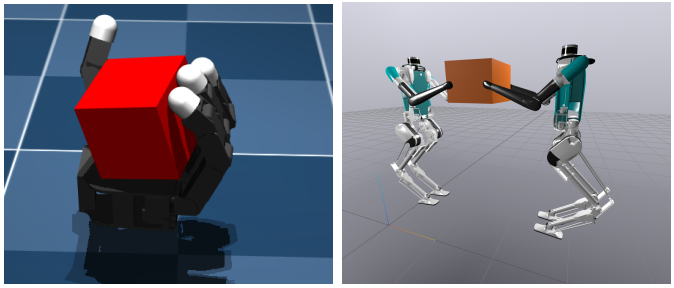
Constrained Articulated Body Algorithms for Closed-Loop Mechanisms

Ajay Suresha Sathya[1] and Justin Carpentier[1]

***Abstract*—Efficient rigid-body dynamics algorithms are instrumental in enabling high-frequency dynamics evaluation for resource-intensive applications (e.g., model predictive control, large-scale simulation, reinforcement learning), potentially on resource-constrained hardware. Existing recursive algorithms with low computational complexity are mostly restricted to kinematic trees with external contact constraints or are sensitive to singular cases (e.g., linearly dependent constraints and kinematic singularities), severely impacting their practical usage in existing simulators. This article introduces two original, low-complexity recursive algorithms: the loop-constrained articulated body algorithm (LCABA) and proxBBO, both based on a proximal dynamics formulation for forward simulation of mechanisms with loops. These algorithms are derived from first principles using non-serial dynamic programming, exhibit linear complexity in practical scenarios, and are numerically robust in the face of singular cases. They extend the existing constrained articulated body algorithm (constrainedABA) to handle internal loops and the pioneering BBO algorithm from the 1980s to singular cases. Both algorithms have been implemented by leveraging the open-source Pinocchio library, benchmarked in detail, and demonstrate state-of-the-art performance for various robot topologies, including over 6x speed-ups compared to existing non-recursive algorithms for high degree-of-freedom systems with internal loops, such as recent humanoid robots.**

I. Introduction

Efficient and reliable simulation of rigid-body dynamics is an important and extensively studied [1] problem in robotics. Efficient simulation is key for enabling computationally demanding downstream applications like model predictive control (MPC) [2] and reinforcement learning (RL) [3], and for generating large-scale synthetic datasets for modern data-hungry foundation models [4]. Faster simulation can unlock whole-body MPC for complex robots, allow for longer planning horizons that enhance optimality and safety, reduce the significant training and deployment time required for RL due to iterative reward and environment shaping, and even permit RL training on less expensive hardware. These simulation-driven techniques are at the forefront of research advancing reliable and real-time loco-manipulation with high degree-of-freedom (DoF) robots, potentially in contact-rich scenarios. Forward simulation algorithms in constrained scenarios, arising from bilateral [1], [5], unilateral or frictional contact constraints [6], [7], [8] use constrained dynamics algorithms (CDA) as a core inner solver, which solves the constrained forward dynamics problem. Constrained forward dynamics involves computing accelerations and constraint forces for a robot, given its configuration, velocity, applied forces, and desired constraint accelerations. To effectively simulate diverse loco-manipulation scenarios, the underlying CDA must efficiently handle a range of constraints, including those from contacts and kinematic loops, while being numerically robust to singular cases occurring at kinematic singularities or linearly dependent constraints.



(a) Several internal closed loops formed due to contact between the hand and the cube.

(b) Humanoid platforms consisting of internal loops in addition to the loop formed due to collaborative manipulation.

Fig. 1: Examples of closed-loops interactions or mechanisms classically encountered in robotics.

Internal loops, i.e., a kinematic loop of links that does not consist of the ground link, constitute a particularly challenging class of constraints to simulate efficiently for existing CDAs [5], [1]. This constraint class is increasingly gaining importance due to modern robot designs increasingly containing kinematic loops due to improved mechanical properties. Even robots without inherent loops can form internal loops dynamically during operation, e.g., a robot hand grasping a cube in Fig. 1a or two humanoids in Fig. 1b jointly transporting a heavy object. Submechanisms like gears and belt-transmissions also result in internal loops [1, Sec. 9.6], which need to be accounted for simulation accuracy.

Despite their high relevance, only a few simulators, such as Bullet [9], MuJoCo [10], or Simple [7], support internal loops, likely due to the challenge of simulating them efficiently. Even these select simulators use joint-space algorithms based on Featherstone's LTL algorithm [11], [12], [1]. However, the LTL algorithm does not fully exploit the available problem structure and is computationally expensive, with a complexity of $O(nd^2 + m^2d + md^2 + m^3)$, where n, d, and m are the robot DoFs, kinematic spanning tree depth, and the constraint dimensionality, respectively. Simple employs an extension of the LTL algorithm, that we refer

[1] Inria - Département d'Informatique de l'École normale supérieure, PSL Research University. {ajay.sathya,justin.carpentier}@inria.fr

to as the proximal LTL (proxLTL) algorithm [5]. ProxLTL handles loops and leverages a proximal dynamics formulation to address singular cases exactly and efficiently. This avoids the need for approximate techniques such as Tikhonov regularization or expensive techniques such as the singular value decomposition (SVD). However, proxLTL retains the same computational complexity as the original LTL algorithm.

In contrast, for kinematic trees with only external loops (such as ground contact), there exist efficient recursive CDAs with lower computational cost, such as the PV algorithm [13], [14] with $O(n+m^2d+m^3)$ complexity and recent $O(n+m)$ complexity algorithms such as PV-soft, PV-early [15], proxPV and constrainedABA [16]. Efficient C++ implementation of these recursive algorithms are also available within the widely used open-source PINOCCHIO [17] library. There also exist pioneering recursive algorithms [18], [19] that can handle internal loops. These two works independently proposed what is practically the same algorithm, which we refer to as BBO (Brandl, Bae, and others) after the first authors of the two papers. The BBO algorithm is challenging to derive and implement, and is vulnerable to singular cases. Likely due to these reasons, to the best of our knowledge, there is no efficient and user-friendly implementation of BBO available, it has not benchmarked against the LTL algorithms, and has seen limited use despite its potential advantages.

Addressing a key need for fast and reliable CDA that can handle internal loops, this paper explores *recursive* algorithms, that are robust to singular cases by leveraging the proximal dynamics formulation [5]. We introduce two efficient recursive CDAs: the loop-constrained articulated body algorithm (LCABA) and the proximal BBO (proxBBO) algorithm. LCABA and proxBBO extend the constrainedABA and proxPV algorithms [16] to handle internal loops, respectively. Additionally, proxBBO generalizes the original BBO algorithm [18], [19] to be robust to singular cases using the proximal approach, and is the recursive counterpart to the proxLTL algorithm [5]. These algorithms are derived by solving the quadratic program (QP) [20] associated with the Gauss' principle of least constraint (GPLC) [21], [22], [23] using Dynamic Programming (DP) [24] similarly to the approach in [15], [16]. However, internal loops introduce a graph structure that requires non-serial DP [25, Chap. 10], [26], [27], leading to a variable elimination (also known as bucket elimination in probabilistic inference [28]) approach.

Article contributions. Our contributions are listed as follows:

1) **LCABA:** LCABA, a recursive efficient algorithm that can handle singular cases and internal loops with a best-case complexity of $O(n+m)$, is derived by applying non-serial dynamic programming on GPLC problem.
2) **ProxBBO:** The BBO algorithm [18], [19] is generalized to handle singular cases using a proximal dynamics formulation to obtain the proxBBO algorithm. The proxBBO algorithm is derived similarly to LCABA, but with a different variable elimination order, and also has a best-case complexity of $O(n+m)$.
3) **Open-source and efficient implementations, detailed benchmarking and analysis:** Both the algorithms have been implemented in C++, leveraging the open-source library PINOCCHIO [17], and will be made publicly available after the review process. These algorithms are extensively benchmarked with the prevalent non-recursive LTL algorithm for different robot topologies.

Article organization. Section II reviews existing literature to situate our contributions, followed by Section III introducing the notation and necessary prior knowledge. The LCABA algorithm is derived, analyzed, and presented in an algorithmic form in Section IV, with an analogous treatment for the proxBBO algorithm in Section V. Section VI presents implementation details and benchmarking results, followed by a discussion in Section VII. Finally, Section VIII concludes the paper and outlines future work.

