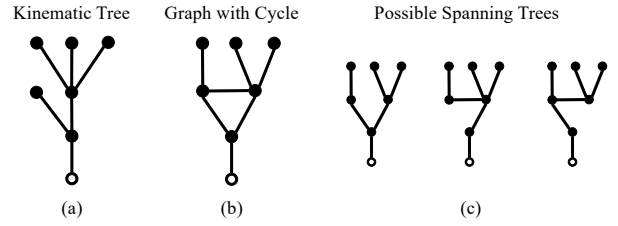


Fig. 2. (a) Example of a RB-CG that is kinematic tree. (b) Example of a RB-CG that has a cycle and is therefore not a kinematic tree. (c) Every possible spanning tree for the example in (b).

- The graph is connected.

A RB-CG is a topological tree if there exists only one path between any two nodes in the graph (i.e., no cycles). If the RB-CG of a system is a topological tree, then the system is referred to as a kinematic tree. For a RB-CG  $\mathcal{G}$ , a spanning tree  $\mathcal{G}_t$  is a subgraph of  $\mathcal{G}$  that contains all nodes in  $\mathcal{G}$  along with a subset of arcs such that  $\mathcal{G}_t$  is a topological tree. The relationship between connectivity graphs, kinematic trees, and spanning trees is visualized in Fig. 2. To describe the connectivity of a spanning tree, we use the following definitions from [12]:

- $\lambda(B_i)$ : the parent body of body  $i$  (toward the root),
- $\kappa(B_i)$ : the set of bodies supporting body  $i$ , defined by

$$\kappa(B_i) = B_i \cup \kappa(\lambda(B_i)), \text{ where } \kappa(B_0) = \emptyset,$$

- $\mu(B_i)$ : the set of children of body  $i$ , defined by

$$\mu(B_i) = \{B_j : \lambda(B_j) = B_i\},$$

- $\nu(B_i)$ : the subtree of bodies starting at body  $i$ .

$$\nu(B_i) = \{B_j : B_i \in \kappa(B_j)\}$$

Every body  $B_i$  in the spanning tree, except the fixed base, is connected to its parent via the tree joint  $J_i$ . Bodies with no children, i.e., bodies where  $\mu(B_i)$  is empty, are referred to as “leaves” of the tree. The leftover joints - the arcs that are in  $\mathcal{G}$  but not  $\mathcal{G}_t$  - are referred to as loop joints. Conventional recursive algorithms for rigid-body dynamics are only compatible with kinematic trees. The equations of motion for such a system are given by

$$\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau}, \quad (1)$$

where  $\mathbf{q} \in \mathbb{R}^n$  is the set of spanning tree coordinates,  $\mathbf{H} \in \mathbb{R}^{n \times n}$  is the mass matrix,  $\mathbf{c} \in \mathbb{R}^n$  the bias force, and  $\boldsymbol{\tau} \in \mathbb{R}^n$  is the set of generalized forces applied at each joint. Beyond (1), systems with loop joints must also consider loop constraint forces and satisfy the kinematic constraints associated with those loop joints.

### B. Loop Constraints

An independent kinematic loop of an RB-CG is a cycle that traverses one loop joint. The number of independent kinematic loops always equals the number of loop joints [12]. Loop joints encode kinematic constraints between all the bodies involved in an independent kinematic loop. These constraints can be expressed in either “implicit” form [12],

$$\phi(\mathbf{q}) = 0, \quad \mathbf{K}(\mathbf{q})\dot{\mathbf{q}} = 0, \quad \mathbf{K}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{k}(\mathbf{q}, \dot{\mathbf{q}}), \quad (2)$$

or, in cases where an independent set of coordinates exists, explicit form [12]

$$\mathbf{q} = \boldsymbol{\gamma}(\mathbf{y}), \quad \dot{\mathbf{q}} = \mathbf{G}(\mathbf{q})\dot{\mathbf{y}}, \quad \ddot{\mathbf{q}} = \mathbf{G}(\mathbf{q})\ddot{\mathbf{y}} + \mathbf{g}(\mathbf{q}, \dot{\mathbf{y}}), \quad (3)$$

where  $\mathbf{q} \in \mathbb{R}^n$  is the set of complete (i.e., spanning) coordinates of the robot and  $\mathbf{y} \in \mathbb{R}^m$  ( $m \leq n$ ) is the set of independent coordinates. The constraint Jacobians and biases for the implicit and explicit constraints are given by

$$\mathbf{K} = \frac{\partial \phi(\mathbf{q})}{\partial \mathbf{q}}, \quad \mathbf{k} = -\dot{\mathbf{K}}\dot{\mathbf{q}}, \quad \mathbf{G} = \frac{\partial \boldsymbol{\gamma}(\mathbf{y})}{\partial \mathbf{y}}, \quad \mathbf{g} = \dot{\mathbf{G}}\dot{\mathbf{y}}.$$

For ease of exposition, we treat all constraints in this work as explicit. However, as we discuss later in Sec. VII, our final algorithm does not rely on a definition of  $\boldsymbol{\gamma}(\mathbf{y})$ , and so only depends on the adoption of independent velocities, making it immediately compatible with implicit constraints.

The existence of independent coordinates leads to the definition of a new subgraph:  $\mathcal{G}_{\text{ind}}$ . The independent tree  $\mathcal{G}_{\text{ind}}$  is the subgraph of  $\mathcal{G}$  that contains all independent tree joints (i.e., tree joints whose state is included in  $\mathbf{y}, \dot{\mathbf{y}}$ ) and their corresponding bodies. Thus, there are typically bodies in  $\mathcal{G}$  that are not present in  $\mathcal{G}_{\text{ind}}$ . For example, consider the systems shown in Fig. 3, which depicts the RB-CGs and corresponding independent trees for two systems: a serial chain of links actuated by a geared transmission and a four-bar mechanism. All constraints on the serial chain can be written down explicitly. The position and velocity of every rotor joint are equal to the position and velocity of its respective link joint scaled by its

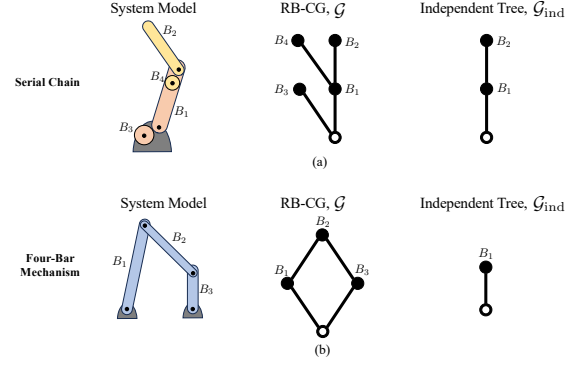


Fig. 3. Connectivity graphs and corresponding independent trees for (a) a serial chain of links actuated by geared transmissions and (b) a four-bar mechanism.

gear ratio. Thus, the set of independent coordinates can be the link joint states. The four-bar mechanism, however, is more naturally described via an implicit constraint to determine valid joint positions. However, given the position of every joint and the velocity  $\dot{\mathbf{y}}$  of a single joint, the velocities of the rest of the joints can be determined via  $\dot{\mathbf{q}} = \mathbf{G}(\mathbf{q})\dot{\mathbf{y}}$  since the mechanism has a single degree of freedom. Therefore, the four-bar mechanism also has a set of independent joint velocities.

### C. Inverse Dynamics Algorithms

Inverse dynamics involves finding the generalized forces  $\boldsymbol{\tau}$  that will produce a given acceleration  $\ddot{\mathbf{q}}$  when the robot is in configuration  $(\mathbf{q}, \dot{\mathbf{q}})$ . When the system is a kinematic tree, the most efficient algorithm for computing the inverse dynamics is the RNEA [7], which has computational complexity of  $\mathcal{O}(n)$ , where  $n$  is the number of bodies in the graph. However, two common alternatives exist when loop joints are present: Approximate RNEA, which deals with the independent tree, and Projected RNEA, which deals with the spanning tree.

Approximate RNEA applies conventional RNEA to the independent tree and then accounts for the effects of loop joints and dependent bodies via a constant offset to the mass matrix,

$$\boldsymbol{\tau}_{\text{ind}} \approx \text{RNEA}(\mathcal{G}_{\text{ind}}, \mathbf{y}, \dot{\mathbf{y}}, \ddot{\mathbf{y}}) + \tilde{\mathbf{H}}\ddot{\mathbf{y}},$$

where  $\tilde{\mathbf{H}}$  represents that constant offset. For example, considering the systems in Fig. 3, the following approximation is commonly used for geared transmissions:

$$\tilde{\mathbf{H}} = \begin{bmatrix} N_1^2 I_1^{\text{rot}} & 0 \\ 0 & N_2^2 I_2^{\text{rot}} \end{bmatrix},$$

where  $N_i$  is the gear ratio of the  $i$ -th transmission and  $I_i^{\text{rot}}$  is the rotational inertia of the  $i$ -th rotor. However, in the case of the four-bar mechanism, no straightforward approximation exists, so if one were to ignore the rest of the transmission elements, this would correspond to  $\tilde{\mathbf{H}} = 0$ . We refer to this approach as Unconstrained RNEA.

Projected RNEA [12], which does not rely on approximation, uses the explicit constraint Jacobian to project the

spanning joint torques back to a set of joint torques applied at the independent joints

$$\begin{aligned}\boldsymbol{\tau} &= \text{RNEA}(\mathcal{G}_t, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}), \\ \boldsymbol{\tau}_{\text{ind}} &= \mathbf{G}^\top \boldsymbol{\tau}.\end{aligned}$$

While the Approximate RNEA is appealing for its favorable computational complexity, in cases where accuracy is critical or where the dynamic effects of loop constraints cannot be easily approximated, Projected RNEA must be used. While it is exact, Projected RNEA is slower than Approximate RNEA because (i) it loops over all spanning tree joints rather than just the independent joints and (ii) the potentially large matrix multiplication involving  $\mathbf{G}^\top$ , which, in the worst case, can make the method  $\mathcal{O}(n^2)$ .

#### D. Forward Dynamics Algorithms

Forward dynamics involves finding the set of accelerations  $\ddot{\mathbf{q}}$  that results from generalized forces  $\boldsymbol{\tau}$  being applied to the robot when it is in state  $(\mathbf{q}, \dot{\mathbf{q}})$ . When the system is a kinematic tree, the most efficient algorithm for computing the forward dynamics is the ABA [6], which also has computational complexity  $\mathcal{O}(n)$ . Three common alternatives exist for computing constrained forward dynamics: Approximate ABA, which uses the independent tree, and the Projection and Lagrange Multiplier Methods, which use the spanning tree.

Similar to Approximate RNEA, Approximate ABA applies conventional ABA to the independent tree and then uses an approximate mass matrix to account for the effects of loop joint and dependent bodies. It proceeds by efficiently solving

$$\ddot{\mathbf{y}} = (\mathbf{H}_{\text{ind}} + \tilde{\mathbf{H}})^{-1} (\boldsymbol{\tau}_{\text{ind}} - \mathbf{c}_{\text{ind}}),$$

where  $\mathbf{H}_{\text{ind}}$  and  $\mathbf{c}_{\text{ind}}$  are the mass matrix and bias force for the independent tree.

The Projection method uses the explicit constraint Jacobian  $\mathbf{G}$  to project the equations of motion for the spanning tree coordinates onto the independent coordinates via

$$\ddot{\mathbf{y}} = (\mathbf{G}^\top \mathbf{H} \mathbf{G})^{-1} \mathbf{G}^\top (\boldsymbol{\tau} - \mathbf{c} - \mathbf{H} \mathbf{g}).$$

This method requires factorizing a square matrix with size equal to the number of independent coordinates. While this is theoretically slow ( $\mathcal{O}(n^3)$ ), high-performance linear algebra libraries such as Eigen [39] make it competitive, especially for robots with relatively few DOFs and/or favorable sparsity patterns in  $\mathbf{G}$ .

The Lagrange Multiplier Method simultaneously solves for the constrained acceleration  $\ddot{\mathbf{q}}$  and constraint forces  $\boldsymbol{\lambda}^c$  by solving the linear system

$$\begin{bmatrix} \mathbf{H} & \mathbf{K}^\top \\ \mathbf{K} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ -\boldsymbol{\lambda}^c \end{bmatrix} = \begin{bmatrix} \boldsymbol{\tau} - \mathbf{c} \\ \mathbf{k} \end{bmatrix}.$$

Sparsity-exploiting methods can speed this up [40], but the method still involves increasing the size of the problem by introducing the Lagrange Multipliers.

#### E. Inverse Operational-Space Inertia Matrix Algorithms

For a rigid-body system with end-effectors, the operational-space dynamics [41] describe the motion of the system from the perspective of the end-effectors rather than from the perspective of the joints as in (1). Thus, for a system whose operational-space velocities are given by

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}},$$

the operational-space equations of motion are given by

$$\ddot{\mathbf{x}} = \mathbf{J} \mathbf{H}^{-1} \boldsymbol{\tau} - \boldsymbol{\Omega} \mathbf{F} + \boldsymbol{\beta}, \quad (4)$$

where

$$\boldsymbol{\beta}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{J}} \dot{\mathbf{q}} - \mathbf{J} \mathbf{H}^{-1} \mathbf{c}, \quad \boldsymbol{\Omega}(\mathbf{q}) = \mathbf{J} \mathbf{H}^{-1} \mathbf{J}^\top,$$

and  $\mathbf{F}$  is a collection of forces applied at the respective end-effectors.

The EFPA is a reduced-order algorithm for computing the inverse operational-space inertia matrix,  $\boldsymbol{\Omega}$ , for kinematic trees. Despite their importance in robotics applications, operational space algorithms for systems with loop constraints have not received as much focus as inverse and forward dynamics algorithms. Thus, for comparison, we consider two approaches for computing the inverse operational-space inertia matrix for a system with loop constraints: the Approximate EFPA and the Projected EFPA. Similar to the algorithms in Sections II-C and II-D, the Approximate EFPA works by efficiently computing

$$\boldsymbol{\Omega} \approx \mathbf{J}_{\text{ind}} (\mathbf{H}_{\text{ind}} + \tilde{\mathbf{H}})^{-1} \mathbf{J}_{\text{ind}}^\top,$$

where  $\mathbf{J}_{\text{ind}}$  is operational-space Jacobian for the independent tree. The Projection method, which considers the full spanning tree, uses the constraint Jacobian to compute

$$\boldsymbol{\Omega} = \mathbf{J} (\mathbf{G}^\top \mathbf{H} \mathbf{G})^{-1} \mathbf{J}^\top.$$

### III. SPATIAL VECTOR ALGEBRA REVIEW

SVA is a mathematical framework widely used in the analysis and control of robotic systems [11]. The framework provides a concise and elegant way of representing and manipulating the kinematics and dynamics of rigid bodies in three-dimensional space. In this section, we review the basic concepts of conventional SVA for rigid bodies.

#### A. Spatial Motion and Spatial Force Vectors

The foundational pieces of SVA are the 6D spatial motion and spatial force vectors. Spatial motion vectors combine the angular and linear components of motion via

$${}^O \mathbf{v} = \begin{bmatrix} {}^O \boldsymbol{\omega} \\ {}^O \mathbf{v}_O \end{bmatrix},$$

where the superscript  $O$  denotes a quantity expressed in the coordinate frame  $O$ ,  ${}^O \mathbf{v}$  is the spatial motion,  ${}^O \boldsymbol{\omega} \in \mathbb{R}^3$  is the angular velocity of the body, and  ${}^O \mathbf{v}_O \in \mathbb{R}^3$  is the linear velocity of a body-fixed point instantaneously coincident with

the origin of frame  $O$ . Similarly, spatial force vectors combine the linear and angular components of force interactions via

$${}^O\mathbf{f} = \begin{bmatrix} {}^O\mathbf{n}_O \\ {}^O\mathbf{f} \end{bmatrix},$$

where  ${}^O\mathbf{f}$  is the spatial force,  ${}^O\mathbf{n}_O \in \mathbb{R}^3$  is the net moment about the origin of frame  $O$ , and  ${}^O\mathbf{f} \in \mathbb{R}^3$  is the net linear force on the body.