II. Related work

Constrained dynamics algorithms (CDAs) can be broadly classified based on whether they solve forward, inverse or hybrid dynamics [1]. In inverse dynamics, a robot's joint torques are computed given its configuration, velocity, acceleration, and constraint forces. In hybrid dynamics [1, Chapter 9], torques for a subset of the joints and accelerations for the remaining joints are provided as inputs, and the problem is to compute the accelerations and torques respectively of these joints. Due to our focus on simulation, this paper focuses exclusively on *forward dynamics* CDAs, where the robot's joint accelerations and constraint forces are computed given its configuration, velocity, joint torques and motion constraints. Forward dynamics CDAs can be further classified as recursive algorithms or joint-space algorithms (also called generalized/minimal/reduced coordinates). All the CDAs considered in this paper leverage a spanning tree of the kinematic graph representing the mechanism, where links and joints correspond to nodes and edges respectively. For completeness, we note an alternate non-spanning tree approach, inspired by [29], that involves constructing a large and sparse system consisting of each link's Newton-Euler equations and joint constraint equations, which is then solved using a general-purpose sparse linear solver. This approach is typically not computationally competitive against spanning-tree-based algorithms [1].

CDAs can be further classified based on whether motion constraints are formulated implicitly or explicitly [1, Eq.3.11]. Implicit formulation imposes constraints through additional equations that must be solved simultaneously with the equations of motion. Explicit constraint formulation computes a set of *independent* coordinates parametrizing the mechanism's constrained motion, projects the mechanism's equations of motion onto these independent coordinates and solves them.

In the remainder of this section, we will review existing algorithms for the unconstrained kinematic tree followed by CDAs using explicit or implicit constraint formulations.

Unconstrained kinematic trees. Dynamics algorithms for even unconstrained kinematic trees can be joint-space-based or recursive. The joint-space approach computes the joint-space inertia matrix (JSIM) (typically using the efficient composite rigid body algorithm (CRBA) [30]), and factorizes the JSIM efficiently using the LTL algorithm [11] that exploits

branching-induced sparsity in the JSIM to reduce factorization cost from $O(n^3)$ operations to $O(nd^2)$ operations. In contrast, the articulated body algorithm (ABA) [31], [32], [33], a recursive algorithm, has a linear computational complexity of $O(n)$, and scales better to high DoF robot systems.

Implicit constraint approach. The implicit approach simultaneously solves the dynamics equations and constraint equations. External loops, arising commonly out of robot-ground contacts, constitute a special class of implicit constraints that can be solved efficiently by cutting the loop at the ground link to preserve a tree structure. The joint-space LTL algorithm was extended in [12] to account for external loops. It exploits branching-induced sparsity to efficiently compute a computationally expensive intermediate quantity known as the Delassus matrix [34], [35] (also named inverse operational space inertia matrix (OSIM) [36]). This results in a $O(nd^2 + m^2d + md^2 + m^3)$ complexity algorithm. [5] introduced an extension of the LTL factorization in [11], called proxLTL, that leverages proximal algorithms [37] to cope with singular cases accurately. Among recursive algorithms for the implicit formulations of external loops constraints, the PV algorithm [13], [14] is a pioneering contribution with a $O(n + m^2d + m^3)$ computational complexity. The PV algorithm was recently revisited in [15], which provides a DP-based derivation for the PV algorithm by solving an equivalent discrete-time linear quadratic regulator [2] problem. [15] further proposed two original algorithms, PV-soft and PV-early, by relaxing motion constraints with quadratic penalties and through early elimination of constraint forces, respectively, both of which have a computational complexity of $O(n + m)$. However, PV-soft violates motion constraints, while PV-early is significantly challenging to implement, especially for singular cases (e.g., when there are redundant constraints). Finally, [16] applied the augmented Lagrangian method (ALM) [38], [39] on the proximal dynamics formulation [5] to derive a iterative algorithm constrainedABA, that retains the optimal $O(n + m)$ complexity of PV-early, while handling singular cases and being significantly simpler to implement.

For both joint-space methods and recursive algorithms, handling internal loops is more challenging and represents a significant increase in computational cost. [1, Chap.8] provides a detailed discussion on exploiting branching-induced sparsity for kinematics quantities associated with loop joints. Among joint-space methods, the proxLTL algorithm [5] also extends the LTL algorithm to support internal loops and also handles singular cases using proximal algorithms [40]. ProxLTL is also available as an efficient C++ implementation in the PINOCCHIO library [17]. Both LTL and proxLTL algorithms retain the $O(nd^2 + m^2d + md^2 + m^3)$ complexity for the internal loop case. Recursive algorithms for the internal loop case were pioneered by the BBO algorithm [18], [19] by independently rediscovering the PV algorithm and extending it to internal loops, resulting in a worst-case $O(n + m^2d + m^3)$ complexity algorithm. Both these works also pioneered the early elimination of internal loop constraints, resulting in a best-case computational complexity of $O(n + m)$ for local loops. The worst-case complexity manifests only when all the loops are coupled with each other (e.g., external loops).

Despite its low computational complexity, the BBO algorithm has not been adopted in existing simulators, perhaps due to its complexity, sensitivity to singular cases, absence of open-source implementations, and lack of benchmarking. Considering the demonstrated computational speed-ups provided by the recursive algorithms [15], [16] for external loops, there exists an unexplored opportunity to exploit recursive algorithms for internal loops, with the need for making them robust to singular cases. This is addressed by this article.

Explicit constraint approach. The explicit constraint formulation directly parametrizes the effective DoFs of the constrained system using independent coordinates, which are generally obtained by computing the nullspace of the implicit constraint formulation. This scales cubically, with the worst-case manifesting for mechanisms with a large loop consisting of $\propto n$ links. Both external and internal loops are identical in computational cost and implementation difficulty for the explicit constraint approach. In the joint-space approach, the JSIM and the joint torques are projected onto the independent coordinates and solved, which typically corresponds to $O(n^2n_m + nn_m^2 + n_m^3)$ operations, where n_m is the mobility of the constrained system. *Local* loops, e.g., due to a four-bar linkage, permit an efficient recursive approach through linear constraint embedding (LCE) [41], [42], [43]. LCE aggregates the links constituting a loop and defines aggregate-level generalization of rigid-body quantities such as spatial inertia, motion, and force vectors. This aggregation transforms a kinematic graph into a tree of the aggregated links, to which the articulated-body algorithm is straightforwardly adapted to, resulting in a recursive algorithm. When all loops are local, LCE-ABA has a best-case computational complexity of $O(n + m)$. It is particularly well-suited to handle local loops resulting from submechanisms such as gears, where the explicit constraint formulation is readily available and the loops are local. This algorithm has recently been implemented and open-sourced in [43]. The explicit approach has limited generality, as it becomes inefficient for large loops or coupled loops, which can occur frequently, e.g., for external loops when a robot makes multi-point contact with the ground. They are, moreover, sensitive to singular cases that can occur when loop-closure constraints become linearly dependent.

III. PRELIMINARIES

This section introduces the notation used in the paper, the connectivity graph representation of a mechanism, and the Gauss' principle of least constraint. We also review the two equivalent QP solver approaches, the augmented Lagrangian method (ALM) and the dual proximal point algorithm (PPA), that will be used to derive LCABA and proxBBO respectively. This is followed by a brief introduction to non-serial DP and joint-space CDAs.

A. Notation

Lower-case symbols (x), bold-faced lower-case symbols ($\mathbf{x}$), and upper-case symbols (X) represent scalars, vectors, and matrices, respectively. $\mathcal{C}(.)$ operator returns the cardinality of a given set. The operator $:=$ defines the left side symbol with the right side expression. The operators $\leftarrow$, $\overset{+}{\leftarrow}$ and $\overset{-}{\leftarrow}$ assigns, increments or decrements respectively the left side variable with the right side expression. The set of symmetric positive definite and symmetric positive semi-definite matrices of size $m \times m$ are denoted as $\mathbb{S}^m_{++}$ and $\mathbb{S}^m_{+}$ respectively. Any variable x overset with a bar and indexed with a set $\mathcal{Y}$ as $\bar{x}_{\mathcal{Y}}$ concatenates all the $x_i, \forall i \in \mathcal{Y}$. Any list $\mathcal{X}$ is reversed and denoted by $\mathcal{X}_r$.