### B. Spatial Transforms

The frame of expression for spatial vectors can be changed through spatial transforms. Spatial transforms depend on the relative orientation and position of the coordinate frames that they transform between. The spatial motion transform  ${}^P\mathbf{X}_O$  provides the coordinate transform  ${}^P\mathbf{v} = {}^P\mathbf{X}_O {}^O\mathbf{v}$ , while the spatial force transform  ${}^P\mathbf{X}_O^*$  provides the coordinate transform  ${}^P\mathbf{f} = {}^P\mathbf{X}_O^* {}^O\mathbf{f}$ . However, these transforms are related via the following identities:

$${}^O\mathbf{X}_P = ({}^P\mathbf{X}_O)^{-1}, \quad {}^O\mathbf{X}_P^* = ({}^P\mathbf{X}_O^*)^{-1}, \quad {}^O\mathbf{X}_P^\top = {}^P\mathbf{X}_O^*.$$

### C. Spatial Cross Products

The derivatives of spatial vectors can be computed using the spatial cross-product operator. Consider spatial motion and force vectors  ${}^P\mathbf{m}$  and  ${}^P\mathbf{f}$ , respectively, expressed in a reference frame that is moving with spatial velocity  ${}^P\mathbf{v}$ . Following the notation from [12] where, for a fixed frame  $O$ ,

$${}^P\dot{\mathbf{m}} = {}^P\left(\frac{d}{dt}\mathbf{m}\right) = {}^P\mathbf{X}_O\left(\frac{d}{dt}{}^O\mathbf{m}\right),$$

the derivatives of  ${}^P\mathbf{m}$  and  ${}^P\mathbf{f}$  are given by

$${}^P\dot{\mathbf{m}} = \frac{d}{dt}{}^P\mathbf{m} + {}^P\mathbf{v} \times {}^P\mathbf{m}, \quad {}^P\dot{\mathbf{f}} = \frac{d}{dt}{}^P\mathbf{f} + {}^P\mathbf{v} \times^* {}^P\mathbf{f}$$

where the spatial “cross products” are defined

$$\mathbf{v} \times = \begin{bmatrix} \boldsymbol{\omega} \times & \mathbf{0}_{3 \times 3} \\ \mathbf{v} \times & \boldsymbol{\omega} \times \end{bmatrix}, \quad \mathbf{v} \times^* = \begin{bmatrix} \boldsymbol{\omega} \times & \mathbf{v} \times \\ \mathbf{0}_{3 \times 3} & \boldsymbol{\omega} \times \end{bmatrix}. \quad (5)$$

### D. Spatial Inertia

The spatial inertia  $\mathbf{I} \in \mathbb{R}^{6 \times 6}$  of a rigid body relates its spatial velocity to its spatial momentum  $\mathbf{h}$  via

$${}^P\mathbf{h} = \begin{bmatrix} \bar{\mathbf{I}}_C + m\mathbf{r}_C \times \mathbf{r}_C \times^\top & m\mathbf{r}_C \times \\ m\mathbf{r}_C \times^\top & m\mathbf{1}_{3 \times 3} \end{bmatrix} \begin{bmatrix} {}^P\boldsymbol{\omega} \\ {}^P\mathbf{v}_P \end{bmatrix} = {}^P\mathbf{I} {}^P\mathbf{v},$$

where  $\bar{\mathbf{I}}_C \in \mathbb{R}^{3 \times 3}$  is the rotational inertia about the body’s center of mass, and  $m \in \mathbb{R}$  is the mass of the body, and  $\mathbf{r}_C \in \mathbb{R}^3$  is the vector from the origin of frame  $P$  to the center of mass. The rate of change of the spatial momentum is equal to the net spatial force acting on the body, which leads to the generic spatial equation of motion

$${}^P\mathbf{f}^{\text{net}} = {}^P\left(\frac{d}{dt}(\mathbf{I}\mathbf{v})\right) = {}^P\mathbf{I} {}^P\mathbf{a} + {}^P\mathbf{v} \times^* {}^P\mathbf{I} {}^P\mathbf{v}, \quad (6)$$

where  $P$  is an arbitrary frame that can be moving or fixed and  $\mathbf{a} \in \mathbb{R}^6$  is the spatial acceleration of the body.

### E. Motion and Force Subspace Matrices

Conventional motion subspace matrices encode the relative motion between two bodies permitted by a tree joint. For example, if body  $i$  is in a rigid-body system connected to its parent  $\lambda(i)$  via tree joint  $J_i$ , then the velocity of body  $i$  is given by

$$\mathbf{v}_i = {}^i\mathbf{X}_{\lambda(i)}\mathbf{v}_{\lambda(i)} + {}^i\mathbf{S}_i(\mathbf{q}_i)\dot{\mathbf{q}}_i,$$

where  $\mathbf{q}_i$ ,  $\dot{\mathbf{q}}_i$ , and  $\mathbf{S}_i$  are the joint position, joint velocity, and motion subspace matrix [12] for  $J_i$ , respectively. We drop the configuration dependence of  ${}^i\mathbf{S}_i$  for brevity while also noting that for the common tree joints,  ${}^i\mathbf{S}_i$  is constant. When working with quantities associated with a body (e.g.,  $\mathbf{v}_i$  and  $\mathbf{v}_{\lambda(i)}$ ), unless otherwise noted, we will hereafter assume they are expressed in their body-local frame.

Force subspace matrices encode the spatial forces that can be transmitted between bodies across a tree joint. A free body analysis of body  $i$  shows that the spatial force transmitted across  $J_i$  is equal to,

$$\mathbf{f}_i^J = \mathbf{f}_i^{\text{net}} + \sum_{B_j \in \mu(B_i)} \mathbf{f}_j^J. \quad (7)$$

The force  $\mathbf{f}_i^J$  represents the force needed to move the subtree of bodies starting at body  $i$ . The force subspace matrices  $\mathbf{T}_i^a$  and  $\mathbf{T}_i^c$  are defined such that  $\mathbf{f}_i^J$  can be decomposed into

$$\mathbf{f}_i^J = \mathbf{T}_i^a \boldsymbol{\tau}_i + \mathbf{T}_i^c \boldsymbol{\lambda}_i^c, \quad (8)$$

where  $\boldsymbol{\tau}_i$  is the generalized force,  $\boldsymbol{\lambda}_i^c$  is the constraint force, and  $\mathbf{T}_i^a$  and  $\mathbf{T}_i^c$  are the active and constraint force subspace matrices, respectively. By definition, motion and force subspace matrices must always satisfy:

$$\mathbf{S}^\top \mathbf{T}^a = \mathbf{1}, \quad \mathbf{S}^\top \mathbf{T}^c = \mathbf{0} \quad (9)$$

where  $\mathbf{1}$  and  $\mathbf{0}$  are the appropriately sized identity and zero matrices, respectively. Therefore, left multiplying (8) by  $\mathbf{S}^\top$  leads to the following key relationship between a tree joint’s generalized force and its transmitted force

$$\boldsymbol{\tau}_i = \mathbf{S}_i^\top \mathbf{f}_i^J. \quad (10)$$

In subsequent sections, we will introduce generalized versions of these conventional subspace matrices. The key distinction of the generalized subspace matrices is that they account for the effects of loop constraints.

## IV. CONSTRAINT EMBEDDING VIA CLUSTERING

Conventional recursive algorithms for rigid-body dynamics are only compatible with kinematic trees. In this section, we detail how kinematic trees can be created out of arbitrary rigid-body systems by clustering bodies involved in kinematic loops. To that end, we introduce our cluster-based connectivity graph representation for systems with loop-closure constraints.



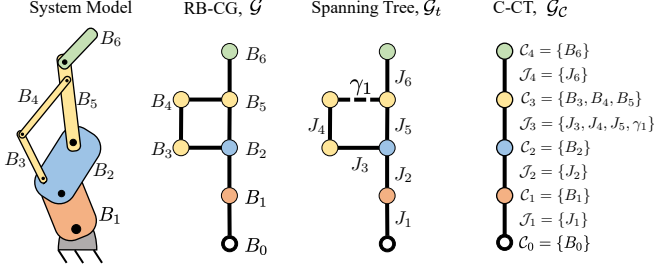


Fig. 4. Different representations for the connectivity of a system model.

### A. Cluster Connectivity Graphs

The technique of clustering can be used to convert arbitrary RB-CGs to kinematic trees where each node in the tree represents a *collection of rigid bodies*,  $\mathcal{C}$ , and each arc represents a *collection of tree joints and any associated loop-closure constraints*. In this paper, we refer to these collections of bodies as “body clusters” or simply “clusters,” and we refer to collections of tree joints and their loop constraints as “cluster joints.” Furthermore, we refer to a connectivity tree containing clusters and cluster joints as a Cluster Connectivity Tree (C-CT). Our clusters serve functionally the same role as the “aggregate links” developed by Jain in [33]. However, Jain arrives at his definition of aggregate links via partitioning of the system’s transformation operator - a matrix used in Spatial Operator Algebra [16] to collect the spatial transforms between all bodies in the system. We instead develop our clusters by appealing to the first principles of physics and via generalized versions of Featherstone-style spatial vector operations from conventional rigid bodies. Specifically, we demonstrate physically meaningful equivalents between the dynamic interactions of conventional bodies and the dynamic interactions of clusters. This alternative framing of constraint embedding enables the physically grounded derivations of the generalized dynamics algorithm provided in Sec. VI, VII, and VIII.

The RB-CG is always stored along with the C-CT because the topology of both the individual rigid bodies and the clusters is required for subsequent algorithms. A comparison of a system’s RB-CG, spanning tree, and C-CT is shown in Fig. 4. While we treat every node and arc in a C-CT as a cluster and cluster joint, some nodes and arcs might behave as traditional rigid bodies and joints (e.g.,  $\mathcal{C}_1$ ,  $\mathcal{C}_2$ , and  $\mathcal{C}_4$  in Fig 4). In other words, a cluster can contain only one body, and a cluster joint can have one tree joint and no loop joints.

For the most part, C-CTs can be treated the same as RB-CGs. They both follow the “regular numbering” convention, wherein each node must have a higher number than its parent and arc  $k$  connects node  $k$  and its parent [12]. For clarity, we will use indices  $i$  and  $j$  to refer to rigid bodies exclusively, and we will use indices  $k$  and  $\ell$  to refer to clusters exclusively. We define  $N_B$  as the number of rigid bodies contained by a RB-CG and  $N_C$  as the number of clusters contained by a C-CT. We use  $\mathcal{C}_k$  to refer to the  $k$ -th cluster in the C-CT and  $J_k$  to refer to its cluster joint that connects to the preceding

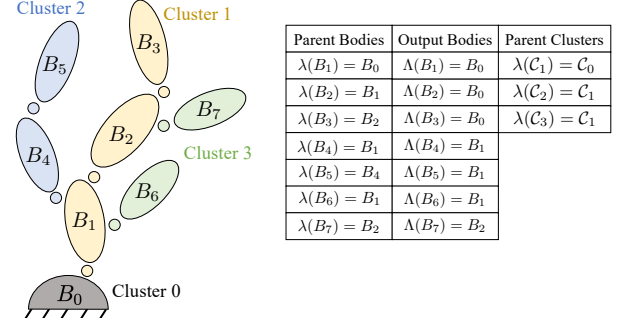


Fig. 5. Illustration of the difference between parent bodies and output bodies for a clustered rigid-body system.

cluster in the tree. The number of bodies contained by  $\mathcal{C}_k$  is given by  $n_b(\mathcal{C}_k)$ , and the  $j$ -th body in  $\mathcal{C}_k$  is given by  $\mathcal{C}_k(j)$ .

We distinguish quantities associated with individual rigid bodies and joints from quantities associated with clusters by using different fonts. Quantities associated with individual bodies and joints use the following font:  $\mathbf{v}, \mathbf{f}, \mathbf{z}, \mathbf{X}, \mathbf{Z}$ , while quantities for clusters use  $\mathbf{v}, \mathbf{f}, \mathbf{z}, \mathbf{X}, \mathbf{Z}$ . See Table I for more examples. Furthermore, for brevity, we will sometimes use the following condensed notation when denoting the quantities associated with specific bodies or clusters,

$$\mathbf{z}_i = \mathbf{z}_{B_i}, \quad \mathbf{z}_i = \mathbf{z}_{J_i}, \quad \mathbf{z}_j = \mathbf{z}_{\mathcal{C}_k(j)}, \quad \mathbf{z}_k = \mathbf{z}_{\mathcal{C}_k}.$$

The usage will be apparent based on context.

### B. Connectivity

In Sec II-A, we provided definitions that describe the connectivity of individual rigid bodies. These same definitions can be used to describe the connectivity of clusters. Whether these definitions to refer to bodies or clusters will be evident based on the context and whether they take a body or cluster as input. There are, however, some additional nuances of C-CTs that are not present in RB-CGs. For example, in the case of C-CTs, we introduce the concept of an “output body”. Output bodies play a crucial role in developing the recurrence relationships between clusters that enable the derivation of our generalized algorithms. The output body associated with body  $i$  is defined as the nearest supporting body of body  $i$  that is not in the cluster containing body  $i$ :

$$\Lambda(B_i) = \max \{B_j : \kappa(B_i) \setminus \mathcal{C}_k\}.$$

Conversely, the output children of body  $i$  are defined as:

$$\mathbf{M}(B_i) = \{B_j : \Lambda(B_j) = B_i\}.$$

The concept of an output body is an extension of the concept of a parent body. Thus, we use  $\Lambda(\cdot)$  to denote its analogous role to  $\lambda(\cdot)$ . Likewise, the concepts of output children and conventional children are related, so we use the symbols  $\mathbf{M}(\cdot)$  and  $\mu(\cdot)$ . The distinction between parent and output bodies is depicted in Fig. 5.

## V. SPATIAL VECTOR ALGEBRA: BODIES VS. CLUSTERS

Conventionally, SVA has only been applied to individual rigid bodies, as described in Sec. III. In this section, we apply SVA to body clusters, demonstrating that its favorable properties are preserved in the generalization. This novel generalization of SVA provides a mathematical foundation upon which the generalized RNEA, ABA, and EFPA algorithms will be derived in subsequent sections.

### A. Cluster Operators

The SVA quantities used to describe the motion and force interactions of body clusters are simply the concatenations of the SVA quantities for that cluster's constituent bodies. To that end, we define the following concatenation operators: "stack ( $\{\mathbf{z}_i\}_{i \in C_k}\mathbf{z}_i\}$ " creates a single vector containing the vector quantities  $\mathbf{z}$  corresponding to every rigid body in cluster  $k$ , while "block ( $\{\mathbf{Z}_i\}_{i \in C_k}\mathbf{Z}_i\}$ " creates a block diagonal matrix of the matrix quantities  $\mathbf{Z}$  corresponding to every rigid body in cluster  $k$ . Note that the order of this concatenation must follow regular numbering order. For example, considering the example in Fig 5, the only valid concatenations for a vector  $\mathbf{z}$  would be

$$\text{stack}(\{\mathbf{z}_i\}_{i \in C_1}) = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \end{bmatrix},$$

$$\text{stack}(\{\mathbf{z}_i\}_{i \in C_2}) = \begin{bmatrix} \mathbf{z}_4 \\ \mathbf{z}_5 \end{bmatrix}, \quad \text{stack}(\{\mathbf{z}_i\}_{i \in C_3}) = \begin{bmatrix} \mathbf{z}_6 \\ \mathbf{z}_7 \end{bmatrix}.$$

For the conventional spatial vectors in Sec. III, we used a preceding superscript to denote the coordinate frame of expression for the vector it accompanied. However, for the remainder of the paper, we assume that all constituent spatial quantities within a cluster spatial quantity are expressed in their local frames. Expressing the constituent quantities in alternative coordinate systems is valid and compatible with all subsequently derived algorithms but is omitted for notational conciseness. Some common applications of these concatenation operators are:

1) *Spanning Tree coordinates*: If the state of joint  $J_i$  is described by the coordinates  $\mathbf{q}_{J_i}$  and  $\dot{\mathbf{q}}_{J_i}$ , the state of cluster joint  $\mathcal{J}_k$  is described by its spanning tree coordinates

$$\mathbf{q}_{\mathcal{J}_k} = \text{stack}(\{\mathbf{q}_i\}_{i \in \mathcal{J}_k}), \quad \dot{\mathbf{q}}_{\mathcal{J}_k} = \text{stack}(\{\dot{\mathbf{q}}_i\}_{i \in \mathcal{J}_k}). \quad (11)$$

We likewise stack the spanning tree joint torques as

$$\boldsymbol{\tau}_{\mathcal{J}_k} = \text{stack}(\{\boldsymbol{\tau}_i\}_{i \in \mathcal{J}_k}).$$

2) *Spatial Motion and Force Vectors*: The motion of body  $i$  is described by spatial motion vectors  $\mathbf{v}_i$  and  $\mathbf{a}_i$ . The motion of cluster  $k$  is described by

$$\mathbf{v}_{C_k} = \text{stack}(\{\mathbf{v}_i\}_{i \in C_k}), \quad \mathbf{a}_{C_k} = \text{stack}(\{\mathbf{a}_i\}_{i \in C_k}). \quad (12)$$

Similarly, the force acting on body  $i$  is given by the spatial force vector  $\mathbf{f}_i$ , and the force acting on cluster  $k$  is

$$\mathbf{f}_{C_k} = \text{stack}(\{\mathbf{f}_i\}_{i \in C_k}). \quad (13)$$

3) *Spatial Inertia*: The spatial inertia of body  $i$  is given by  $\mathbf{I}_i$ . The spatial inertia of cluster  $k$  is given by

$$\mathbf{I}_{C_k} = \text{block}(\{\mathbf{I}_i\}_{i \in C_k}). \quad (14)$$

4) *Spatial Cross Product*: The conventional spatial cross product (5) between two spatial motion vectors for body  $i$  is given by  $\mathbf{m}_i^a \times \mathbf{m}_i^b$ . The spatial cross product between two spatial motion vectors for cluster  $k$  is given by

$$\mathbf{m}_{C_k}^a \times \mathbf{m}_{C_k}^b = \text{block}(\{\mathbf{m}_i^a \times \mathbf{m}_i^b\}_{i \in C_k}) \text{stack}(\{\mathbf{m}_i^b\}_{i \in C_k}). \quad (15)$$

The same generalization applies to the dual operator  $\times^*$  for spatial forces.

### B. Spatial Transforms

We generalize the concept of spatial transforms from Sec III by considering block matrices composed of individual spatial transform matrices, where each block row/column is associated with a conventional frame. Specifically, we define two classes of generalized spatial transforms: one that handles spatial transforms between different clusters and one that remaps parent-child relationships for a single cluster.

1) *Spatial Transforms Between Clusters*: The spatial transform between cluster  $k$  and cluster  $\ell$  serves the dual purpose of (i) performing coordinate transforms on the constituent quantities of cluster  $k$  and (ii) accounting for the output body connectivity between the clusters. We define the following indicator function  $\mathcal{E}_{\ell,k}(\cdot, \cdot)$  to describe connectivity from cluster  $k$  to cluster  $\ell$ ,

$$\mathcal{E}_{\ell,k}(i, j) = \begin{cases} 1, & \text{if } \Lambda(C_\ell(i)) = C_k(j) \\ 0, & \text{otherwise} \end{cases}. \quad (16)$$

In the case of a spatial motion transform from cluster  $k$  to cluster  $\ell$ , accounting for the output body connectivity involves stacking the constituent quantities of cluster  $k$  such that they correspond to the bodies in cluster  $\ell$  for which they are output bodies:

$${}^\ell \mathbf{X}_k = \begin{bmatrix} \mathcal{E}_{\ell,k}(1, 1)^{C_\ell(1)} \mathbf{X}_{C_k(1)} & \cdots & \mathcal{E}_{\ell,k}(1, m)^{C_\ell(1)} \mathbf{X}_{C_k(m)} \\ \vdots & \ddots & \vdots \\ \mathcal{E}_{\ell,k}(n, 1)^{C_\ell(n)} \mathbf{X}_{C_k(1)} & \cdots & \mathcal{E}_{\ell,k}(n, m)^{C_\ell(n)} \mathbf{X}_{C_k(m)} \end{bmatrix}. \quad (17)$$

In the case of the spatial force transform from cluster  $\ell$  to cluster  $k$ , accounting for the output body connectivity involves collecting the set of forces in cluster  $\ell$  that act on a common output body in cluster  $k$ :

$${}^k \mathbf{X}_\ell^* = \begin{bmatrix} \mathcal{E}_{\ell,k}(1, 1)^{C_k(1)} \mathbf{X}_{C_\ell(1)}^* & \cdots & \mathcal{E}_{\ell,k}(n, 1)^{C_k(1)} \mathbf{X}_{C_\ell(n)}^* \\ \vdots & \ddots & \vdots \\ \mathcal{E}_{\ell,k}(1, m)^{C_k(m)} \mathbf{X}_{C_\ell(1)}^* & \cdots & \mathcal{E}_{\ell,k}(n, m)^{C_k(m)} \mathbf{X}_{C_\ell(n)}^* \end{bmatrix}. \quad (18)$$

Notice that spatial transforms between clusters preserve the following crucial property of conventional spatial transforms:  ${}^\ell \mathbf{X}_k^\top = {}^k \mathbf{X}_\ell^*$ . Intuitively, this property can be understood by the dual relationship between stacking motion vectors such that they can propagate outward to their subtrees and collecting



force vectors such that they can propagate inward as net forces on shared output bodies.

Consider, for example, the system in Fig. 5. The spatial motion transforms between the clusters are

$${}^1\mathbf{X}_0 = \begin{bmatrix} {}^1\mathbf{X}_0 \\ {}^2\mathbf{X}_0 \\ {}^3\mathbf{X}_0 \end{bmatrix}, \quad {}^2\mathbf{X}_1 = \begin{bmatrix} {}^4\mathbf{X}_1 & \mathbf{0} & \mathbf{0} \\ {}^5\mathbf{X}_1 & \mathbf{0} & \mathbf{0} \end{bmatrix},$$

$${}^3\mathbf{X}_1 = \begin{bmatrix} {}^6\mathbf{X}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & {}^7\mathbf{X}_2 & \mathbf{0} \end{bmatrix},$$

while the spatial force transforms between clusters are

$${}^1\mathbf{X}_3^* = \begin{bmatrix} {}^1\mathbf{X}_6^* & \mathbf{0} \\ \mathbf{0} & {}^2\mathbf{X}_7^* \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad {}^1\mathbf{X}_2^* = \begin{bmatrix} {}^1\mathbf{X}_4^* & {}^1\mathbf{X}_5^* \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$${}^0\mathbf{X}_1^* = \begin{bmatrix} {}^0\mathbf{X}_1^* & {}^0\mathbf{X}_2^* & {}^0\mathbf{X}_3^* \end{bmatrix}.$$

By definition of  $\mathcal{E}_{\ell,k}(\cdot, \cdot)$ , there is exactly one non-zero spatial transform per block row of a generalized spatial motion transform and one per block column of a generalized spatial force transform. Thus, in software implementation, we can exploit this sparsity for efficient matrix multiplication. Such exploitation is important since it allows the computational cost of transforming spatial vectors to scale linearly with the number of bodies in the cluster. It is further worth noting that these cluster transforms are not generally square and, further, may not be invertible even when square.

2) *Spatial Transforms for Remapping Parent-Child Relationships*: While spatial transforms between clusters are analogous to conventional spatial transforms, spatial transforms for remapping parent-child relationships are a novel concept necessitated by the fact that quantities computed relative to parent bodies must be converted to quantities relative to output bodies and vice versa. For a cluster  $k$ , parent-relative motion quantities can be converted to output-relative motion quantities via

$$\text{stack} \left( \{ \mathbf{m}_i - {}^i\mathbf{m}_{\Lambda(i)} \}_{i \in \mathcal{C}_k} \right) = \Lambda^{(k)}\mathbf{X}_{\Lambda(k)} \text{stack} \left( \{ \mathbf{m}_i - {}^i\mathbf{m}_{\lambda(i)} \}_{i \in \mathcal{C}_k} \right). \quad (19)$$

where

$$\Lambda^{(k)}\mathbf{X}_{\Lambda(k)} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{X}_{2,1} & \mathbf{1} & & \vdots \\ \vdots & & \ddots & \mathbf{0} \\ \mathbf{X}_{n,1} & \dots & \mathbf{X}_{n,n-1} & \mathbf{1} \end{bmatrix}, \quad (20)$$

with

$$\mathbf{X}_{i,j} = \begin{cases} {}^{C_k(i)}\mathbf{X}_{C_k(j)}, & \text{if } C_k(j) \in \kappa(C_k(i)) \\ \mathbf{0}, & \text{otherwise} \end{cases}.$$

Since we follow regular numbering,  $\Lambda^{(k)}\mathbf{X}_{\Lambda(k)}$  is always unit lower triangular. Furthermore, spatial transforms that remap

parent-child relationships also preserve the inverse and transpose properties of conventional spatial transforms such that

$$\Lambda^{(k)}\mathbf{X}_{\Lambda(k)}^\top = {}^{\lambda(k)}\mathbf{X}_{\Lambda(k)}^*, \quad \left( {}^{\lambda(k)}\mathbf{X}_{\Lambda(k)}^* \right)^{-1} = \Lambda^{(k)}\mathbf{X}_{\Lambda(k)}. \quad (21)$$

Here,  $\Lambda^{(k)}\mathbf{X}_{\Lambda(k)}^*$  converts parent-relative force quantities to output-relative force quantities in the following sense. Suppose a collection of forces  $\mathbf{f}_k^J = \text{stack} \left( \{ \mathbf{f}_i^J \}_{i \in \mathcal{C}_k} \right)$ , where  $+\mathbf{f}_i^J$  is applied on  $B_i$  and  $-{}^{\lambda(i)}\mathbf{X}_i^* \mathbf{f}_i^J$  is applied on its parent  $B_{\lambda(i)}$ . We consider a second force system  $\mathbf{f}_k^{\mathcal{J}} = \text{stack} \left( \{ \mathbf{f}_i^{\mathcal{J}} \}_{i \in \mathcal{C}_k} \right)$  where  $+\mathbf{f}_i^{\mathcal{J}}$  is applied on  $B_i$  and  $-{}^{\Lambda(i)}\mathbf{X}_i^* \mathbf{f}_i^{\mathcal{J}}$  is applied on  $B_{\Lambda(i)}$ . Then, relating these two force systems via:

$$\mathbf{f}_k^{\mathcal{J}} = \Lambda^{(k)}\mathbf{X}_{\Lambda(k)}^* \mathbf{f}_k^J \quad (22)$$

gives a statically equivalent set of forces on every body in cluster  $k$  and on every output body in its parent. The forces in  $\mathbf{f}_k^J$  represent forces exchanged across conventional joints in the spanning tree, with  $\mathbf{f}_k^{\mathcal{J}}$  the forces we view as exchanged across a cluster joint (between bodies and their output bodies).

The general formula for  $\Lambda^{(k)}\mathbf{X}_{\Lambda(k)}^*$  is given by

$$\Lambda^{(k)}\mathbf{X}_{\Lambda(k)}^* = \begin{bmatrix} \mathbf{1} & -\mathbf{X}_{1,2}^* & \dots & -\mathbf{X}_{1,n}^* \\ \mathbf{0} & \mathbf{1} & & \vdots \\ \vdots & & \ddots & -\mathbf{X}_{n-1,n}^* \\ \mathbf{0} & \dots & \mathbf{0} & \mathbf{1} \end{bmatrix}, \quad (23)$$

with

$$\mathbf{X}_{i,j}^* = \begin{cases} {}^{C_k(i)}\mathbf{X}_{C_k(j)}^*, & \text{if } C_k(i) = \lambda(C_k(j)) \\ \mathbf{0}, & \text{otherwise} \end{cases}.$$

It can be shown that each column of  $\Lambda^{(k)}\mathbf{X}_{\Lambda(k)}^*$  will always have at most two non-zero blocks (1) the identity on the block diagonal and (2) an entry  $-{}^{\lambda(i)}\mathbf{X}_i^*$  in the row corresponding to the parent, which appears only when the parent is in cluster  $k$ .

Consider, for example, for the system in Fig. 5. The spatial transforms remapping the parent-child motion relationships are

$$\Lambda^{(1)}\mathbf{X}_{\Lambda(1)} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ {}^2\mathbf{X}_1 & \mathbf{1} & \mathbf{0} \\ {}^3\mathbf{X}_1 & {}^3\mathbf{X}_2 & \mathbf{1} \end{bmatrix},$$

$$\Lambda^{(2)}\mathbf{X}_{\Lambda(2)} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ {}^5\mathbf{X}_4 & \mathbf{1} \end{bmatrix}, \quad \Lambda^{(3)}\mathbf{X}_{\Lambda(3)} = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}.$$

while the spatial transforms remapping the parent-child force relationships are

$$\Lambda^{(1)}\mathbf{X}_{\Lambda(1)}^* = \begin{bmatrix} \mathbf{1} & -{}^1\mathbf{X}_2^* & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & -{}^2\mathbf{X}_3^* \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix},$$

$$\Lambda^{(2)}\mathbf{X}_{\Lambda(2)}^* = \begin{bmatrix} \mathbf{1} & -{}^4\mathbf{X}_5^* \\ \mathbf{0} & \mathbf{1} \end{bmatrix}, \quad \Lambda^{(3)}\mathbf{X}_{\Lambda(3)}^* = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}.$$

### C. Cluster Motion Subspace Matrix

In the same way that the conventional motion subspace matrix  $\mathbf{S}$  described in Sec III-E encodes the relative velocities permitted between two bodies connected by a joint, a cluster joint's motion subspace matrix  $\mathbf{S}$  encodes the relative velocities permitted between two clusters. The nuance is that the relative velocity for clusters is the velocity between the bodies in the current cluster and their *output bodies*. It is straightforward to handle this nuance using the spatial transform  ${}^{\Lambda(\cdot)}\mathbf{X}_{\lambda(\cdot)}$  described in Sec. V-B2.

We define the spatial velocity across a cluster joint as the concatenation of the spatial velocity of every body in cluster  $k$  relative to its output body:

$$\mathbf{v}_k^{\mathcal{J}} = \text{stack} \left( \left\{ \mathbf{v}_i - {}^i\mathbf{v}_{\Lambda(i)} \right\}_{i \in \mathcal{C}_k} \right).$$

The velocity of every body in cluster  $k$  relative to its parent can be found using the conventional motion subspace matrices

$$\mathbf{v}_i - {}^i\mathbf{X}_{\lambda(i)}\mathbf{v}_{\lambda(i)} = \mathbf{S}_i\dot{\mathbf{q}}_i \quad \forall i \in \mathcal{C}_k.$$

Thus, recalling (19), the spatial velocity across cluster joint  $\mathcal{J}_k$  can be computed via

$$\begin{aligned} \mathbf{v}_k^{\mathcal{J}} &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)} \text{stack} \left( \left\{ \mathbf{S}_i\dot{\mathbf{q}}_i \right\}_{i \in \mathcal{C}_k} \right) \\ &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)} \text{block} \left( \left\{ \mathbf{S}_i \right\}_{i \in \mathcal{C}_k} \right) \dot{\mathbf{q}}_k \\ &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)} \text{block} \left( \left\{ \mathbf{S}_i \right\}_{i \in \mathcal{C}_k} \right) \mathbf{G}_k \dot{\mathbf{y}}_k, \end{aligned}$$

where  $\mathbf{G}_k$  is the explicit constraint Jacobian associated with cluster  $k$ . This leads to the definition of the cluster motion subspace matrix

$$\mathbf{S}_k = {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)} \text{block} \left( \left\{ \mathbf{S}_i \right\}_{i \in \mathcal{C}_k} \right) \mathbf{G}_k. \quad (24)$$

Computing the spatial acceleration across a cluster joint involves differentiating the cluster motion subspace matrix,

$$\mathbf{a}_k^{\mathcal{J}} = \left( \frac{d}{dt} \mathbf{S}_k \dot{\mathbf{y}}_k \right) = \dot{\mathbf{S}}_k \dot{\mathbf{y}}_k + \mathbf{S}_k \ddot{\mathbf{y}}_k. \quad (25)$$

Recalling the rules for differentiation in moving coordinates from Sec III,

$${}^k\dot{\mathbf{S}}_k = \frac{d}{dt} {}^k\mathbf{S}_k + {}^k\mathbf{v}_k \times {}^k\mathbf{S}_k. \quad (26)$$

While  $\frac{d}{dt} {}^i\mathbf{S}_i$  is zero for most conventional tree joints,  $\frac{d}{dt} {}^k\mathbf{S}_k$  is frequently non-zero due to the presence of configuration-dependent quantities  ${}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}$  and  $\mathbf{G}_k$ .