Let $\mathbf{q} \in \mathcal{Q}$, $\boldsymbol{\nu} \in \mathcal{T}_{\mathbf{q}}\mathcal{Q} \simeq \mathbb{R}^n$ and $\dot{\boldsymbol{\nu}}$ be the robot generalized configuration, generalized velocity, and generalized accelerations respectively, where $\mathcal{Q}$ and $\mathcal{T}_{\mathbf{q}}\mathcal{Q}$ are the robot's configuration space and $\mathcal{Q}$'s tangent space at $\mathbf{q}$ respectively. Let $\boldsymbol{\tau} \in \mathcal{T}^*_{\mathbf{q}}\mathcal{Q} \simeq \mathbb{R}^n$ be the generalized forces exerted on the robot. If the generalized coordinates $\mathbf{q}$ span the Euclidean space $\mathbb{R}^n$, as it does for robot manipulator arms, $\mathbf{q}$, $\boldsymbol{\nu}$ and $\dot{\boldsymbol{\nu}}$ are simply the joint positions, velocities $\dot{\mathbf{q}}$ and accelerations $\ddot{\mathbf{q}}$ respectively. When $\mathcal{Q}$ is a non-trivial manifold, such as for floating-base robots, $\boldsymbol{\nu}$ and $\dot{\boldsymbol{\nu}}$ are not the time derivatives of $\mathbf{q}$, but are related to $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ through the tangent space of the local parameterization of $\mathcal{Q}$ [44]. We use Featherstone's spatial algebra [1] for rigid body quantities. The 6D spatial velocity and acceleration of a rigid body indexed by i is $\mathbf{v}_i \in \mathbb{M}^6$ and, $\mathbf{a}_i \in \mathbb{M}^6$ respectively, where $\mathbb{M}^6$ is the spatial motion vector space. The spatial forces acting on the i^{th} body is $\mathbf{f}_i \in \mathbb{F}^6$, where $\mathbb{F}^6$ is the spatial force vector space that is dual to $\mathbb{M}^6$. The spatial inertia of the i^{th} link is $H_i \in \mathbb{I}^{6\times6} \simeq \mathbb{S}^6_{++}$ and maps $\mathbb{M}^6$ to $\mathbb{F}^6$. $\times$ and $\times^*$ are the cross-product operators on the spatial motion and force vectors. Refer [1] for more details on the spatial algebra.

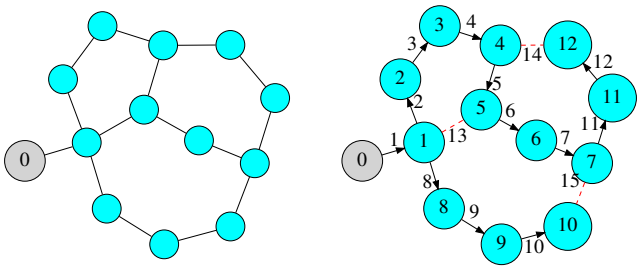
B. Kinematic graph

Let a mechanism with n_b links be modeled via a connectivity graph as shown in Fig. 2a for an illustrative mechanism. The mechanism's links and joints are denoted by nodes and edges, respectively. The ground is the 0^{th} link, also called the root link. The floating-base link in floating-base mechanisms, such as legged robots, is connected to the root through a 'free-flying joint', that permits relative motion freely. A tree is a special type of graph that is acyclic, i.e., there is a unique path between any two nodes. A subgraph of a graph consists of a subset of the original graph's links and joints. A spanning tree is a subgraph that is a tree and consists of all the nodes of the original graph. The original graph's edges absent in the spanning tree are termed cut edges or cut joints. For a given joint indexed i in the spanning tree connecting two links, the link closer to the root and the other link are termed the joint's parent link and child link, respectively, and are numbered $\pi(i)$ and i respectively. All the non-root links and joints are numbered topologically from 1 to n_b, such that $\pi(i) < i$, and the cut-joints are numbered from $n_b + 1$ to $n_b + m_b$. Let the lists $\mathcal{S} = \{1, 2, \ldots, n_b\}$ and $\mathcal{E} = \{n_b+1, n_b+2, \ldots, n_b+m_b\}$ index the non-root links/joints and cut-joints respectively. Let l^1_i and l^2_i index the two links connected by cut-joint i. Let the set $\gamma(i) = \{j \in \mathcal{S} \,|\, \pi(j) = i\}$ consist of the i^{th} link's children links. For the illustrative graph in Fig. 2a, Fig. 2b shows a spanning tree that is appropriately numbered, where spanning tree joints are shown as directed edges from parents to children links, and cut-joints are shown as dashed edges.

A spanning tree defines m_b number of *fundamental loops* in the graph, where each fundamental loop, indexed i, is the loop created when a cut-joint i is added to the spanning tree. From now on, we will refer to each fundamental loop simply as a

TABLE I: List of symbols and notations used in the article.

Symbol	Meaning
n	Number of robot degrees of freedom.
m	Number of motion constraints (excluding spanning-tree joints).
d	Depth of the kinematic tree.
$\mathcal{C}(\cdot)$	Returns cardinality of a set.
$\mathbf{q}$	Robot configuration.
$\mathcal{Q}$	Robot configuration space.
$\boldsymbol{\nu}$	Generalized robot velocities.
$\mathcal{T}_{\mathbf{q}}\mathcal{Q}$	Tangent space of $\mathcal{Q}$ at $\mathbf{q}$.
$\dot{\boldsymbol{\nu}}$	Generalized robot accelerations.
$\boldsymbol{\tau}$	Generalized robot forces.
$\mathcal{T}^*_{\mathbf{q}}\mathcal{Q}$	Dual tangent space of $\mathcal{Q}$ at $\mathbf{q}$.
$\mathbf{v}_i$	i^{th} link's 6D spatial velocity.
$\mathbb{M}^6$	Motion vector space in spatial algebra.
$\mathbf{a}_i$	i^{th} link's 6D spatial acceleration.
$\mathbf{f}_i$	6D spatial forces acting on the i^{th} link.
$\mathbb{F}^6$	Force vector space, dual of $\mathbb{M}^6$.
H_i	i^{th} link's 6D spatial inertia tensor.
$\times$	Cross-product operator on spatial motion vectors.
$\times^*$	Cross-product operator on spatial force vectors.
$\pi(i)$	i^{th} link's parent link.
n_b	Number of links in the mechanism.
m_b	Number of cut-joints in the mechanism.
$\mathcal{E}$	Set of all cut-joint indices.
l^j_i	Index of the j^{th} link in the i^{th} cut-joint.
$\gamma(i)$	Set of i^{th} link's children link indices.
$\mathcal{S}$	Topologically ordered list of tree link indices.
$\mathcal{S}_r$	List $\mathcal{S}$ reversed.
SL(i)	Set of indices of supporting links of the i^{th} loop.
LS(i)	Set of indices of loops supported by the i^{th} link.
$\mathfrak{r}_i$	i^{th} loop's root link index.
$\mathcal{R}_i$	Set indexing loops rooted at the i^{th} link.
S_i	Spans i^{th} joint's motion subspace.
K^j_i	Constraint matrix of the i^{th} link for the j^{th} loop.
$\mathbf{k}_i$	Desired constraint accelerations of the i^{th} constraint.
$\mathbf{a}_{b,i}$	i^{th} link's bias acceleration vector.
M	Joint-space inertia matrix (JSIM).
$\dot{\boldsymbol{\nu}}_{\text{free}}$	Unconstrained spanning-tree generalized acceleration.
$\boldsymbol{a}_c$	Desired constraint accelerations.
$\mathbf{J}_{\mathbf{f}_c}$	Geometric Jacobian of $\mathbf{f}_c$.
$\boldsymbol{\lambda}$	Lagrange multipliers and constraint force magnitudes.
μ	Proximal operator / ALM parameter.
Λ_μ	Damped Delassus inverse matrix.
M_μ	Constraint augmented Inertia matrix.
$H_{i,j}$	Inertia matrix coupling the i^{th} and j^{th} links in LCABA.
$\mathcal{N}_i$	Set of link indices neighboring the i^{th} link in LCABA and LS(i) in proxBBO.
D_i	Apparent constrained inertia felt at the i^{th} joint.
P_i	Backward force propagation matrix at the i^{th} joint.
$\mathcal{S}^{\mathcal{E}}$	List of link indices in the LCABA elimination order.
U_i	$H_{i,i}S_i$.
$\mathfrak{m}_c$	Maximum number of neighbors for any link in LCABA.
$K_{i,j}$	Constraint matrix felt at the i^{th} link due to the j^{th} loop.
$L_{i,j}$	Constraint coupling matrix for the i^{th} and j^{th} cut-joint constraints.
$\mathcal{U}_i$	LS(i) - $\mathcal{R}_i$.
$\mathfrak{m}_b$	Maximum number loops supported by any link.