### D. Cluster Force Subspace Matrix

Cluster force subspace matrices will be used to encode how the generalized cluster force  $\tau$  and the cluster joint constraint forces  $\lambda^c$  relate to the total spatial force transmitted across the cluster joint. For a given cluster motion subspace, the choice of cluster force subspace matrices is not, in general, unique. However, in this section, we provide a physically grounded method to select these matrices.

Recall from (7) that the force exchanged across a conventional spanning tree joint,  $\mathbf{f}_i^{\mathcal{J}}$ , represents the force needed to move the subtree starting at body  $i$ . When body  $i$  is in cluster  $k$ , that subtree can be decomposed into three parts:

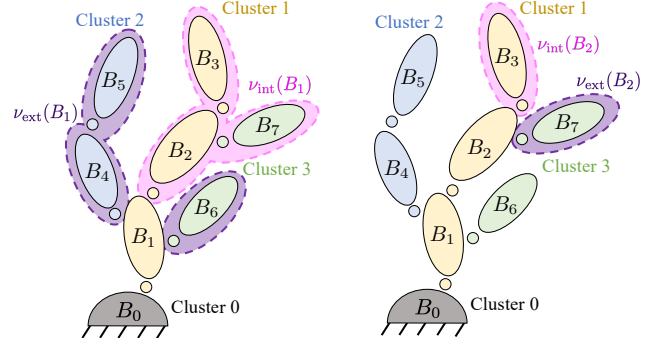


Fig. 6. Decomposition of subtrees into their internal (pink) and external (purple) components.

the current body  $i$ , the *internally* supported subtree and the *externally* supported subtree. The internally supported subtree combines all of the subtrees of the children of body  $i$  that are in cluster  $k$ ,

$$\nu_{\text{int}}(B_i) = \bigcap_{B_j \in \mu(B_i) \cap \mathcal{C}_k} \nu(B_j),$$

while the externally supported subtree combines all of the subtrees of the children of body  $i$  that are not in cluster  $k$

$$\nu_{\text{ext}}(B_i) = \bigcap_{B_j \in \mu(B_i) \setminus \mathcal{C}_k} \nu(B_j).$$

Using the same system from Fig. 5, Fig. 6 illustrates the decomposition of the subtrees of  $B_1$  and  $B_2$  into their internal and external components.

The force exchanged across a cluster joint,  $\mathbf{f}_i^{\mathcal{J}}$ , represents the force needed to move body  $i$  as well as the bodies in the externally supported subtree. Mathematically,  $\mathbf{f}_i^{\mathcal{J}}$  can be recovered by subtracting away the force needed to move the internally supported subtree:

$$\mathbf{f}_i^{\mathcal{J}} = \mathbf{f}_i^J - \sum_{B_j \in \mu(B_i) \cap \mathcal{C}_k} {}^i\mathbf{X}_j^* \mathbf{f}_j^J. \quad (27)$$

which, equivalently, leaves the forces required to move body  $i$  and the externally supported subtree:

$$\mathbf{f}_i^{\mathcal{J}} = \mathbf{f}_i^{\text{net}} + \sum_{j \in \nu_{\text{ext}}(i)} \mathbf{f}_j^{\text{net}} \quad (28)$$

For example, for the system shown in Fig. 5

$$\begin{aligned} \begin{bmatrix} \mathbf{f}_1^{\mathcal{J}} \\ \mathbf{f}_2^{\mathcal{J}} \\ \mathbf{f}_3^{\mathcal{J}} \end{bmatrix} &= \begin{bmatrix} \mathbf{f}_1^J - {}^1\mathbf{X}_2^* \mathbf{f}_2^J \\ \mathbf{f}_2^J - {}^2\mathbf{X}_3^* \mathbf{f}_3^J \\ \mathbf{f}_3^J \end{bmatrix} = {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}^* \begin{bmatrix} \mathbf{f}_1^J \\ \mathbf{f}_2^J \\ \mathbf{f}_3^J \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{f}_1^{\text{net}} + {}^1\mathbf{X}_4^* \mathbf{f}_4^{\text{net}} + {}^1\mathbf{X}_5^* \mathbf{f}_5^{\text{net}} + {}^1\mathbf{X}_6^* \mathbf{f}_6^{\text{net}} \\ \mathbf{f}_2^{\text{net}} + {}^2\mathbf{X}_7^* \mathbf{f}_7^{\text{net}} \\ \mathbf{f}_3^{\text{net}} \end{bmatrix} \end{aligned}$$

This relationship between  $\mathbf{f}_i^{\mathcal{J}}$  and  $\mathbf{f}_i^J$ , originally described in (22), holds in general. Furthermore, we can express  $\mathbf{f}_i^J$  in terms of its conventional force subspace matrices (8). Then, assuming that only the tree joints are actuated, the spatial force

across a cluster joint can be computed from parent-relative spatial forces via

$$\begin{aligned} \mathbf{f}_k^{\mathcal{J}} &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}^* \text{stack} \left( \{ \mathbf{T}_i^a \boldsymbol{\tau}_i + \mathbf{T}_i^c \boldsymbol{\lambda}_i^c \}_{i \in \mathcal{C}_k} \right) \\ &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}^* \text{block} \left( \{ \mathbf{T}_i^a \}_{i \in \mathcal{C}_k} \right) \text{stack} \left( \{ \boldsymbol{\tau}_i \}_{i \in \mathcal{C}_k} \right) + \\ &\quad {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}^* \text{block} \left( \{ \mathbf{T}_i^c \}_{i \in \mathcal{C}_k} \right) \text{stack} \left( \{ \boldsymbol{\lambda}_i^c \}_{i \in \mathcal{C}_k} \right) \end{aligned}$$

The identities in (9) are preserved in concatenation,

$$\begin{aligned} \text{block} \left( \{ \mathbf{S}_i^{\top} \}_{i \in \mathcal{C}_k} \right) \text{block} \left( \{ \mathbf{T}_i^a \}_{i \in \mathcal{C}_k} \right) &= \mathbf{1} \\ \text{block} \left( \{ \mathbf{S}_i^{\top} \}_{i \in \mathcal{C}_k} \right) \text{block} \left( \{ \mathbf{T}_i^c \}_{i \in \mathcal{C}_k} \right) &= \mathbf{0} \end{aligned}$$

and so,

$$\mathbf{S}_k^{\top} \mathbf{f}_k^{\mathcal{J}} = \mathbf{G}_k^{\top} \text{stack} \left( \{ \boldsymbol{\tau}_i \}_{i \in \mathcal{C}_k} \right) = \mathbf{G}_k^{\top} \boldsymbol{\tau}_{\mathcal{J}_k} \quad (29)$$

However, since the generalized velocities for the cluster are related to those for the spanning tree joints via  $\dot{\mathbf{q}}_{\mathcal{J}_k} = \mathbf{G}_k \dot{\mathbf{y}}_k$ , it follows (via virtual work principles) that the corresponding generalized force for the cluster  $\boldsymbol{\tau}_k$  is related to the spanning tree generalized forces via:

$$\boldsymbol{\tau}_k = \mathbf{G}_k^{\top} \boldsymbol{\tau}_{\mathcal{J}_k}.$$

Since the generalized force on the cluster is the only set of forces necessary to specify for determining its motion, any other  $\boldsymbol{\tau}_{\mathcal{J}_k}$  satisfying the above would cause the mechanism to move the same way.

Thus, we define any  $\mathbf{P}_k$  to lift the set of cluster generalized forces on the independent joints to a dynamically equivalent set of generalized forces for the spanning joints, where  $\mathbf{P}_k$  satisfies the property  $\mathbf{G}_k^{\top} \mathbf{P}_k = \mathbf{1}$ . For example, the Moore-Penrose pseudoinverse of  $\mathbf{G}_k^{\top}$  would be one such choice. We consider an analogous matrix  $\mathbf{P}_k^c$  for the constraint forces, where any complementary full-rank matrix satisfying  $\mathbf{G}_k^{\top} \mathbf{P}_k^c = \mathbf{0}$  will suffice. This leads to the definition of the active and constraint cluster force subspaces

$$\begin{aligned} \mathbf{T}_k^a &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}^* \text{block} \left( \{ \mathbf{T}_i^a \}_{i \in \mathcal{C}_k} \right) \mathbf{P}_k \\ \mathbf{T}_k^c &= {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}^* \text{block} \left( \{ \mathbf{T}_i^c \}_{i \in \mathcal{C}_k} \right) \mathbf{P}_k^c \end{aligned} \quad (30)$$

Finally, analogous to (10), the generalized cluster force at the independent joints  $\boldsymbol{\tau}_k$  is related to  $\mathbf{f}_k^{\mathcal{J}}$  via

$$\mathbf{S}_k^{\top} \mathbf{f}_k^{\mathcal{J}} = \mathbf{S}_k^{\top} (\mathbf{T}_k^a \boldsymbol{\tau}_k + \mathbf{T}_k^c \boldsymbol{\lambda}_k^c) = \boldsymbol{\tau}_k. \quad (31)$$

## VI. INVERSE DYNAMICS

This section derives the Cluster-RNEA (C-RNEA). The C-RNEA is an extension of the original RNEA that accounts for loop-closure constraints through clusters. The C-RNEA retains the same forward and backward pass structure as the original RNEA, with the only difference being that the C-RNEA iterates over clusters, whereas the original RNEA iterates over individual rigid bodies.

### A. Forward Pass

The forward pass of the conventional RNEA progresses by using the motion subspace matrices and their derivatives to propagate the velocities and accelerations of every body in the tree outward from the base. Each iteration only requires that the parent quantities and joint state be known so that the following recurrence relationship can be propagated,

$$\begin{aligned} \mathbf{v}_i &= {}^i\mathbf{X}_{\lambda(i)} \mathbf{v}_{\lambda(i)} + \mathbf{S}_i \dot{\mathbf{q}}_i, \\ \mathbf{a}_i &= {}^i\mathbf{X}_{\lambda(i)} \mathbf{a}_{\lambda(i)} + \dot{\mathbf{S}}_i \dot{\mathbf{q}}_i + \mathbf{S}_i \ddot{\mathbf{q}}_i. \end{aligned} \quad (32)$$

The generalization of the motion subspace matrix (24) and its derivative (26) permits an analogous relationship for the C-CT,

$$\begin{aligned} \mathbf{v}_k &= {}^k\mathbf{X}_{\lambda(k)} \mathbf{v}_{\lambda(k)} + \mathbf{S}_k \dot{\mathbf{y}}_k, \\ \mathbf{a}_k &= {}^k\mathbf{X}_{\lambda(k)} \mathbf{a}_{\lambda(k)} + \dot{\mathbf{S}}_k \dot{\mathbf{y}}_k + \mathbf{S}_k \ddot{\mathbf{y}}_k. \end{aligned} \quad (33)$$

However, since this recurrence is defined with respect to output body quantities, the forward pass of the C-RNEA iterates over clusters rather than individual bodies. To seed the recurrence, the root body is fixed in place ( $\mathbf{v}_0 = \mathbf{v}_0 = \mathbf{0}$ ) but is assumed to be accelerating upward at the acceleration of gravity ( $\mathbf{a}_0 = \mathbf{a}_g = -\mathbf{a}_g$ ) so that gravity forces do not need to be explicitly accounted for.

The relationship between the net force  $\mathbf{f}_k^{\text{net}}$  acting on cluster  $k$  and the motion of cluster  $k$  extends straightforwardly from (6) via

$$\begin{aligned} \mathbf{f}_k^{\text{net}} &= \text{stack} \left( \{ \mathbf{I}_i \mathbf{a}_i \}_{i \in \mathcal{C}_k} \right) + \text{stack} \left( \{ \mathbf{v}_i \times {}^* \mathbf{I}_i \mathbf{v}_i \}_{i \in \mathcal{C}_k} \right) \\ &= \mathbf{I}_k \mathbf{a}_k + \mathbf{v}_k \times {}^* \mathbf{I}_k \mathbf{v}_k. \end{aligned} \quad (34)$$

Successive iterations of (33) and (34), with  $k$  ranging from 1 to  $N_C$ , constitute the forward pass of the C-RNEA.

### B. Backward Pass

The backward pass of the conventional RNEA develops a recurrence relationship to propagate the spatial forces  $\mathbf{f}_i^{\mathcal{J}}$  exchanged across the joints down the kinematic tree, starting from the leaves. The backward pass of the C-RNEA is similar, with the critical difference being that at each iteration, the forces  $\mathbf{f}_i^{\mathcal{J}}$  on the children within each cluster are propagated backward onto their respective output bodies in the parent cluster. The cluster-based recursion can be derived from a free-body analysis of a cluster. The conventional analysis from (7) yields

$$\mathbf{f}_i^{\text{net}} = \mathbf{f}_i^{\mathcal{J}} - \sum_{B_j \in \mu(B_i)} {}^i\mathbf{X}_j^* \mathbf{f}_j^{\mathcal{J}}.$$

However, recalling from (22) that the forces  $\mathbf{f}_i^{\mathcal{J}}$  exchanged between spanning tree bodies and their parents are statically equivalent to the forces  $\mathbf{f}_i^{\mathcal{J}}$  exchanged between bodies in a cluster and their output bodies in the parent cluster, the net force can also be expressed

$$\mathbf{f}_i^{\text{net}} = \mathbf{f}_i^{\mathcal{J}} - \sum_{B_j \in \mathcal{M}(B_i)} {}^i\mathbf{X}_j^* \mathbf{f}_j^{\mathcal{J}}.$$

This leads to the cluster-based recurrence relationship for the C-CT:

$$\mathbf{f}_k^{\mathcal{J}} = \mathbf{f}_k^{\text{net}} + \sum_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)} {}^k\mathbf{X}_\ell^* \mathbf{f}_\ell^{\mathcal{J}}. \quad (35)$$

For an example of this recurrence relationship, consider  $\mathcal{C}_1$  in the system shown in Fig. 5.

$$\begin{aligned} \begin{bmatrix} \mathbf{f}_1^{\text{net}} \\ \mathbf{f}_2^{\text{net}} \\ \mathbf{f}_3^{\text{net}} \end{bmatrix} &= \begin{bmatrix} \mathbf{f}_1^J - {}^1\mathbf{X}_2^* \mathbf{f}_2^J - {}^1\mathbf{X}_4^* \mathbf{f}_4^J - {}^1\mathbf{X}_6^* \mathbf{f}_6^J \\ \mathbf{f}_2^J - {}^2\mathbf{X}_3^* \mathbf{f}_3^J - {}^2\mathbf{X}_7^* \mathbf{f}_7^J \\ \mathbf{f}_3^J \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{f}_1^J - {}^1\mathbf{X}_4^* (\mathbf{f}_4^J + {}^4\mathbf{X}_5^* \mathbf{f}_5^J) - {}^1\mathbf{X}_6^* \mathbf{f}_6^J \\ \mathbf{f}_2^J - {}^2\mathbf{X}_7^* \mathbf{f}_7^J \\ \mathbf{f}_3^J \end{bmatrix} \\ &= \mathbf{f}_1^J - {}^1\mathbf{X}_2^* \mathbf{f}_2^J - {}^1\mathbf{X}_3^* \mathbf{f}_3^J \end{aligned}$$

By definition, the clusters at the tips of the C-CT have no children. Thus, for these clusters  $\mathbf{f}^J = \mathbf{f}^{\text{net}}$ . With these values initialized in the forward pass (34), the backward pass of the C-RNEA progresses with successive iterations of (35), with  $k$  ranging from  $N_C$  to 1. Once  $\mathbf{f}^J$  has been updated for every cluster in the C-CT, the cluster generalized forces can be computed using (31).