(a) Kinematic graph for an illustrative mechanism.

(b) Spanning tree for the graph in Fig. 2a.

loop. Let the set $\mathrm{SL}(i)$ (supporting links) contain indices of links supporting loop i, which means that these links constitute the loop i. Similarly, let the set $\mathrm{LS}(i)$ (loops supported) contain indices of loops that link i supports. A loop i's root, defined as $\mathfrak{r}_i = \mathbf{min}(\mathrm{SL}(i))$, is the link with the smallest index in the loop. Let the set $\mathcal{R}_i$ denote the set of loops for which link i is the loop root. Two loops i and j are considered to be coupled if they contain at least one joint in common.

C. Gauss' principle of least constraint

We now recall the Gauss' principle of least constraint (GPLC) [21], [23], an optimization-based mechanics formulation, which states that a constrained rigid body's acceleration under forces is the minimizer of the following strongly convex quadratic program (QP) [13], [15]:

$$\underset{\dot{\boldsymbol{\nu}},\mathbf{a}}{\textbf{minimize}} \quad \sum_{i=1}^{n_b}\left\{\frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T\mathbf{a}_i - \boldsymbol{\tau}_i^T\dot{\boldsymbol{\nu}}_i\right\} \tag{1a}$$

$$\textbf{subject to} \quad \mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i\dot{\boldsymbol{\nu}}_i + \mathbf{a}_{b,i}, \quad i \in \mathcal{S}, \tag{1b}$$

$$K_i^1\mathbf{a}_{l_i^1} + K_i^2\mathbf{a}_{l_i^2} = \mathbf{k}_i, \quad i = \mathcal{E}, \tag{1c}$$

$$\mathbf{a}_0 = -\mathbf{a}_{\text{grav}}, \tag{1d}$$

where $\mathbf{f}_i$ is the resultant spatial force on link i due to external forces and the bias forces $(-\mathbf{v}_i \times^* H_i\mathbf{v}_i)$, $\boldsymbol{\nu}_i \in \mathbb{R}^{n_i}$, $\dot{\boldsymbol{\nu}}_i \in \mathbb{R}^{n_i}$ and $\boldsymbol{\tau}_i \in \mathbb{R}^{n_i}$ are the i^{th} joint's generalized velocities, generalized accelerations and the joint torques respectively, where n_i is the i^{th} joint's DoFs. The acceleration recurrence equation in Eq. (1b) explicitly formulates the spanning tree joint constraints. The column vectors of the matrix $S_i \in \mathbb{R}^{6\times n_i}$ span the i^{th} joint's motion subspace. The i^{th} link's bias acceleration is $\mathbf{a}_{b,i} := \dot{S}_i\boldsymbol{\nu}_i$, which, during the common case of the joint axis being fixed w.r.t the parent link, is

$$\mathbf{a}_{b,i} = \mathbf{v}_i \times S_i\boldsymbol{\nu}_i.$$

The cut-joint motion constraints are implicitly formulated in Eq. (1c), where $K_i^1 \in \mathbb{R}^{m_i\times 6}$ and $K_i^2 \in \mathbb{R}^{m_i\times 6}$ are the constraint matrices on the links indexed l_i^1 and l_i^2 respectively, whose relative motion is constrained by the i^{th} cut-joint, $\mathbf{k}_i \in \mathbb{R}^{m_i}$ is desired constraint accelerations, where $m_i = 6 - n_i$ is the constraint dimension. Each row vector of K_i^1 (or K_i^2) is an element in $\mathbb{F}^6$. A uniform acceleration field of minus acceleration-due-to-gravity is added by fixing the root node acceleration in Eq. (1d). This strategy [33] spares us from adding each link's weight to $\mathbf{f}_i$, thereby providing some computational speed up.

Joint-space GPLC formulation results in the following QP problem:

$$\underset{\dot{\boldsymbol{\nu}}}{\textbf{minimize}} \quad \frac{1}{2}\|\dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_{\text{free}}(\mathbf{q},\dot{\boldsymbol{\nu}},\boldsymbol{\tau})\|^2_{M(\mathbf{q})} \tag{2a}$$

$$\textbf{subject to} \quad J_{\mathbf{f}_c}(\mathbf{q})\dot{\boldsymbol{\nu}} = \boldsymbol{a}_c(\mathbf{q},\boldsymbol{\nu}), \tag{2b}$$

where $M(\mathbf{q}) \in \mathbb{S}^n_{++}$, $\dot{\boldsymbol{\nu}}_{\text{free}}$, $J_{\mathbf{f}_c} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{a}_c \in \mathbb{R}^m$ are the joint-space inertia matrix (JSIM), unconstrained spanning-tree joint accelerations in absence of cut-joint constraints, constraint Jacobian and desired constraint accelerations respectively. The terms $J_{\mathbf{f}_c}$ and $\boldsymbol{a}_c$ are typically computed from Eq. (1c) by substituting $\mathbf{a}_i$ with the corresponding kinematic Jacobian equations

$$\mathbf{a}_i = J_i\dot{\boldsymbol{\nu}} + \dot{J}_i\boldsymbol{\nu}. \tag{3}$$

D. QP solver approaches

For solving a strongly convex QP of the form

$$\underset{\mathbf{x}}{\textbf{minimize}} \quad \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{q}^T\mathbf{x} \tag{4a}$$

$$\textbf{subject to} \quad A\mathbf{x} = \mathbf{b}, \tag{4b}$$

where $Q \in \mathbb{S}^n_{++}$, we now review the two effective and equivalent approaches behind LCABA and proxBBO, namely the augmented Lagrangian method (ALM) [39], [38], [20] and the dual proximal point algorithm (PPA) [40], [37] respectively. These approaches do not assume constraint linear independence (A need not have full row-rank), and are particularly efficient by being able to leverage the Cholesky decomposition [45], which is a fast linear solver. Alternate strategies to address redundant constraints like Tikhonov regularization or the truncated singular value decomposition (SVD) [45] undesirably bias the optimal $\mathbf{x}$ towards the origin or incur a high computational cost, respectively.

The QP's Lagrangian function [46] is defined as

$$\mathcal{L}(\mathbf{x},\boldsymbol{\lambda}_x) := \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{q}^T\mathbf{x} + \boldsymbol{\lambda}_x^T\left(A\mathbf{x} - \mathbf{b}\right), \tag{5}$$

where $\boldsymbol{\lambda}_x$ are the QP's dual variables.

Augmented Lagrangian method augments the Lagrangian function with a quadratic penalty on the constraint violation to define the augmented Lagrangian function (ALF)

$$\mathcal{L}^A(\mathbf{x},\boldsymbol{\lambda}_x) := \mathcal{L}(\mathbf{x},\boldsymbol{\lambda}_x) + \frac{\mu}{2}\|A\mathbf{x} - \mathbf{b}\|^2, \tag{6}$$

and alternately minimizes and maximizes the ALF w.r.t the primal and dual variables in an iterative fashion:

$$\mathbf{x}^{k+1} = \left(Q + \frac{\mu}{2}A^TA\right)^{-1}\left(-\mathbf{q} - A^T\boldsymbol{\lambda}_x^k + \mu A^T\mathbf{b}\right), \tag{7a}$$

$$\boldsymbol{\lambda}_x^{k+1} = \boldsymbol{\lambda}_x^k + \mu\left(A\mathbf{x}^{k+1} - \mathbf{b}\right), \tag{7b}$$

till a specified termination criterion is met.