### C. Cluster-based Recursive Newton Euler Algorithm

The ability to preserve the recursive structure of the algorithm while still accounting for loop constraints makes the cluster-based algorithm faster than alternative algorithms for constrained inverse dynamics. However, the cluster-based algorithm is more computationally complex than the conventional RNEA. It is challenging to state the exact computational complexity of the algorithm because, in practice, it depends on the complexity of the loop constraints created by the sub-mechanisms. For example, the constraint created by a geared transmission is linear and, therefore, almost trivial to compute. Differential drives, on the other hand, such as those depicted in Fig. 1, have complex, configuration-dependent explicit constraints Jacobians that can be relatively costly to compute.

However, if we neglect the computational cost associated with computing the constraint Jacobians and biases, the C-RNEA has a computational complexity of  $O(N_C l^2)$ , where  $l$  is the number of bodies in the largest cluster (i.e., the size of the largest loop). This complexity assumes that every cluster contains  $l$  bodies, i.e.,  $N_C = N_B/l$  and can therefore be considered a “worst-case” complexity since clusters may contain less than  $l$  bodies. In the case where the system model can be represented as a RB-CG kinematic tree ( $N_C = N_B$ ,  $l = 1$ ), the C-RNEA is identical to the RNEA.

**Remark 1.** An excellent recent paper by Kumar et al. [35] proposed a conceptually similar inverse dynamics algorithm for addressing local loop closures. Their method similarly exploits the spanning tree but does so by carrying out the conventional RNEA on the spanning tree itself and then projecting the result down to independent coordinates. By comparison, the generalization of SVA to clusters herein, as inspired by Jain’s LCE [33], [42], allows the cluster RNEA to take the same exact structural form and compactness as the original RNEA, but with the internal quantities taking on a slightly different physical meaning. As shown in the subsequent sections, this enables the cluster SVA methodology to

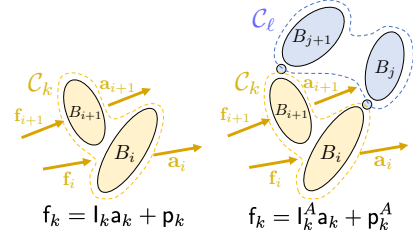


Fig. 7. Illustration of the concept of articulated-cluster inertia.

likewise generalize (with suitable modifications) the concepts of the Gauss principle, articulated-body inertias, and force propagators to the case of working with clusters for forward dynamics and operational-space inertia calculations.

## VII. FORWARD DYNAMICS

The original ABA consists of three passes [6], [8]: a forward pass that computes the first-order forward kinematics, a backward pass to compute the articulated-body quantities, and a forward pass that propagates the accelerations of each body outward from the base. The backward pass depends on a recursive relationship between the articulated-body quantities of child and parent bodies, and the second forward pass depends on a recursive relationship between the joint and spatial accelerations of the child and parent bodies.

There are multiple ways to derive these recursive relationships, such as the assembly method [6] and the matrix-factorization method [17]. However, arguably the most physically intuitive derivation, especially for roboticists, is based on the GPLC [29]. In this section, we demonstrate that a similar recursive relationship for computing articulated-cluster quantities can be intuitively derived via the GPLC, laying the groundwork for the Cluster-based Articulated-Body Algorithm (C-ABA).

### A. Articulated-Cluster Inertia

A rigid body’s articulated-body inertia describes the inertia that the rigid body *appears to have* when it is part of a rigid-body subtree [6]. In isolation, body  $i$  follows the spatial equation of motion

$$\mathbf{f}_i^J = \mathbf{I}_i \mathbf{a}_i + \mathbf{p}_i, \quad (36)$$

where  $\mathbf{p}_i$  is the bias force. By contrast, the “articulated body” associated with body  $i$  results from attaching body  $i$  to its child subtrees via joints, then following the spatial equation of motion

$$\mathbf{f}_i^J = \mathbf{I}_i^A \mathbf{a}_i + \mathbf{p}_i^A, \quad (37)$$

where  $\mathbf{I}_i^A$  and  $\mathbf{p}_i^A$  are the articulated-body inertia and bias force of the multi-body system, respectively.

The concept extends analogously to clusters of bodies, wherein attaching cluster  $k$  to its child subtrees via cluster joints, as shown in Fig. 7 for two clusters, yields the articulated-cluster equation of motion

$$\mathbf{f}_k^J = \mathbf{I}_k^A \mathbf{a}_k + \mathbf{p}_k^A, \quad (38)$$

where  $\mathbf{l}_k^A$  and  $\mathbf{p}_k^A$  are the articulated-*cluster* inertia and bias force of the multi-*cluster* system, respectively. The generalization in (38) is equivalent to a “multi-handle” equation as described in [26].

The key nuance here compared to standard articulated inertias is that articulated cluster inertia  $\mathbf{l}_k^A$  captures the effective inertia of the cluster subtree rooted at cluster  $k$ , with the constraints of cluster joint  $k$  removed, but the remainder of the joints in the subtree left free. That is, the cluster articulated inertia does not account for effective inertia that would be felt due to the interconnection of bodies within the cluster or to parents earlier in the tree. In this regard, for any leaf cluster  $\mathcal{C}_k$ , we have that  $\mathbf{l}_k^A = \mathbf{l}_k$  is block diagonal, simply representing the spatial inertias of each body in the leaf cluster, and, in the absence of external forces  $\mathbf{p}_k^A = \mathbf{v}_k \times^* \mathbf{l}_k \mathbf{v}_k$ .

### B. Calculating Articulated-Cluster Inertia via the Gauss Principle of Least Constraint

In this section, we will demonstrate that the GPLC can be used to determine the update laws for the articulated cluster inertia  $\mathbf{l}_k^A$  and bias force  $\mathbf{p}_k^A$  of every cluster in the C-CT. We approach the problem by using DP to solve the optimization arising from the GPLC for the system. Starting with the leaf clusters, we prove inductively that the optimal Gauss cost to go for each of the cluster subtrees is quadratic with respect to the spatial acceleration of the cluster. The desired update laws emerge from propagating this optimal Gauss cost to go backward along the tree.

The GPLC roughly states the constrained motion of a rigid body system will deviate minimally (in a least squares sense) from the unconstrained motion of that system. The unconstrained motion is defined as the motion that would result in the absence of all constraint forces,  $\lambda^c$ . Considering the system as a whole, the GPLC can be expressed as an optimization problem,

$$\begin{aligned} \min_{\mathbf{a}, \dot{\mathbf{y}}} \quad & \frac{1}{2} \sum_{k=1}^{N_C} \|\bar{\mathbf{a}}_k^{\text{uc}} - \mathbf{a}_k\|_{\mathbf{l}_k}^2 \\ \text{s.t.} \quad & \mathbf{a}_k = {}^k\mathbf{X}_{\lambda(k)} \mathbf{a}_{\lambda(k)} + \mathbf{S}_k \ddot{\mathbf{y}}_k + \dot{\mathbf{S}}_k \dot{\mathbf{y}}_k, \\ & \mathbf{a}_0 = -\mathbf{a}_g. \end{aligned} \quad (39)$$

where, in this case, the unconstrained cluster accelerations are the accelerations that would result from generalized forces on the current cluster, reactions from generalized forces on the child clusters, and bias forces on the current cluster:

$$\mathbf{l}_k \bar{\mathbf{a}}_k^{\text{uc}} = \mathbf{T}_k^a \boldsymbol{\tau}_k - \sum_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)} {}^k\mathbf{X}_\ell^* \mathbf{T}_\ell^a \boldsymbol{\tau}_\ell - \mathbf{v}_k \times^* \mathbf{l}_k \mathbf{v}_k. \quad (40)$$

Note that the applied forces  $\mathbf{T}_k^a \boldsymbol{\tau}_k$  and  $\mathbf{T}_\ell^a \boldsymbol{\tau}_\ell$  include equal and opposite effects between bodies within their respective clusters due to the presence of  ${}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}$  and  ${}^{\Lambda(\ell)}\mathbf{X}_{\lambda(\ell)}$  within  $\mathbf{T}_k^a$  and  $\mathbf{T}_\ell^a$ . However, when transforming the reaction forces  $-\mathbf{T}_\ell^a \boldsymbol{\tau}_\ell$  back to the output bodies in cluster  $k$  via  ${}^k\mathbf{X}_\ell^*$ , it is only reaction torques from the immediate children in the spanning tree that will appear. In this regard,  $\bar{\mathbf{a}}_k^{\text{uc}}$  represents the same unconstrained accelerations one would have for the spanning tree with all joints removed.

Because every C-CT is a kinematic tree, the optimization over  $N_C$  clusters in (39) can be solved via dynamic programming (DP), generalizing the original derivation in [8] for serial chains. Specifically, we use DP to determine the optimal Gauss cost to go for each cluster subtree:

$$\begin{aligned} V_k^*(\mathbf{a}_k) = \min_{\{\mathbf{a}_\ell, \dot{\mathbf{y}}_\ell\}_{\mathcal{C}_\ell \in \bar{\nu}(\mathcal{C}_k)}} \quad & \frac{1}{2} \sum_{\mathcal{C}_\ell \in \nu(\mathcal{C}_k)} \|\bar{\mathbf{a}}_\ell^{\text{uc}} - \mathbf{a}_\ell\|_{\mathbf{l}_\ell}^2 \\ \text{s.t.} \quad & \mathbf{a}_\ell = {}^\ell\mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \dot{\mathbf{S}}_\ell \dot{\mathbf{y}}_\ell, \end{aligned} \quad (41)$$

where  $\bar{\nu}(\mathcal{C}_k)$  defines the exclusive subtree of cluster  $k$  such that  $\bar{\nu}(\mathcal{C}_k) = \nu(\mathcal{C}_k) \setminus \mathcal{C}_k$ . Via a minor generalization of Bellman’s principle, (41) must satisfy:

$$\begin{aligned} V_k^*(\mathbf{a}_k) = \min_{\{\ddot{\mathbf{y}}_\ell\}_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)}} \quad & \frac{1}{2} \|\bar{\mathbf{a}}_k^{\text{uc}} - \mathbf{a}_k\|_{\mathbf{l}_k}^2 + \sum_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)} V_\ell^*(\mathbf{a}_\ell) \\ \text{where} \quad & \mathbf{a}_\ell = {}^\ell\mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \dot{\mathbf{S}}_\ell \dot{\mathbf{y}}_\ell, \end{aligned} \quad (42)$$

With  $\mathbf{a}_k$  fixed, this optimization can be considered a set of decoupled optimizations over the child clusters of cluster  $k$ .

We parameterize the candidate optimal Gauss cost to go function as quadratic in the spatial acceleration of the cluster and will show that it takes the form:

$$V_k^*(\mathbf{a}_k) = \frac{1}{2} \|\bar{\mathbf{a}}_k^{\text{uc}} - \mathbf{a}_k\|_{\mathbf{l}_k^A}^2 + \text{constant}, \quad (43)$$

where  $\bar{\mathbf{a}}_k^{\text{uc}}$  is defined such that:

$$\mathbf{T}_k^a \boldsymbol{\tau}_k = \mathbf{l}_k^A \bar{\mathbf{a}}_k^{\text{uc}} + \mathbf{p}_k^A. \quad (44)$$

Note the distinction between  $\mathbf{a}_k^{\text{uc}}$  and  $\bar{\mathbf{a}}_k^{\text{uc}}$ . The terms  $\bar{\mathbf{a}}_k^{\text{uc}}$  represented the unconstrained accelerations when all joint constraints in the mechanism were removed, while  $\mathbf{a}_k^{\text{uc}}$  will be shown to represent the resulting unconstrained acceleration when the constraints from cluster joint  $k$  alone are removed. In this regard, we can view (43) as representing the Gauss cost for the articulated inertia (where joint constraints within child clusters have already been accounted for), whereas the one in (41) represents the corresponding cost for the bodies considered individually without any joint constraints yet applied.

**Remark 2.** A slightly different version of an optimal Gauss cost-to-go was considered in [29] for rigid bodies. In their formulation, the articulated inertia and bias force show up directly in the quadratic and linear terms of the Gauss cost-to-go. A similar approach could be considered with clusters. However, we favor our form of the optimal Gauss cost-to-go above due to its direct interpretation as a Gauss cost for an articulated cluster system which is then subject to an acceleration constraint on its root cluster.

Starting with the leaf clusters as the base case, (43) is satisfied immediately since,  $\mathbf{l}_k = \mathbf{l}_k^A$ ,  $\bar{\mathbf{a}}_k^{\text{uc}} = \mathbf{a}_k^{\text{uc}}$ , and  $\mu(\mathcal{C}_k)$  is empty for all leaves. As such, the constant term in (43) for the leaf clusters is zero. We will show inductively that (43) holds for all clusters by carrying out (42) parametrically in  $\mathbf{a}_k$ , proceeding backward along the tree.

Noting the form of (42), we consider optimization over individual child clusters,

$$\begin{aligned} \min_{\mathbf{a}_\ell, \ddot{\mathbf{y}}_\ell} \quad & \frac{1}{2} \|\mathbf{a}_\ell^{\text{uc}} - \mathbf{a}_\ell\|_{\mathbf{I}_\ell^A}^2 \\ \text{s.t.} \quad & \mathbf{a}_\ell = {}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \dot{\mathbf{S}}_\ell \dot{\mathbf{y}}_\ell. \end{aligned} \quad (45)$$

showing how the solution to these problems for all children of cluster  $k$  will give us  $V_k^*(\mathbf{a}_k)$ .