Dual proximal point algorithm. The QP's dual function is

$$g(\boldsymbol{\lambda}_x) := \min_{\mathbf{x}} \mathcal{L}(\mathbf{x},\boldsymbol{\lambda}_x), \tag{8}$$

where minimizing $\mathcal{L}$ w.r.t $\mathbf{x}$ is always possible since Q is positive definite, yielding

$$g(\boldsymbol{\lambda}_x) = -\frac{1}{2}\boldsymbol{\lambda}_x^T \left(AQ^{-1}A^T\right)\boldsymbol{\lambda}_x - \left(AQ^{-1}\mathbf{q}+\mathbf{b}\right)^T\boldsymbol{\lambda}_x. \quad (9)$$

Note that the dual Hessian $AQ^{-1}A^T \in \mathbb{S}_+^m$ makes the dual function concave, but not necessarily strongly concave since A need not have full row-rank. The QP can be solved by maximizing the dual function (note that optimal $\mathbf{x}^*$ can be recovered from optimal $\boldsymbol{\lambda}^*$ using Eq. (7a)).

The proximal operator for a convex function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as

$$\text{prox}_{\mu,f}(\mathbf{x}^k) := \quad \underset{\mathbf{x}}{\textbf{arg min}} \left\{ f(\mathbf{x}) + \frac{1}{2\mu}\|\mathbf{x}-\mathbf{x}^k\|^2 \right\}, \quad (10)$$

where $\mu \in (0,\infty)$. The proximal point algorithm (PPA) minimizes the function f by performing fixed-point iterations using the proximal operator

$$\mathbf{x}^{k+1} := \text{prox}_{\mu,f}(\mathbf{x}^k), \quad (11)$$

until a termination criterion is met. The dual function $g(\boldsymbol{\lambda}_x)$ needs to be maximized to solve the optimization problem. This is achieved by minimizing $-g(\boldsymbol{\lambda}_x)$ using the PPA

$$\boldsymbol{\lambda}_x^{k+1} := \text{prox}_{\mu,-g}(\boldsymbol{\lambda}_x^k).$$

The shifted regularization term added in Eq. (10) makes each inner problem solved by the proximal operator strongly convex, even if the dual function is not strongly concave. For convex problems, PPA enjoys theoretical convergence guarantees [40, Sec. 4.1] for any choice of $\mu > 0$ under very mild conditions, assuming that the minimizer exists. However, the specific choice of μ, which can be interpreted as a step-size parameter in PPA, can significantly affect the convergence rate of both the ALM and the PPA methods. Proximal algorithms [40] have been found to be particularly effective in robotics [47], [48]. They are easily implemented, readily warm-startable, can handle infeasible initial guesses and even infeasible problems (by returning closest feasible solution [49]), and most often require few iterations (each of which is efficient) to converge to desired levels of accuracy for robot dynamics problems [5], [16], [7].

E. Joint-space algorithms

The joint-space GLPC problem formulation in Eq. (2) can also be solved using dual PPA or the ALM, resulting in the proxLTL [5] and proxLTLs algorithms, respectively. These two algorithms are the joint-space counterparts of proxBBO and LCABA, respectively, and will be reviewed in this section.

ProxLTL algorithm: The first iteration of dual PPA is equivalent to solving the following KKT system [5]

$$\begin{bmatrix} -\frac{1}{\mu}I_{m\times m} & J_{\mathbf{f}_c} \\ J_{\mathbf{f}_c}^T & M \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda}^1 \\ \dot{\boldsymbol{\nu}}^1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_c - \frac{1}{\mu}\boldsymbol{\lambda}^0 \\ M\dot{\boldsymbol{\nu}}_{\text{free}} \end{bmatrix}, \quad (12)$$

where both M and $J_{\mathbf{f}_c}$ have the spanning-tree-induced sparsity pattern [11],[1, Sec.8.9]. This sparsity was exploited in [5] to efficiently factorize the KKT system above using the UDU^T Cholesky decomposition (reverse of the typical LDL^T ordering). This approach turns out to compute the dual function as an intermediate quantity and the upper left-block of the U matrix corresponds to the Cholesky factor of the damped Delassus matrix $\left(\Lambda_\mu := J_{\mathbf{f}_c}^T M^{-1} J_{\mathbf{f}_c}^T + \frac{1}{\mu}I_{m\times m}\right)$. The Cholesky factor is reused to efficiently compute the subsequent dual PPA iterations

$$\Lambda_\mu \boldsymbol{\lambda}^{k+1} = J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}}_{\text{free}} - \boldsymbol{a}_c + \frac{1}{\mu}\boldsymbol{\lambda}^k. \quad (13)$$

ProxLTLs algorithm: Applying ALM on the joint-space GLPC problem yields the following updated equations

$$\dot{\boldsymbol{\nu}}^{k+1} = M_\mu^{-1}\left(\dot{\boldsymbol{\nu}}_{\text{free}} + J_{\mathbf{f}_c}^T\left[\mu\boldsymbol{a}_c - \boldsymbol{\lambda}^k\right]\right), \quad (14a)$$

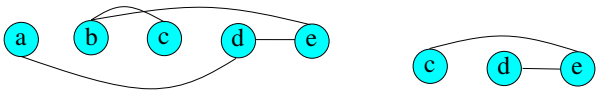
$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mu\left(J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}} - \boldsymbol{a}_c\right), \quad (14b)$$

where

$$M_\mu = M + J_{\mathbf{f}_c}^T\left(\mu I_{m\times m}\right)J_{\mathbf{f}_c} \in \mathbb{S}_{++}^n. \quad (15)$$

The proxLTLs was mentioned in [5], but not implemented, while it has been firstly considered in [16] for the restricted case of kinematic trees with external loops. In this restricted case, M_μ's sparsity pattern is identical to that of M. However, this property no longer holds for mechanisms with internal loops, where M_μ contains a dense block corresponding to all the joints comprising a loop. To see why, consider a row of the $J_{\mathbf{f}_c}$ matrix corresponding to a cut-joint. All the columns of this row corresponding to the joints in the loop will be non-zero in general, and a μ-weighted outer product of this row with itself yields a dense block in M_μ. This loss of sparsity significantly complicates the implementation of proxLTLs and also reduces its computational efficiency, therefore this algorithm is not considered in this paper.

F. Non-serial Dynamic Programming



(a) Visualizing the structure of the objective function minimized by non-serial DP.

(b) The graph structure after optimizing over nodes a and b.

Non-serial DP [25, Chap. 10], [26], [27] is a straightforward generalization of the traditional serial DP [24] to non-chain graphs. Suppose that the function to be optimized is given by

$$f(a,b,c,d,e) = p(a,d) + q(b,c) + r(b,e) + s(d,e),$$

whose structure can be visualized in Fig. 3a, with the variables depicted as nodes and the functions depicted as edges. A function depending on multiple variables is depicted by a clique involving the corresponding nodes and vice versa. DP successivly eliminates variables by optimizing over them to compute the optimal 'cost-to-go' *functions*, that depend on the remaining variables until all variables are eliminated.

Let $\mathcal{N}(g)$ denote the set of all the nodes that neighbor the node g in the graph, and let $\hat{f}(g,\mathcal{N}(g))$ denote the sum of all

the functions that depend on node g, and let $\tilde{f}(\mathcal{N}(g))$ denote the sum of all the functions depending on only the variables in $\mathcal{N}(g)$ and not on g. Optimizing over the variable g gives the function

$$\hat{f}^*(\mathcal{N}(g)) = \min_g \left\{ \hat{f}(g, \mathcal{N}(g)) \right\}, \tag{16}$$

which, in general, depends on all elements in $\mathcal{N}(g)$ and is thereby represented graphically as a clique comprising the nodes in $\mathcal{N}(g)$.The graph's connectivity structure is modified to add edges between any two links in $\mathcal{N}(g)$ if they were not already connected. Then the function $\tilde{f}(\mathcal{N}(g))$ is updated with the terms obtained from optimizing over g as follows

$$\tilde{f}(\mathcal{N}(g)) \overset{+}{\leftarrow} \hat{f}^*(\mathcal{N}(g)). \tag{17}$$

This step is repeated until all the variables are eliminated.

As an illustrative example, suppose that DP optimizes over variables in the order a, b, c, d, e for the case depicted in Fig. 3a. After eliminating a, we get

$$\tilde{f}(d) \leftarrow s(d, e) + \min_a p(a, d), \tag{18}$$

where the connectivity structure of the graph does not change since a has only one neighbor. At the next step, minimizing over b results in the following update:

$$\tilde{f}(c, e) = t(c, e) + u(d, e) + \min_b \left\{ r(b, c) + s(b, e) \right\}, \tag{19}$$

resulting in a new edge being added between the nodes c and e. This new graph connectivity is plotted in Fig. 3b. Similarly, the function is optimized over d and e to get a constant function, which is the optimum.