To solve (45), we first expand the quadratic terms and drop the terms that do not depend on optimization variables

$$\begin{aligned} \min_{\mathbf{a}_\ell, \ddot{\mathbf{y}}_\ell} \quad & \frac{1}{2} \|\mathbf{a}_\ell\|_{\mathbf{I}_\ell^A}^2 - \mathbf{a}_\ell^\top \mathbf{I}_\ell^A \mathbf{a}_\ell^{\text{uc}} \\ \text{s.t.} \quad & \mathbf{a}_\ell = {}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \mathbf{c}_\ell, \end{aligned} \quad (46)$$

where  $\mathbf{c}_\ell = \dot{\mathbf{S}}_\ell \dot{\mathbf{y}}_\ell$ . Next, the unconstrained accelerations can be replaced via (44) such that

$$\begin{aligned} \min_{\mathbf{a}_\ell, \ddot{\mathbf{y}}_\ell} \quad & \frac{1}{2} \|\mathbf{a}_\ell\|_{\mathbf{I}_\ell^A}^2 - \mathbf{a}_\ell^\top (\mathbf{T}_\ell^a \boldsymbol{\tau}_\ell - \mathbf{p}_\ell^A) \\ \text{s.t.} \quad & \mathbf{a}_\ell = {}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \mathbf{c}_\ell. \end{aligned} \quad (47)$$

And finally, the constraint can be eliminated by expressing  $\mathbf{a}_\ell$  in terms of  $\mathbf{a}_k$  and  $\ddot{\mathbf{y}}_\ell$  as dictated by the constraint,

$$\begin{aligned} \min_{\ddot{\mathbf{y}}_\ell} \quad & \frac{1}{2} \|{}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \mathbf{c}_\ell\|_{\mathbf{I}_\ell^A}^2 - \\ & ({}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \mathbf{c}_\ell)^\top (\mathbf{T}_\ell^a \boldsymbol{\tau}_\ell - \mathbf{p}_\ell^A). \end{aligned} \quad (48)$$

The next step in the derivation is to use the first order optimality conditions for  $\ddot{\mathbf{y}}_\ell$  to derive a recursive relationship with  $\mathbf{a}_k$ . Let us first define some helpful intermediate quantities.

$$\mathbf{U}_\ell = \mathbf{I}_\ell^A \mathbf{S}_\ell, \quad \mathbf{D}_\ell = \mathbf{S}_\ell^\top \mathbf{U}_\ell, \quad \mathbf{u}_\ell = \boldsymbol{\tau}_\ell - \mathbf{S}_\ell^\top \mathbf{p}_\ell^A \quad (49)$$

Next, we expand all of the terms in (48), dropping any terms that do not explicitly depend on  $\ddot{\mathbf{y}}_\ell$ ,

$$\begin{aligned} \min_{\ddot{\mathbf{y}}_\ell} \quad & \mathbf{a}_k^\top {}^\ell \mathbf{X}_k^\top \mathbf{U}_\ell \ddot{\mathbf{y}}_\ell + \frac{1}{2} \ddot{\mathbf{y}}_\ell^\top \mathbf{D}_\ell \ddot{\mathbf{y}}_\ell + \mathbf{c}_\ell^\top \mathbf{U}_\ell \ddot{\mathbf{y}}_\ell \\ & - \boldsymbol{\tau}_\ell^\top (\mathbf{T}_\ell^a)^\top \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + (\mathbf{p}_\ell^A)^\top \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell \end{aligned} \quad (50)$$

Applying the first order optimality conditions for  $\ddot{\mathbf{y}}_\ell$ ,

$$\mathbf{0} = \mathbf{D}_\ell \ddot{\mathbf{y}}_\ell + \mathbf{U}_\ell^\top {}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{U}_\ell^\top \mathbf{c}_\ell - \mathbf{S}_\ell^\top \mathbf{T}_\ell^a \boldsymbol{\tau}_\ell + \mathbf{S}_\ell^\top \mathbf{p}_\ell^A, \quad (51)$$

the following recursive relationship for  $\ddot{\mathbf{y}}_\ell$  emerges

$$\begin{aligned} \ddot{\mathbf{y}}_\ell &= \mathbf{D}_\ell^{-1} (\boldsymbol{\tau}_\ell - \mathbf{U}_\ell^\top {}^\ell \mathbf{X}_k \mathbf{a}_k - \mathbf{U}_\ell^\top \mathbf{c}_\ell - \mathbf{S}_\ell^\top \mathbf{p}_\ell^A) \\ &= \mathbf{D}_\ell^{-1} (\boldsymbol{\tau}_\ell - \mathbf{S}_\ell^\top \mathbf{p}_\ell^A - \mathbf{U}_\ell^\top ({}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{c}_\ell)) \\ &= \mathbf{D}_\ell^{-1} (\mathbf{u}_\ell - \mathbf{U}_\ell^\top ({}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{c}_\ell)). \end{aligned} \quad (52)$$

This relationship has the same form as the original ABA.

Lastly, we derive the recursive relationship for the backward pass by considering the new force across the cluster joint  $\mathcal{J}_k$  that results from joining child clusters  $\ell$  and cluster  $k$ ,

$$\mathbf{f}_k^\mathcal{J} = \mathbf{I}_k^A \mathbf{a}_k + \mathbf{p}_k^A + {}^k \mathbf{X}_\ell^* (\mathbf{I}_\ell^A \mathbf{a}_\ell + \mathbf{p}_\ell^A). \quad (53)$$

Expressing  $\mathbf{a}_\ell$  in terms of (52) and arranging terms, we have

$$\begin{aligned} \mathbf{f}_k^\mathcal{J} &= (\mathbf{I}_k^A + {}^k \mathbf{X}_\ell^* (\mathbf{I}_\ell^A - \mathbf{U}_\ell \mathbf{D}_\ell^{-1} \mathbf{U}_\ell^\top) {}^\ell \mathbf{X}_k) \mathbf{a}_k \\ &\quad + \mathbf{p}_k^A + {}^k \mathbf{X}_\ell^* (\mathbf{p}_\ell^A + \mathbf{I}_\ell^A \mathbf{c}_\ell + \mathbf{U}_\ell \mathbf{D}_\ell^{-1} \mathbf{u}_\ell) \end{aligned} \quad (54)$$

Noticing that (54) has the form  $\mathbf{f}_k^\mathcal{J} = \mathbf{I}^A \mathbf{a} + \mathbf{p}^A$ , we have arrived at the recursive relationship for the articulated-cluster

inertia and bias force, which are analogous to the original articulated-body inertia relationships

$$\mathbf{I}_k^A = \mathbf{I}_k + \sum_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)} {}^k \mathbf{X}_\ell^* \mathbf{I}_\ell^A {}^\ell \mathbf{X}_k, \quad \mathbf{p}_k^A = \mathbf{p}_k + \sum_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)} {}^k \mathbf{X}_\ell^* \mathbf{p}_\ell^A, \quad (55)$$

where

$$\begin{aligned} \mathbf{I}_\ell^a &= \mathbf{I}_\ell^A - \mathbf{U}_\ell \mathbf{D}_\ell^{-1} \mathbf{U}_\ell^\top, \\ \mathbf{p}_\ell^a &= \mathbf{p}_\ell^A + \mathbf{I}_\ell^a \mathbf{c}_\ell + \mathbf{U}_\ell \mathbf{D}_\ell^{-1} (\boldsymbol{\tau}_\ell - \mathbf{S}_\ell^\top \mathbf{p}_\ell^A). \end{aligned} \quad (56)$$

Finally, by plugging in (52) to (42), after substantial algebra, one will find:

$$\begin{aligned} V_k^*(\mathbf{a}_k) &= \frac{1}{2} \|\mathbf{a}_k^{\text{uc}} - \mathbf{a}_k\|_{\mathbf{I}_k^A}^2 + \sum_{\mathcal{C}_\ell \in \mu(\mathcal{C}_k)} V_\ell^*({}^\ell \mathbf{X}_k \mathbf{a}_k + \mathbf{S}_\ell \ddot{\mathbf{y}}_\ell + \dot{\mathbf{S}}_\ell \dot{\mathbf{y}}_\ell) \\ &= \frac{1}{2} \|\mathbf{a}_k^{\text{uc}} - \mathbf{a}_k\|_{\mathbf{I}_k^A}^2 + \text{constant} \end{aligned}$$

which finally proves our inductive hypothesis, providing the correctness of this form of  $V_k^*(\mathbf{a}_k)$  for all bodies.

### C. Cluster-Based Articulated Body Algorithm

Having derived all of the necessary recursive relationships, we now present the full C-ABA for computing the forward dynamics of a robot with closure constraints, shown in Algorithm 1. While the basic structure of the C-ABA matches the structure of the conventional ABA, there are some important differences to be observed. For example, lines 4-7 and 14-16 show the nested for loops within the forward pass that are not present in the conventional ABA. They are necessary because each iteration of the forward pass deals with a group of bodies rather than an individual body. Furthermore, the need to compute the spatial transform that remaps parent-child relationships (line 10) and the more complex derivative of the motion subspace matrix (lines 17 and 18) are present only in the C-ABA.

Despite these added complexities, the C-ABA has a computational complexity of  $O(N_C l^3)$  when neglecting the cost associated with computing constraint Jacobians and biases. For robots with many “local” closure constraints (i.e.,  $N_C \approx N_B/l$  and small  $l$ ), the C-ABA offers a significant reduction in computational complexity compared to the state-of-the-art Lagrange Multiplier-based approach for computing constrained forward dynamics [12]. The Lagrange Multiplier approach has  $O(N_B + d c^2 + c^3)$  complexity, where  $d$  is the depth of the system’s connectivity tree and  $c$  is the number of closure constraints. Many relevant actuation sub-mechanisms have small  $l$ , e.g. geared motors ( $l = 2$ ), four-bar linkages ( $l = 3$ , differential drives ( $l = 4$ ), and parallel belt transmissions ( $l = 4$ ).

### VIII. INVERSE OPERATIONAL-SPACE INERTIA MATRIX

This section derives the C-EFPA, which efficiently computes the inverse OSIM for one or more end-effectors while accounting for the constraints imposed by actuation sub-mechanisms. For the complete derivation of the original EFPA, please see [13]. The purpose of this section is to prove that the generalization of the algorithm to C-CT’s is possible and to highlight the nuanced differences that occur when dealing with clusters versus individual rigid bodies.



---

**Algorithm 1** Cluster-based Articulated-Body Algorithm
 

---

```

1: Forward Pass #1
2:  $\mathbf{v}_0 = \mathbf{0}$ 
3: for  $k = 1$  to  $N_C$  do
4:   for  $i = 1$  to  $n_b(\mathcal{C}_k)$  do
5:      $[\mathbf{X}_{J(i)}, \mathbf{S}_i, \frac{d}{dt} \mathbf{S}_i] = \text{jcalc}(\text{jtype}(i), \mathbf{q}_i, \dot{\mathbf{q}}_i)$ 
6:      ${}^i\mathbf{X}_{\lambda(i)} = \mathbf{X}_{J(i)}\mathbf{X}_{T(i)}$ 
7:   end for
8:    $\bar{\mathbf{S}}_k = \text{block}(\{\mathbf{S}_i\}_{i \in \mathcal{C}_k}), \dot{\bar{\mathbf{S}}}_k = \text{block}(\{\frac{d}{dt} \mathbf{S}_i\}_{i \in \mathcal{C}_k})$ 
9:   Compute  ${}^i\mathbf{X}_{\lambda(i)}$  using  ${}^i\mathbf{X}_{\lambda(i)}$  via (17)
10:  Compute  ${}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}$  using  ${}^i\mathbf{X}_{\lambda(i)}$  via (20)
11:   $[\mathbf{G}_k, \mathbf{g}_k] = \text{updateConstraints}(\mathbf{q}_k, \dot{\mathbf{q}}_k)$ 
12:   $\mathbf{S}_k = {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}\bar{\mathbf{S}}_k$ 
13:   $\mathbf{v}_k = {}^k\mathbf{X}_{\lambda(k)}\mathbf{v}_{\lambda(k)} + \mathbf{S}_k\dot{\mathbf{y}}_k$ 
14:  for  $i = 1$  to  $n_b(\mathcal{C}_k)$  do
15:     $\frac{d}{dt} ({}^i\mathbf{X}_{\lambda(i)}) = ({}^i\mathbf{X}_{\lambda(i)}\mathbf{v}_{\lambda(i)} - \mathbf{v}_i) \times {}^i\mathbf{X}_{\lambda(i)}$ 
16:  end for
17:  Compute  $\frac{d}{dt} {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}$  using  $\frac{d}{dt} ({}^i\mathbf{X}_{\lambda(i)})$ 
18:   $\mathbf{c}_k = \left( \frac{d}{dt} ({}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}) \bar{\mathbf{S}}_k + \mathbf{X}_{I(k)}\dot{\bar{\mathbf{S}}}_k \right) \mathbf{G}_k\mathbf{y}_k$ 
19:     $+ {}^{\Lambda(k)}\mathbf{X}_{\lambda(k)}\bar{\mathbf{S}}_k\mathbf{g}_k$ 
20:   $\mathbf{l}_k^A = \mathbf{l}_k$ 
21:   $\mathbf{p}_k^A = \mathbf{v}_k \times^* \mathbf{l}_k\mathbf{v}_k$ 
22: end for
23:
24: Backward Pass
25: for  $k = N_C$  to 1 do
26:    $\mathbf{U}_k = \mathbf{l}_k^A\mathbf{S}_k$ 
27:    $\mathbf{D}_k = \mathbf{S}_k^\top \mathbf{U}_k$ 
28:    $\mathbf{u}_k = \boldsymbol{\tau}_k - \mathbf{S}_k^\top \mathbf{p}_k^A$ 
29:   if  $\lambda(k) \neq 0$  then
30:      $\mathbf{l}^a = \mathbf{l}_k^A - \mathbf{U}_k\mathbf{D}_k^{-1}\mathbf{U}_k^\top$ 
31:      $\mathbf{p}^a = \mathbf{p}_k^A + \mathbf{l}^a\mathbf{c}_k + \mathbf{U}_k\mathbf{D}_k^{-1}\mathbf{u}_k$ 
32:      $\mathbf{l}_{\lambda(k)}^A = \mathbf{l}_{\lambda(k)}^A + {}^{\lambda(k)}\mathbf{X}_k^* \mathbf{l}^a {}^k\mathbf{X}_{\lambda(k)}$ 
33:      $\mathbf{p}_{\lambda(k)}^A = \mathbf{p}_{\lambda(k)}^A + {}^{\lambda(k)}\mathbf{X}_k^* \mathbf{p}^a$ 
34:   end if
35: end for
36:
37: Forward Pass #2
38:  $\mathbf{a}_0 = -\mathbf{a}_g$ 
39: for  $k = 1$  to  $N_C$  do
40:    $\mathbf{a}' = {}^k\mathbf{X}_{\lambda(k)}\mathbf{a}_{\lambda(k)} + \mathbf{c}_k$ 
41:    $\ddot{\mathbf{y}}_k = \mathbf{D}_k^{-1}(\mathbf{u}_k - \mathbf{U}_k^\top \mathbf{a}')$ 
42:    $\mathbf{a}_k = \mathbf{a}' + \mathbf{S}_k\ddot{\mathbf{y}}_k$ 
43: end for

```

---

### A. End-Effector Connectivity

Dealing with end-effectors requires a minor extension of the connectivity graph representation in Sec IV. First, for a system with  $N_B$  rigid bodies and  $N_{ee}$  end-effectors, the end-effectors will be numbered from  $N_B + 1$  to  $N_B + N_{ee}$ . To distinguish end-effectors from rigid bodies and clusters, we will use the index  $e$  to refer to end-effectors exclusively. The previous definitions for  $\lambda(\cdot)$  and  $\kappa(\cdot)$  also apply to end-effectors. For example,  $\lambda(e)$  refers to the parent body of the  $e$ -th end-effector. We must also introduce a new definition that

denotes the set of end-effectors supported (ES) by a given cluster joint in the tree:

$$ES(\mathcal{C}_k) = \{e : e > N_B \text{ and } \exists B_i \in \mathcal{C}_k \cap \kappa(e)\}.$$

### B. Force Propagators

The main feature of the EFPA is the recursive calculation and re-use of the extended force propagator matrices  ${}^e\mathbf{X}_i^\top$ . Unlike standard spatial transforms  ${}^i\mathbf{X}_e^* = {}^e\mathbf{X}_i^\top$ , force propagators provide an *articulated* transform of a spatial force from an end-effector to any body in the system. Articulated transforms differ from standard spatial force transforms in that they transform the force from end-effector to body as if the joints of the system are free to move rather than locked in place. As such, these transforms capture the effect that a force on an arbitrary end-effector has on any earlier body that supports it.

Identically to [13], we initialize the propagators for each of the end-effector bodies with  ${}^e\mathbf{X}_{\lambda(e)} = {}^e\mathbf{X}_{\lambda(e)}$ . These individual rigid-body propagators are then used to initialize the cluster propagators via

$${}^e\mathbf{X}_k = \begin{bmatrix} \sigma_e(1){}^e\mathbf{X}_{\mathcal{C}_k(1)} \\ \vdots \\ \sigma_e(n){}^e\mathbf{X}_{\mathcal{C}_k(n)} \end{bmatrix} \quad (57)$$

where  $\mathcal{C}_k$  is the cluster containing  $\lambda(e)$  and  $\sigma_e(i)$  is an indicator function the encodes whether the  $i$ -th body of  $\mathcal{C}_k$  is the parent body of the end effector,

$$\sigma_e(i) = \begin{cases} 1, & \text{if } \lambda(e) = \mathcal{C}_k(i) \\ 0, & \text{otherwise} \end{cases}. \quad (58)$$

The backward pass of the algorithm, derived in Sec VIII-C, details how, starting with these cluster propagators at the leaves of the tree, we can perform an inward recursion to compute the remaining articulated transforms for the rest of the clusters.

### C. Backward Pass

The EFPA derivation is based on the ABA derivation, and likewise for the C-EFPA and the C-ABA. An important difference between computing forward dynamics versus computing the inverse OSIM is that when computing the inverse OSIM, we assume zero joint torques and zero velocities. So in the backward pass of the C-EFPA, we again aim to propagate (38) from the leaves to the base, but the key difference is the initialization of the articulated bias forces for each cluster. For the C-EFPA, instead of initializing each cluster with

$$\mathbf{p}_k = \mathbf{v}_k \times^* \mathbf{l}_k\mathbf{v}_k,$$

we initialize the parent clusters of the end-effectors with

$$\mathbf{p}_k = - \sum_{e \in ES(\mathcal{C}_k)} {}^e\mathbf{X}_k^\top \mathbf{f}_e, \quad (59)$$

where  $\mathbf{f}_e$  represents the force that acts at end-effector  $e$ , and we initialize the rest of the clusters with  $\mathbf{p}_k = \mathbf{0}$ .

Following the same steps from Section VII that we used to arrive at (55) and (56), we can derive the recursive relationship

needed to compute the articulated bias force for the rest of the clusters in the C-CT. However, the assumption of zero torques and velocities allows us to make the following simplification to (56),

$$\mathbf{p}_\ell^a = \mathbf{p}_\ell^A - \mathbf{U}_\ell \mathbf{D}_\ell^{-1} \mathbf{S}_\ell^\top \mathbf{p}_\ell^A. \quad (60)$$

Now consider an arbitrary cluster  $k$  with at least one end-effector as its child. Using (55) and (60), we can propagate (38) to cluster  $\lambda(k)$  via

$$\mathbf{f}_{\lambda(k)}^\mathcal{J} = \left( \mathbf{I}_{\lambda(k)} + \sum_{\mathcal{C}_\ell \in \mu(\lambda(\mathcal{C}_k))} \lambda^{(k)} \mathbf{X}_\ell^* \mathbf{L}_\ell^\top \mathbf{I}_\ell^A \mathbf{X}_{\lambda(k)} \right) \mathbf{a}_{\lambda(k)} - \sum_{\mathcal{C}_\ell \in \mu(\lambda(\mathcal{C}_k))} \sum_{e \in ES(\mathcal{C}_\ell)} \lambda^{(k)} \mathbf{X}_\ell^* \mathbf{L}_\ell^\top d \mathbf{X}_\ell^\top \mathbf{f}_e, \quad (61)$$

where each  $\mathbf{L}_\ell^\top$  is a force propagator across the  $l$ -th cluster joint and takes the form

$$\mathbf{L}_\ell^\top = \mathbf{1}_{6n_b(\mathcal{C}_\ell) \times 6n_b(\mathcal{C}_\ell)} - \mathbf{U}_\ell \mathbf{D}_\ell^{-1} \mathbf{S}_\ell^\top.$$

Thus, we arrive at the cluster-based analog of the propagator recursion originally derived in [13]:

$${}^e \chi_{\lambda(k)} := {}^e \chi_k \mathbf{L}_k^k \mathbf{X}_{\lambda(k)}. \quad (62)$$

#### D. Forward Pass

The forward pass of the C-EFPA closely resembles the forward pass of the original EFPA. The goal is to produce the combined force-acceleration relationship  $\mathbf{a} = \boldsymbol{\Omega}^{-1} \mathbf{f}$  for each end-effector. This is accomplished by propagating the following equation from the base to the leaves of the C-CT:

$$\mathbf{a}_k = \sum_{e=N_B+1}^{N_B+N_{ee}} \boldsymbol{\Omega}_{ke}^{-1} \mathbf{f}_e, \quad (63)$$

where  $\boldsymbol{\Omega}_{ke}^{-1}$  relates the acceleration resulting at cluster  $k$  to the end-effector force  $\mathbf{f}_e$ . Similar to the other forward passes, the fixed base provides the initial value  $\boldsymbol{\Omega}_{0e}^{-1} = \mathbf{0}$  for all end-effectors.

We can use the same steps to get  $\ddot{\mathbf{y}}_k$  in terms of  $\mathbf{a}_{\lambda(k)}$ , simplifying under the assumption of no torques or velocities:

$$\ddot{\mathbf{y}}_k = -\mathbf{D}_k^{-1} (\mathbf{S}_k^\top \mathbf{p}_k^A + \mathbf{U}_k^k \mathbf{X}_{\lambda(k)} \mathbf{a}_{\lambda(k)}). \quad (64)$$

Substituting this into (33) and recalling the modification from (59) results in

$$\begin{aligned} \mathbf{a}_k &= {}^k \mathbf{X}_{\lambda(k)} \mathbf{a}_{\lambda(k)} - \mathbf{S}_k \mathbf{D}_k^{-1} \mathbf{U}_k^k \mathbf{X}_{\lambda(k)} \mathbf{a}_{\lambda(k)} - \mathbf{S}_k \mathbf{D}_k^{-1} \mathbf{S}_k^\top \mathbf{p}_k^A \\ &= \mathbf{L}_k^k \mathbf{X}_{\lambda(k)} \mathbf{a}_{\lambda(k)} + \mathbf{S}_k \mathbf{D}_k^{-1} \mathbf{S}_k^\top \sum_{e \in ES(\mathcal{C}_k)} {}^e \chi_k^\top \mathbf{f}_e. \end{aligned} \quad (65)$$

Expressing the parent acceleration in the form of (63) leads to

$$\begin{aligned} \mathbf{a}_k &= \sum_{e \notin ES(k)} \mathbf{L}_k^k \mathbf{X}_{\lambda(k)} \boldsymbol{\Omega}_{\lambda(k)e}^{-1} \mathbf{f}_e + \\ &\quad \sum_{e \in ES(\mathcal{C}_k)} \left( \mathbf{L}_k^k \mathbf{X}_{\lambda(k)} \boldsymbol{\Omega}_{\lambda(k)e}^{-1} + \mathbf{S}_k \mathbf{D}_k^{-1} \mathbf{S}_k^\top {}^e \chi_k^\top \right) \mathbf{f}_e, \end{aligned} \quad (66)$$

which yields the recursion that forms the basis of the forward pass

$$\boldsymbol{\Omega}_{ke}^{-1} := \begin{cases} \mathbf{L}_k^k \mathbf{X}_{\lambda(k)} \boldsymbol{\Omega}_{\lambda(k)e}^{-1} + \mathbf{S}_k \mathbf{D}_k^{-1} \mathbf{S}_k^\top {}^e \chi_k^\top, & \text{if } e \in ES(\mathcal{C}_k) \\ \mathbf{L}_k^k \mathbf{X}_{\lambda(k)} \boldsymbol{\Omega}_{\lambda(k)e}^{-1}, & \text{otherwise} \end{cases}. \quad (67)$$

The remainder of the algorithm follows straightforwardly from [13], including the computational savings of computing  $\boldsymbol{\Omega}_{e_1 e_2}^{-1}$  at the greatest common ancestor. In this case, that ancestor is a cluster rather than an individual rigid body.

#### E. Cluster-Based Extended Force Propagator Algorithm

The conventional EFPA is closely related to the conventional ABA, and that similarity is preserved between the C-EFPA and the C-ABA. The most notable difference is that, similar to how the concept of spatial transforms was generalized to work with clusters of bodies in Sec. V, deriving the C-EFPA required the concept of force propagators to be generalized for clusters of bodies as well. Again neglecting the cost associated with computing constraint Jacobians, the EFPA has a computational complexity of  $\mathcal{O}(N_C l^3 + N_{ee} d + N_{ee}^2)$  where  $d$  is the depth of the C-CT. This is a significant improvement over the Projection method for computing the inverse OSIM, which is  $\mathcal{O}(N_B^3 N_{ee})$ . The method in [18] offers a more efficient algorithm for computing the inverse OSIM for open-chain systems but also relies heavily on force propagators. This method could be combined with the clustering strategies herein for application to systems with internal loop constraints.

## IX. RESULTS

In this section, we will demonstrate how the performance of our generalized algorithms compares to existing algorithms, both in terms of accuracy and speed. We test the algorithms on six different robots: the 24 DoF MIT Humanoid [1], the 24 DoF Tello Humanoid [2], the 18 DoF MIT Mini Cheetah [37], a modified version of the 38 DoF JVRC-1 Humanoid [38], a 12 DoF serial chain of links actuated with geared transmissions (GT), and a 12 DoF serial chain of links actuated by parallel belt transmissions (PBT). The JVRC-1 robot is modified so all its joints are actuated with geared transmissions. We chose this robot in particular because (i) its design, in terms of number and arrangement of degrees of freedom, is similar to many state-of-the-art humanoid robots, (ii) its design is open source [43], and (iii) it has numerous degrees of freedom and therefore shows how the cluster-based algorithms scale well. The robots and their respective connectivity graphs are shown in Fig 8.

The benchmarking code is available at [44]. The results presented in this paper were gathered on a Macbook Pro laptop with an M1 Pro chip, where the code was compiled with Clang 14.0. We recognize that the computation speed results presented in this work are significantly slower than in comparable libraries such as Pinocchio [45]. For example, for robots with identical connectivity (HyQ [46] and MIT Mini Cheetah [37]) ABA requires about 3  $\mu\text{s}$  in Pinocchio and about 20  $\mu\text{s}$  with our library. Thus, our results are focused on presenting, to the best of our ability, a fair *relative* comparison

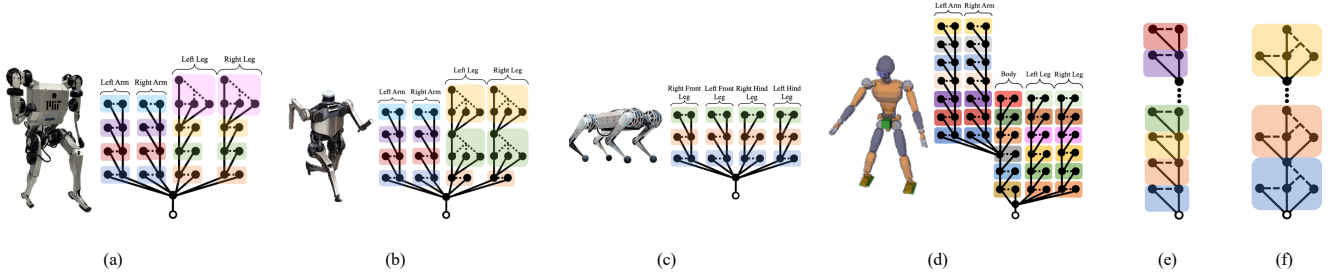


Fig. 8. The proposed algorithms are benchmarked on a variety of robots, including (a) the MIT Humanoid, (b) the UIUC Tello Humanoid, (c) the MIT Mini Cheetah, and (d) the JVRC-1 Humanoid, (e) a 12 DoF serial chain of links actuated with geared transmissions, and (f) 12 DoF serial chain of links actuated by parallel belt transmissions. The RB-CGs are shown for each robot, with the colored boxes used to show clustered bodies.

of our algorithms with state-of-the-art. To reduce the absolute times of the algorithms, we could implement techniques suggested by the Pinocchio developers [45] such as static polymorphism via Curiously Repeating Template Patterns and code generation. This, however, is left as future work.

#### A. Speed Comparison

1) *Inverse Dynamics*: We first compare the performance of our algorithm for computing constrained inverse dynamics to the alternative approaches described in Sec. II-C. These results are especially relevant in whole-body control, which relies on accurate and efficient inverse dynamics. The benchmark in Fig. 9 shows that the C-RNEA consistently outperforms the Projected RNEA for all robots considered in this work. The relative speed of the alternative algorithms compared to the C-RNEA is shown in Table II. The Projected RNEA is 22 – 38% slower than the C-RNEA, depending on the robot. The approximate RNEA, on the other hand, is 36 – 60% faster than C-RNEA depending on the robot. However, the approximate algorithm sacrifices accuracy. This trade-off between accuracy and speed is analyzed in Sec. IX-B.

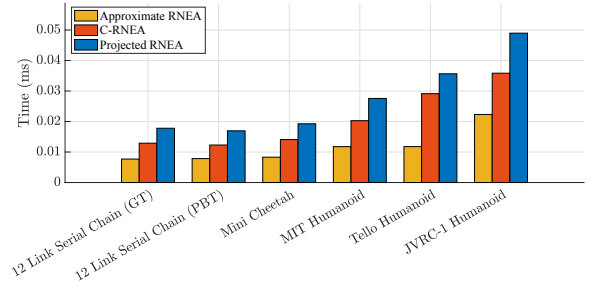


Fig. 9. Speed performance of the various algorithms for computing constrained inverse dynamics.

TABLE III  
PERCENT REDUCTION/ADDITION OF COMPUTATION TIME COMPARED TO C-ABA

	12 Link GT Chain	12 Link PBT Chain	Mini Cheetah	MIT Humanoid	Tello Humanoid	JVRC-1
Approximate ABA	-34.1	-34.6	-35.3	-38.6	-49.4	-38.4
Projection Method	+86.8	+73.5	+21.0	+32.5	+22.2	+87.3
Lagrange Multiplier Method	+133.1	+119.8	+76.3	+106.2	+82.7	+212.8

TABLE II  
PERCENT REDUCTION/ADDITION OF COMPUTATION TIME COMPARED TO C-RNEA

	12 Link GT Chain	12 Link PBT Chain	Mini Cheetah	MIT Humanoid	Tello Humanoid	JVRC-1
Approximate RNEA	-40.3	-36.2	-41.0	-42.0	-59.6	-37.7
Projected RNEA	+38.0	+37.8	+36.7	+35.9	+22.4	+36.6

2) *Forward Dynamics*: Accurate and efficient computation of a robot’s constrained forward dynamics is crucial not only for simulation but also for control strategies that involve planning over a horizon by rolling out the robot’s dynamics (e.g., Differential Dynamic Programming [47]). We compare our algorithm for computing constrained forward dynamics with the alternative approaches described in Sec. II-D. The results in Fig. 10 show that our algorithm outperforms the Projection and Lagrange Multiplier methods for all of the robots we considered. As shown in Table III, the Projection Method is 21 – 87% slower while the Lagrange Multiplier Method is 76 – 212% slower depending on the robot.

We further demonstrate the favorable complexity of algorithms with respect to the number of the robot’s degrees

of freedom by comparing the time required to compute for constrained forward dynamics of serial chains with varying numbers of links. In Fig. 11, we vary the number of links in the respective chains from 2 to 24. The results demonstrate that while the Projection and Lagrange Multiplier Methods scale cubically with the number of degrees of freedom of the chain, the C-ABA scales linearly while still obtaining the exact solution.

The Approximate ABA, however, is 34 – 49% faster than the C-ABA depending on the robot. This speed improvement comes at the cost of accuracy, though. We investigate the trade-off further in Sec. IX-B.

3) *Contact Dynamics*: Lastly, we benchmark the performance of the C-EFPA against other methods for computing the inverse OSIM. The results in Fig. 12 again demonstrate that our proposed algorithm outperforms the Projection Method for all the robots we considered. Table IV shows that the Projection Method is 6 – 107% slower than the C-EFPA.

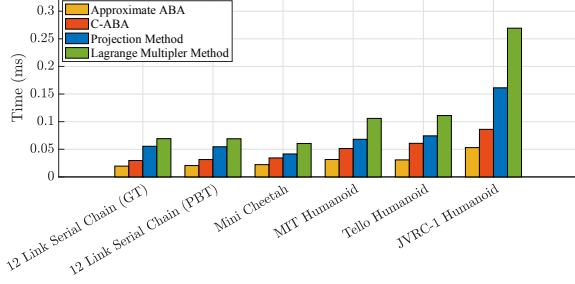


Fig. 10. Speed performance of the various algorithms for computing constrained forward dynamics.