DP is in general intractable due to curse-of-dimensionality, which is an even greater problem for non-serial DP, where the cost of representing and optimizing the cost-to-go function at any step is exponential in the number of neighbors of the variable being optimized over. However, similarly to how the LQR problem is a tractable special case for the serial DP problem, we will find the mechanics problem of solving GPLC over graphs also to be a tractable problem since the cost at every stage can be parameterized efficiently as a quadratic form. Finally, it is worth noting that the traditional serial DP is a special case of the non-serial DP algorithm, where each leaf node of the graph has only one neighbor, due to which eliminating that variable does not modify the graph structure.

IV. Loop-constrainedABA

This section derives the LCABA algorithm, presents it in an algorithmic form, and analyzes its computational complexity.

A. LCABA derivation

The derivation applies the ALM on the GPLC problem from Eq. (1) and solves each inner primal problem (see Eq. (7a)) using DP. We will follow the notation and style of [15] for this DP-based derivation.

The ALF associated with GPLC (1) is given by:

$$\begin{aligned}\mathcal{L}^A(\dot{\boldsymbol{\nu}}, \mathbf{a}, \boldsymbol{\lambda}) = \sum_{i \in \mathcal{S}} \left\{ \frac{1}{2} \mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a}_i - \boldsymbol{\tau}_i^T \dot{\boldsymbol{\nu}}_i \right\} + \\ \sum_{i \in \mathcal{E}} \left\{ \boldsymbol{\lambda}_i^T \left(K_i^1 \mathbf{a}_{l_i^1} + K_i^2 \mathbf{a}_{l_i^2} - \mathbf{k}_i \right) + \right. \\ \left. \frac{\mu}{2} \| K_i^1 \mathbf{a}_{l_i^1} + K_i^2 \mathbf{a}_{l_i^2} - \mathbf{k}_i \|^2 \right\},\end{aligned} \tag{20}$$

where the spanning tree joint constraints (see Eq. (1b)) are excluded because these constraints will be eliminated by substitution similarly to single-shooting transcription [2] in optimal control.

The ALM's inner primal problem is defined by:

$$\dot{\boldsymbol{\nu}}^{k+1}, \mathbf{a}^{k+1} = \underset{\dot{\boldsymbol{\nu}}, \mathbf{a}}{\operatorname{argmin}} \, \mathcal{L}^A(\dot{\boldsymbol{\nu}}, \mathbf{a}, \boldsymbol{\lambda}^k). \tag{21}$$

Rearranging the constraint terms due to the i^{th} cut-joint in $\mathcal{L}^A$ yields the following quadratic form

$$\begin{aligned}\frac{1}{2} \begin{bmatrix} \mathbf{a}_{l_i^2} \\ \mathbf{a}_{l_i^1} \end{bmatrix}^T \begin{bmatrix} \mu K_i^{2T} K_i^2 & \mu K_i^{2T} K_i^1 \\ \mu K_i^{1T} K_i^2 & \mu K_i^{1T} K_i^1 \end{bmatrix} \begin{bmatrix} \mathbf{a}_{l_i^2} \\ \mathbf{a}_{l_i^1} \end{bmatrix} - \\ \begin{bmatrix} K_i^{2T} \left(\mu \mathbf{k}_i - \boldsymbol{\lambda}_i^k \right) \\ K_i^{1T} \left(\mu \mathbf{k}_i - \boldsymbol{\lambda}_i^k \right) \end{bmatrix}^T \begin{bmatrix} \mathbf{a}_{l_i^2} \\ \mathbf{a}_{l_i^1} \end{bmatrix}\end{aligned} \tag{22}$$

The quadratic form's off-diagonal blocks reveal an inertial coupling term $\mu K_i^{1T} K_i^2$ between links that are connected due to a cut-joint. To account for such inertial coupling during DP, we hypothesize the following quadratic form for the optimal cost-to-go function for the DP sub-problem at a spanning tree leaf link indexed i

$$\begin{aligned}V_i^* \left(\mathbf{a}_i, \bar{\mathbf{a}}_{\mathcal{N}_i} \right) := \frac{1}{2} \begin{bmatrix} \mathbf{a}_i \\ \bar{\mathbf{a}}_{\mathcal{N}_i} \end{bmatrix}^T \begin{bmatrix} H_{i,i} & \bar{H}_{i,\mathcal{N}_i}^T \\ \bar{H}_{i,\mathcal{N}_i} & \bar{H}_{\mathcal{N}_i,\mathcal{N}_i} \end{bmatrix} \begin{bmatrix} \mathbf{a}_i \\ \bar{\mathbf{a}}_{\mathcal{N}_i} \end{bmatrix} \\ - \begin{bmatrix} \mathbf{f}_i \\ \bar{\mathbf{f}}_{\mathcal{N}_i} \end{bmatrix}^T \begin{bmatrix} \mathbf{a}_i \\ \bar{\mathbf{a}}_{\mathcal{N}_i} \end{bmatrix},\end{aligned} \tag{23}$$

where $\mathcal{N}_i$ denotes the set of neighboring link indices for the i^{th} link. Note that initially, before any eliminations, $\mathcal{N}_i$ consists of the cut-joint neighbors of the i^{th} link, represented by the red dashed edges in Fig. 2b. The terms $\bar{\mathbf{a}}_{\mathcal{N}_i}$, $\bar{H}_{i,\mathcal{N}_i}$, $\bar{H}_{\mathcal{N}_i,\mathcal{N}_i}$ and $\bar{\mathbf{f}}_{\mathcal{N}_i}$ concatenate the $\mathcal{N}_i$ links' acceleration, coupled inertia and forces as follows. For all $j, k \in \mathcal{N}_i$

$$\bar{\mathbf{a}}_{\mathcal{N}_i} = \begin{bmatrix} \dots & \mathbf{a}_j^T & \dots \end{bmatrix}^T, \bar{\mathbf{f}}_{\mathcal{N}_i} = \begin{bmatrix} \dots & \mathbf{f}_j^T & \dots \end{bmatrix}^T \tag{24}$$

$$\bar{H}_{i,\mathcal{N}_i} = \begin{bmatrix} \vdots \\ H_{j,i} \\ \vdots \end{bmatrix}, \bar{H}_{\mathcal{N}_i,\mathcal{N}_i} = \begin{bmatrix} H_{k,k} & \dots & H_{k,j} & \dots \\ \vdots & \ddots & \vdots & \ddots \\ H_{k,j}^T & \dots & H_{j,j} & \dots \\ \vdots & \ddots & \vdots & \ddots \end{bmatrix}.$$

The quadratic form coefficients above are initialized by iterating over cut-joint constraints. For a cut-joint j, the quadratic form updates to the corresponding links are

$$\begin{bmatrix} H_{l_j^1, l_j^1} \\ H_{l_j^2, l_j^2} \\ H_{l_j^1, l_j^2} \end{bmatrix} \overset{+}{\leftarrow} \mu \begin{bmatrix} K_j^{1T} K_j^1 \\ K_j^{2T} K_j^2 \\ K_j^{1T} K_j^2 \end{bmatrix}, \begin{bmatrix} \mathbf{f}_{l_j^1} \\ \mathbf{f}_{l_j^2} \end{bmatrix} \overset{+}{\leftarrow} \begin{bmatrix} K_j^{1T} \left(\mu \mathbf{k}_j - \boldsymbol{\lambda}_j^k \right) \\ K_j^{2T} \left(\mu \mathbf{k}_j - \boldsymbol{\lambda}_j^k \right) \end{bmatrix}. \tag{25}$$

The DP's elimination process starts by selecting a leaf link indexed i in the spanning tree and eliminating its acceleration $\mathbf{a}_i$ and joint acceleration $\dot{\boldsymbol{\nu}}_i$. Apart from its cut-joint neighbor set $\mathcal{N}_i$, the i^{th} link is also connected to its parent link indexed $\pi(i)$ through the acceleration recurrence relation. Therefore, the functions and constraints depending on the i^{th} link may additionally depend on only the links in $\{\pi(i)\} \cup \mathcal{N}_i$. After eliminating the link i, the DP cost function involving these connected links is given by the DP recurrence relation as

$$V^*\left(\mathbf{a}_{\pi(i)}, \bar{\mathbf{a}}_{\mathcal{N}_i}\right) \leftarrow \frac{1}{2}\mathbf{a}_{\pi(i)}^T H_{\pi(i),\pi(i)}\mathbf{a}_{\pi(i)} - \mathbf{f}_{\pi(i)}^T\mathbf{a}_{\pi(i)} + \min_{\dot{\boldsymbol{\nu}}_i, \mathbf{a}_i}\left\{V_i^*\left(\mathbf{a}_i, \bar{\mathbf{a}}_{\mathcal{N}_i}\right) - \boldsymbol{\tau}_i^T\dot{\boldsymbol{\nu}}_i\right\}. \quad (26)$$

Substituting for $\mathbf{a}_i$ above using the acceleration recurrence relation from Eq. (1b) transforms the minimization above to

$$\min_{\dot{\boldsymbol{\nu}}_i}\left\{V_i^*\left(\mathbf{a}_{\pi(i)} + S_i\dot{\boldsymbol{\nu}}_i + \mathbf{a}_{b,i}, \bar{\mathbf{a}}_{\mathcal{N}_i}\right) - \boldsymbol{\tau}_i^T\dot{\boldsymbol{\nu}}_i\right\}. \quad (27)$$

Expanding the expression V_i^* above using the hypothesized cost-to-go parametrization from Eq. (23), and collecting the terms containing $\dot{\boldsymbol{\nu}}_i$ yields an unconstrained QP

$$\underset{\dot{\boldsymbol{\nu}}_i}{\text{minimize}} \quad \frac{1}{2}\dot{\boldsymbol{\nu}}_i^T D_i\dot{\boldsymbol{\nu}}_i - \Big[\boldsymbol{\tau}_i + S_i^T\Big(\mathbf{f}_i - \bar{H}_{i,\mathcal{N}_i}^T\bar{\mathbf{a}}_{\mathcal{N}_i} - H_{i,i}\left(\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i}\right)\Big)\Big]^T\dot{\boldsymbol{\nu}}_i, \quad (28)$$

where $D_i := \left(S_i^T H_{i,i} S_i\right)$ is the i^{th} link's inertia projected onto the i^{th} joint's subspace, and $D_i \in \mathbb{S}_{++}^{n_i}$ [19]. Solving the QP above yields the following optimal joint accelerations

$$\dot{\boldsymbol{\nu}}_i^* = D_i^{-1}\Big[\boldsymbol{\tau}_i + S_i^T\Big(\mathbf{f}_i - \bar{H}_{i,\mathcal{N}_i}^T\bar{\mathbf{a}}_{\mathcal{N}_i} - H_{i,i}\left(\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i}\right)\Big)\Big]. \quad (29)$$

Substituting the expression $\dot{\boldsymbol{\nu}}_i^*$ back in Eq. (26) results in following updates

$$H_{\pi(i),\pi(i)} \overset{+}{\leftarrow} P_i H_{i,i}; \quad (30a)$$

$$\mathbf{f}_{\pi(i)} \overset{+}{\leftarrow} P_i\left(\mathbf{f}_i - H_{i,i}\mathbf{a}_{b,i}\right) - H_{i,i}S_i D_i^{-1}\boldsymbol{\tau}_i; \quad (30b)$$

$$\bar{H}_{\pi(i),\mathcal{N}_i} \overset{+}{\leftarrow} \bar{H}_{i,\mathcal{N}_i}P_i^T; \quad (30c)$$

$$\bar{H}_{\mathcal{N}_i,\mathcal{N}_i} \overset{-}{\leftarrow} \bar{H}_{i,\mathcal{N}_i}S_i D_i^{-1}S_i^T\bar{H}_{i,\mathcal{N}_i}^T; \quad (30d)$$

$$\bar{\mathbf{f}}_{\mathcal{N}_i} \overset{+}{\leftarrow} \bar{H}_{i,\mathcal{N}_i}\Big[P_i\mathbf{a}_{b,i} + S_i D_i^{-1}\left(\boldsymbol{\tau}_i + S_i^T\mathbf{f}_i\right)\Big], \quad (30e)$$

where

$$P_i = I_{6\times 6} - H_{i,i}S_i D_i^{-1}S_i^T.$$

After eliminating the i^{th} link, it must be removed from the neighbor list $\mathcal{N}_j$, $\forall j \in \mathcal{N}_i$

$$\mathcal{N}_j \leftarrow \mathcal{N}_j - \{i\}, \text{ for } j \in \mathcal{N}_i, \quad (31)$$

and note from Eq. (30c) and Eq. (30d) that eliminating link i introduces mutual coupling between all the links in $\mathcal{N}_i \cup \{\pi(i)\}$, due to which the neighbor list of these links are updated as follows

$$\mathcal{N}_j \leftarrow \mathcal{N}_j \cup \left(\left(\mathcal{N}_i \cup \{\pi(i)\}\right) - \{j\}\right), \text{ for } j \in \mathcal{N}_i \cup \{\pi(i)\}, \quad (32)$$

and any undefined $H_{j,k}$ term for $\forall\, j,k \in \mathcal{N}_i \cup \{\pi(i)\}$ is set to $0_{6\times 6}$ before executing Eq. (30c) and Eq. (30d).

Subsequently, the next leaf link in the spanning tree is selected and eliminated, and this process is repeated until all the links in the tree are eliminated. Note that any link k that was not originally a leaf link becomes a leaf link itself, once all its child links $\gamma(k)$ are eliminated. The k^{th} link's neighbors $\mathcal{N}_k$ at this DP stage will have been recursively computed using Eq. (31) and Eq. (32), whenever any neighbor or child of link k is eliminated. Furthermore, the function $V_k^*(\mathbf{a}_k, \bar{\mathbf{a}}_{\mathcal{N}_k})$ is obtained as a quadratic form hypothesized in Eq. (23) using the recursive formulae in Eqs. 30. Therefore, it can be inductively shown that the DP hypothesis in Eq. (23) is valid throughout the elimination process.

Once all the links are eliminated, the optimal joint accelerations are calculated using Eq. (29) in reverse of the elimination order. This solves the ALM's inner primal problem. Subsequently, the ALM's dual variable updates (see Eq. (7b)) are straightforwardly performed using the constraint violations

$$\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \mu\left(K_i^1\mathbf{a}_{l_i^1}^{k+1} + K_i^2\mathbf{a}_{l_i^2}^{k+1} - \mathbf{k}_i\right). \quad (33)$$

When $\boldsymbol{\pi(i) \in \mathcal{N}_i}$, a special case occurs that must be addressed. The recursive formula in Eq. (30c) evaluates an off-diagonal block $\bar{H}_{\pi(i),\mathcal{N}_i}$ of the quadratic form assumed in Eq. (23). Because the quadratic form's Hessian is symmetric, each off-diagonal block has a symmetric counterpart. But if $\pi(i) \in \mathcal{N}_i$, the symmetric counterpart term also needs to be added to the single $H_{\pi(i),\pi(i)}$ block as follows

$$H_{\pi(i),\pi(i)} \leftarrow H_{\pi(i),\pi(i)} + \left(H_{i,\pi(i)}P_i^T\right)^T. \quad (34)$$

Subsequent primal iterations are efficient. The inner problem of the subsequent ALM iterations needs to only recompute the force and acceleration terms that change due to the updated Lagrange multipliers. The inertia terms H do not change and can be reused. Therefore, a reduced recursion that avoids the expensive matrix-matrix operations in Eq. (25) and Eq. (30) is devised. This reduced sweep only computes the less expensive operations needed for updating the force and acceleration terms using Eq. (25), Eq. (29), and Eq. (30). It is akin to re-using a factorized linear system for linear solves, making subsequent ALM iterations efficient. The reduced computations will be detailed later in this section.

Elimination ordering. A spanning tree can have multiple leaf links, any of which can be chosen for elimination at a DP step, yielding multiple valid elimination orders. This yields multiple elimination orders with potentially different computational costs. The cost of eliminating a link increases quadratically with the link's neighbor count (the i^{th} link's neighbor count is given by the cardinality of the set $\mathcal{N}_i$), and link elimination introduces coupling between neighbors and parent links. A poor elimination ordering choice can result in costly coupling among numerous links. Finding an optimal elimination order for the variable elimination, in general, has been shown to be an NP-hard problem [50]. However, various effective greedy heuristics such as minimum degree [51], nested dissection [52], and Cuthill-Mckee algorithm [53] have been proposed. LCABA adopts the minimum degree heuristic for its simplicity, whereby at each DP step, the leaf link with the lowest neighbor count is selected for elimination, with

ties being broken randomly. Let $\mathfrak{m}_c$ be the highest neighbor count encountered for an eliminated link during the LCABA elimination.