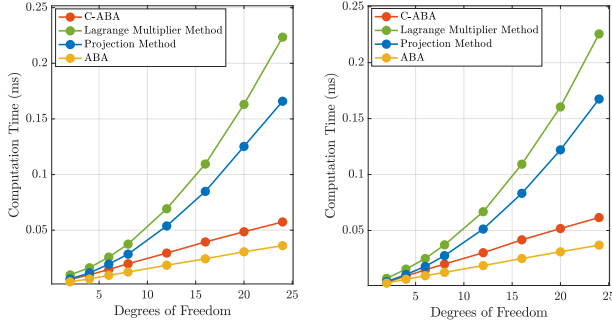


Fig. 11. Time required to compute the constrained forward dynamics of a serial chain of links actuated by (Left) geared motors and (Right) parallel belt transmissions.

In the case of the 12 link serial chain actuated by parallel belt transmissions and the Tello Humanoid, the speedups afforded by the C-EFPA relative to the Projection Method are small. This is because of the favorable sparsity pattern of these systems (see Fig. 8), which allows efficient factorization of the mass matrix. However, as we demonstrate in Fig 13, the sparse factorization method scales cubically in the number of links, while the C-EFPA scales linearly and thus eventually performs better as the number of links increases.

### B. Accuracy Comparison

While the approximate and unconstrained algorithms in Sec. IX-A1-IX-A3 can be significantly faster than even their cluster-based analogs, they can also significantly reduce the accuracy compared to the exact algorithms. To demonstrate this trade-off, we performed the following numerical experi-

TABLE IV  
PERCENT REDUCTION/ADDITION OF COMPUTATION TIME COMPARED TO C-EFPA

	12 Link GT Chain	12 Link PBT Chain	Mini Cheetah	MIT Humanoid	Tello Humanoid	JVRC-1
Approximate EFPA	-35.7	-52.1	-28.6	-37.4	-51.4	-22.54
Projected EFPA	+51.8	+6.3	+19.0	+23.2	+7.80	+107.4

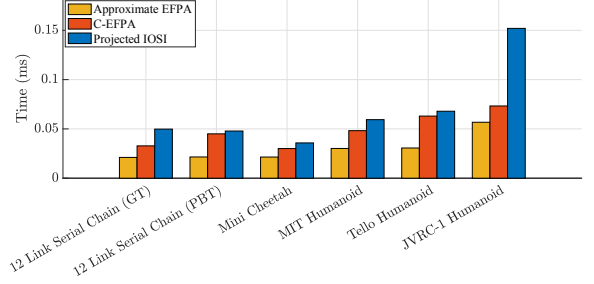


Fig. 12. Speed performance of the various algorithms for computing constrained inverse OSIM.

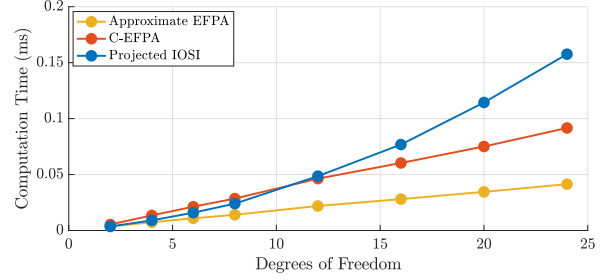


Fig. 13. Time required to compute the constrained inverse OSIM of a serial chain of links actuated by parallel belt transmissions.

ment using one of the legs of the MIT Humanoid, shown in Fig. 1. We prescribe sinusoidal trajectories

$$\begin{aligned} \mathbf{q}(t) &= A \sin(2\pi\omega t), \\ \dot{\mathbf{q}}(t) &= 2A\pi\omega \cos(2\pi\omega t), \\ \ddot{\mathbf{q}}(t) &= -4A\pi^2\omega^2 \sin(2\pi\omega t) \end{aligned}$$

for each of the joints of the leg. Every  $\Delta t$  seconds, we use exact, approximate, and unconstrained inverse dynamic algorithms to compute the torques required to induce the desired acceleration. This results in discrete torque trajectories corresponding to the unconstrained algorithm,  $\tau_{1:n}^{\text{unc}}$ , the approximate algorithm,  $\tau_{1:n}^{\text{approx}}$ , and the exact, cluster-based algorithm,  $\tau_{1:n}^{\text{cluster}}$ , where  $n$  is the number of timesteps. The results in Fig. 14 show the difference in the required torque computed by the various algorithms.

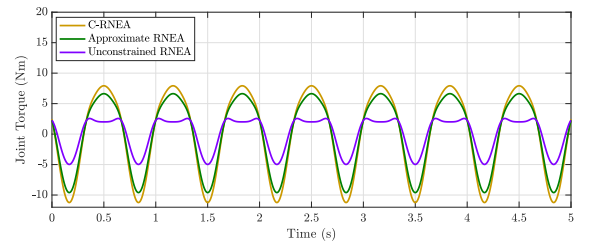


Fig. 14. Comparison of inverse dynamics-based torque profiles required to achieve sinusoidal trajectories of the MIT Humanoid's ankle joint with  $A = 0.5 \text{ rad}$ ,  $\omega = 1.5 \text{ rad s}^{-1}$ .

The results of this numerical experiment validate that neglecting or even approximating the effects of loop constraints arising from actuation sub-mechanisms can lead to significant errors. For example, the results in Fig. 14 show that using the unconstrained inverse dynamics to produce the feedforward torque trajectory for the ankle joint yields a root mean square of 4.31 N m relative to the exact algorithm. The approximate algorithm, which requires only a negligible amount of additional computation, reduces that root mean square error to 1.04 N m. These errors are 39.2% and 9.5% of the maximum torque, respectively. We chose to focus on the ankle joint in this experiment because its couplings with the ankle rotor joint, knee joint, and knee rotor joint are especially difficult for approximate algorithms to account for.

## X. DISCUSSION

The main advantage of the algorithms we have presented in this work is that they maintain the recursive structure of the original algorithms that they generalize. Consequently, they scale well with the number of degrees of freedom of the robot. A further advantage of our library of algorithms is that the loop constraints imposed by the actuation sub-mechanisms are handled within the rigid-body dynamics algorithms rather than separately as part of a time-stepping scheme (with, e.g., Baumgaurte stabilization or viscoelastic joint constraints). This makes implementing our library simple since the only requirement on the part of the user is to specify the sub-mechanisms comprising their system. Therefore, any application requiring accurate and efficient computation of rigid-body dynamics, e.g., model-based controllers and parallelized simulation for reinforcement learning, should be able to adopt these algorithms easily.

## XI. CONCLUSION

In this paper, we have presented a library of generalized algorithms for computing constrained rigid-body dynamics. We first described the concept of clustering rigid bodies involved in kinematic loops. Then, we developed a novel generalization of SVA to describe the dynamics of these clusters of bodies. The critical insight behind SVA for clusters is the concept of output-relative quantities, which permit recurrence relationships between clusters. This generalization of SVA provides the mathematical framework upon which subsequent recursive algorithms are derived. Algorithms derived in this work include the C-RNEA for inverse dynamics, the C-ABA for forward dynamics, and the C-EFPA for the inverse OSIM. While previous work had presented the C-ABA with LCE [42], we provided a self-contained derivation based on the Gauss principle of least constraint that offers physical meaning for the intermediate quantities of the algorithm. We presented numerical results demonstrating that our algorithms outperform comparable state-of-the-art algorithms.

The clustering and SVA-based framework we present in this work can be used to generalize many algorithms beyond those derived in this work. Prominent examples include algorithms for computing the Coriolis matrix and Christoffel symbols for rigid-body systems [48] as well as for characterizing the space

of identifiable inertial parameters in system identification [49]. Furthermore, recent research in optimization-based control has generated interest in computing not only the robot's forward and inverse dynamics but also their derivatives with respect to state and control inputs [14], [15], [50]. Generalizing these algorithms involves re-deriving them such that all recursive relationships are output relative rather than parent relative and then resolving any of the complications that arise, which is left as future work.

## ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) Graduate Research Fellowship Program under Grant No. 4000092301 and under NSF award CMMI-2220924 with a subaward to the University of Notre Dame. The authors would also like to thank Youngwoo Sim from the University of Illinois Urbana-Champaign for his assistance with CAD files and design questions related to the Tello humanoid robot.

## REFERENCES

- [1] M. Chignoli, D. Kim, E. Stanger-Jones, and S. Kim, "The mit humanoid robot: Design, motion planning, and control for acrobatic behaviors," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pp. 1–8, IEEE, 2021.
- [2] Y. Sim and J. Ramos, "Tello leg: The study of design principles and metrics for dynamic humanoid robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 9318–9325, 2022.
- [3] Y. Liu, J. Shen, J. Zhang, X. Zhang, T. Zhu, and D. Hong, "Design and control of a miniature bipedal robot with proprioceptive actuation for dynamic behaviors," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 8547–8553, IEEE, 2022.
- [4] "Digit." <https://robotsguide.com/robots/cassie>, 2023.
- [5] "Kangaroo." <https://pal-robotics.com/robots/kangaroo/>, 2023.
- [6] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *The international journal of robotics research*, vol. 2, no. 1, pp. 13–30, 1983.
- [7] D. E. Orin, R. McGhee, M. Vukobratović, and G. Hartoch, "Kinematic and kinetic analysis of open-chain linkages utilizing newton-euler methods," *Mathematical Biosciences*, vol. 43, no. 1-2, pp. 107–130, 1979.
- [8] A. Vereshchagin, "Computer simulation of the dynamics of complicated mechanisms of robot-manipulators," *Eng. Cybernet.*, vol. 12, pp. 65–70, 1974.
- [9] D. Bertsekas, *Dynamic programming and optimal control: Volume I*, vol. 4. Athena scientific, 2012.
- [10] C. F. Gauß, "Über ein neues allgemeines grundgesetz der mechanik," 1829.
- [11] R. Featherstone, "A beginner's guide to 6-d vectors (part 1)," *IEEE robotics & automation magazine*, vol. 17, no. 3, pp. 83–94, 2010.
- [12] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.
- [13] P. Wensing, R. Featherstone, and D. E. Orin, "A reduced-order recursive algorithm for the computation of the operational-space inertia matrix," in *2012 IEEE International Conference on Robotics and Automation*, pp. 4911–4917, IEEE, 2012.
- [14] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Robotics: Science and systems (RSS 2018)*, 2018.
- [15] S. Singh, R. P. Russell, and P. M. Wensing, "Efficient analytical derivatives of rigid-body dynamics using spatial vector algebra," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1776–1783, 2022.
- [16] A. Jain, "Unified formulation of dynamics for serial rigid multibody systems," *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 3, pp. 531–542, 1991.
- [17] A. Jain, *Robot and multibody dynamics: analysis and algorithms*. Springer Science & Business Media, 2010.
- [18] A. S. Sathya, W. Decre, and J. Swevers, "Pv-osimr: A lowest order complexity algorithm for computing the delassus matrix," 2023.
- [19] R. Featherstone, "Efficient factorization of the joint-space inertia matrix for branched kinematic trees," *The International Journal of Robotics Research*, vol. 24, no. 6, pp. 487–500, 2005.

- [20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.
- [21] L. Sciavicco, B. Siciliano, and L. Villani, "Lagrange and newton-euler dynamic modeling of a gear-driven robot manipulator with inclusion of motor inertia effects," *Advanced robotics*, vol. 10, no. 3, pp. 317–334, 1995.
- [22] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.
- [23] M. Becke and T. Schlegel, "Extended newton-euler based centrifugal/coriolis matrix factorization for geared serial robot manipulators with ideal joints," in *Proceedings of 15th International Conference MECHATRONIKA*, pp. 1–7, IEEE, 2012.
- [24] S. H. Murphy, J. T. Wen, and G. N. Saridis, "Recursive calculation of geared robot manipulator dynamics," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 839–844, IEEE, 1990.
- [25] A. Jain and G. Rodriguez, "Recursive dynamics for geared robot manipulators," in *29th IEEE Conference on Decision and Control*, pp. 1983–1988, IEEE, 1990.
- [26] R. Featherstone, "A divide-and-conquer articulated-body algorithm for parallel  $O(\log(n))$  calculation of rigid-body dynamics. part 1: Basic algorithm," *The International Journal of Robotics Research*, vol. 18, no. 9, pp. 867–875, 1999.
- [27] R. Featherstone, "A divide-and-conquer articulated-body algorithm for parallel  $O(\log(n))$  calculation of rigid-body dynamics. part 2: Trees, loops, and accuracy," *The International Journal of Robotics Research*, vol. 18, no. 9, pp. 876–892, 1999.
- [28] K. Yamane and Y. Nakamura, "Comparative study on serial and parallel forward dynamics algorithms for kinematic chains," *The International Journal of Robotics Research*, vol. 28, no. 5, pp. 622–629, 2009.
- [29] A. S. Sathya, H. Bruyninckx, W. Decre, and G. Pipeleers, "Efficient constrained dynamics algorithms based on an equivalent lqr formulation using gauss' principle of least constraint," 2023.
- [30] H. Brandl, "An algorithm for the simulation of multibody systems with kinematic loops," in *Proc. the IFToMM Seventh World Congress on the Theory of Machines and Mechanisms*, Sevilla, 1987.
- [31] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810, IEEE, 2018.
- [32] J. Critchley and K. S. Anderson, "A generalized recursive coordinate reduction method for multibody system dynamics," *International Journal for Multiscale Computational Engineering*, vol. 1, no. 2&3, 2003.
- [33] A. Jain, "Recursive algorithms using local constraint embedding for multibody system dynamics," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 49019, pp. 139–147, 2009.
- [34] A. Müller, "A constraint embedding approach for dynamics modeling of parallel kinematic manipulators with hybrid limbs," *Robotics and Autonomous Systems*, p. 104187, 2022.
- [35] S. Kumar, K. A. v. Szadkowski, A. Mueller, and F. Kirchner, "An analytical and modular software workbench for solving kinematics and dynamics of series-parallel hybrid robots," *Journal of Mechanisms and Robotics*, vol. 12, no. 2, 2020.
- [36] R. Kumar, S. Kumar, A. Müller, and F. Kirchner, "Modular and hybrid numerical-analytical approach-a case study on improving computational efficiency for series-parallel hybrid robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3476–3483, IEEE, 2022.
- [37] B. Katz, J. Di Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 international conference on robotics and automation (ICRA)*, pp. 6295–6301, IEEE, 2019.
- [38] M. Okugawa, K. Oogane, M. Shimizu, Y. Ohtsubo, T. Kimura, T. Takahashi, and S. Tadokoro, "Proposal of inspection and rescue tasks for tunnel disasters—task development of japan virtual robotics challenge," in *2015 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, pp. 1–2, IEEE, 2015.
- [39] "Eigen." <http://eigen.tuxfamily.org>.
- [40] R. Featherstone, "Exploiting sparsity in operational-space dynamics," *The International Journal of Robotics Research*, vol. 29, no. 10, pp. 1353–1368, 2010.
- [41] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.
- [42] A. Jain, "Multibody graph transformations and analysis, part ii: Closed-chain constraint embedding," *Nonlinear dynamics*, vol. 67, no. 4, p. 2153–2170, 2012.
- [43] "Jvrc-1." [https://github.com/robot-descriptions/jvrc\\_description/tree/master](https://github.com/robot-descriptions/jvrc_description/tree/master), 2023.
- [44] ROAM Lab, "Generalized rbda." [https://github.com/ROAM-Lab-ND/generalized\\_rbda](https://github.com/ROAM-Lab-ND/generalized_rbda), 2023. Commit hash: d763e829857fdec7d6bc5aab044a0490738a743.
- [45] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiroux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, pp. 614–619, IEEE, 2019.
- [46] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq—a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, 2011.
- [47] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.
- [48] S. Echeandia and P. M. Wensing, "Numerical methods to compute the coriolis matrix and christoffel symbols for rigid-body systems," *Journal of Computational and Nonlinear Dynamics*, vol. 16, no. 9, 2021.
- [49] P. M. Wensing, G. Niemeyer, and J.-J. E. Slotine, "A geometric characterization of observability in inertial parameter identification," *arXiv preprint arXiv:1711.03896*, 2023.
- [50] A. Jain and G. Rodriguez, "Linearization of manipulator dynamics using spatial operators," *IEEE transactions on Systems, Man, and Cybernetics*, vol. 23, no. 1, pp. 239–248, 1993.