Constraints with respect to ground are a special case. The constraint model in Eq. (1) can also model external contact constraints that do not lead to internal loops, e.g., the four-foot-ground contact constraints for a quadruped. Suppose that the ground link is indexed l_i^1 without loss of generality for constraint i. Since the ground acceleration $\mathbf{a}_0 = -\mathbf{a}_{\text{grav}}$ is a constant and known in advance, its value is directly substituted, and $\mathbf{a}_0$ not being a decision variable, only the $H_{l_i^2,l_i^2}$ block and $\mathbf{f}_{l_i^2}$ needs to be updated in Eq. (25). To account for the $H_{l_i^1,l_i^2}$ coupling term, the force update needs to be modified as follows

$$\mathbf{f}_{l_i^2} \overset{+}{\leftarrow} K_i^{2T}\left(\mu\mathbf{k}_i - \mu K_i^1\mathbf{a}_0 - \boldsymbol{\lambda}_i^k\right). \tag{35}$$

Since the inertial coupling term is completely absorbed into the force update, the ground link and the link l_i^2 are not connected as neighbors, substantially simplifying the DP algorithm. The remaining LCABA steps continue exactly as derived previously. If all loops are external, LCABA reduces exactly to constrainedABA [16]. Therefore, LCABA generalizes constrainedABA, supporting both internal loops and external loops while retaining constrainedABA's efficiency for external loops.

Remark 1. *Though the constraint model in Eq.* (1) *constrains relative motion between **two** links, the presented DP-based algorithms can readily support constraints on multiple links of the form*

$$\sum_j \left\{K_i^j \mathbf{a}_{l_i^j}\right\} = \mathbf{k}_i, \tag{36}$$

with trivial modification. Only the quadratic form updates in Eq. (25) *need to be modified to compute the coupling among all the links involved in this constraint.*

However, LCABA's efficiency can drop drastically if too many links are involved in a constraint since the cost of each DP elimination step increases quadratically with the neighbour count. Therefore, LCABA is not recommended for constraints involving all the links (e.g. center-of-mass constraints).

B. Illustration of LCABA

LCABA's DP elimination is illustrated in Fig. 4 for an example mechanism shown in Fig. 4a, with eleven links and two cut-joints connecting the links 6&8 and links 8&11. Spanning-tree joint constraints are depicted by solid directed edges and the coupling terms in the DP's objective arising due to cut-joints are depicted by dashed red edges. Link 6 is first eliminated due to the minimum degree heuristic because it has the fewest neighbors among the leaf-links 6, 8 and 11 (though not strictly since the 8^{th} link has the same number of neighbors as the 6^{th} link). This elimination couples links 5&8, subsequently the new leaf links 5 and 4 are eliminated, coupling links 3&8 as shown in Fig. 4b. Then the leaf-links with least neighbors, 11, 10 and 9 are eliminated as seen in Fig. 4c. Note that since the links 3&8 were already coupled, eliminating link 9 does not introduce additional coupling and only modifies the existing coupling between links 3&8. Subsequently, the links 8 and 7 are eliminated as shown in Fig. 4d. Since the remaining links do not have neighbors, LCABA elimination proceeds identically to the ABA algorithm. Suppose that link 8 was eliminated first in Fig. 4a instead of links 6 or 11 which have fewer neighbors, it would have introduced coupling between links 6&11 as seen in Fig. 4e. This coupling would have increased the algorithm's computational cost since each leaf link now has two neighbors instead of one. Note that a link's neighbors are those with which the link has inertial coupling (denoted by the red dashed edges), which may not include the parent link in the spanning tree. This example demonstrates the minimum degree heuristic's benefit.

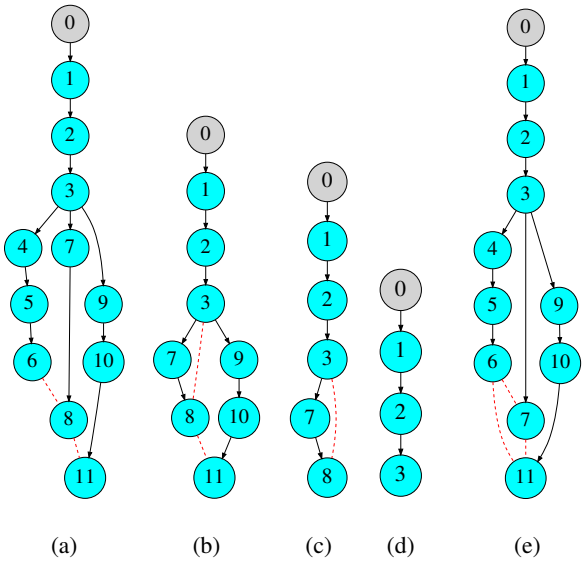


Fig. 4: Graphical illustration of LCABA's elimination steps.

C. LCABA algorithm

This subsection presents LCABA in an algorithmic form. Let $\mathcal{S}^{\mathcal{E}}$ be the order in which spanning-tree links are eliminated. The first ALM iteration is presented in Algorithm 1, followed by the more efficient reduced-sweep for subsequent ALM iterations in Algorithm 2, before presenting the whole LCABA in Algorithm 3. The terms in parentheses, such as $(U_i D_i^{-1})$, are stored in variables to avoid their re-computation. Since blocks in coupling matrices satisfy $H_{i,j} = H_{j,i}^T$, only $H_{i,j}$, for $i < j$ is computed and stored. However, this aspect is avoided in the algorithm for clarity. Furthermore, to highlight how LCABA extends constrainedABA, which itself extends ABA: LCABA, constrainedABA and ABA lines are marked in blue, brown, and black respectively. The reduced sweeps depicted in Algorithm 2 compute only the delta changes in forces and accelerations due to constraint force updates in an ALM iteration.

1) Forward kinematic sweep: The link velocities and the bias accelerations are computed as

$$\mathbf{v}_i = \mathbf{v}_{\pi(i)} + S_i\boldsymbol{\nu}_i, \quad \mathbf{a}_{b,i} = \mathbf{v}_i \times S_i\boldsymbol{\nu}_i$$

Link inertias and the link wrenches due to external forces and bias forces are initialized as

$$H_{i,i} \leftarrow H_i, \quad \mathbf{f}_i \leftarrow \mathbf{f}_i^{\text{ext}} - \mathbf{v}_i \times^* (H_i \mathbf{v}_i)$$

2) Resultant torques, apparent inertia, inverse inertia and projection matrix: are computed as

$$\mathbf{u}_i = \tau + S_i^T \mathbf{f}_i, \quad U_i = H_{i,i} S_i, \; D_i = S_i^T U_i, \; D_i^{-1}; \\ P_i = I_{6\times 6} - U_i \left(D_i^{-1} S_i^T\right).$$

3) Backward inertia and forces: The propagated inertias and forces to the parent link are updated as

$$H^a_{i,i} = H_{i,i} - \left(U_i D_i^{-1}\right) U_i^T; \quad H_{\pi(i),\pi(i)} \overset{+}{\leftarrow} H^a_{i,i} \\ \mathbf{f}_{\pi(i)} \overset{+}{\leftarrow} H^a_{i,i} \mathbf{a}_{b,i} + \mathbf{f}_i - \left(U_i D_i^{-1}\right) \mathbf{u}_i$$

4) Second forward sweep: The resulting joint and link accelerations at the $k+1$ iteration are computed as

$$\dot{\boldsymbol{\nu}}_i^{(k+1)} = D_i^{-1} \mathbf{u}_i - \left(U_i D_i^{-1}\right)^T (\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i}); \\ \mathbf{a}_i^{k+1} = \mathbf{a}_{\pi(i)}^{k+1} + S_i \dot{\boldsymbol{\nu}}_i^{(k+1)} + \mathbf{a}_{b,i}$$

Algorithm 1 LCABA three-sweep algorithm

Require: $\mathbf{q}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, S_is, H_is, K_is, $\mathbf{k}_i$s, $\mathbf{f}_i^{\text{ext}}$, $\mathcal{S}^{\mathcal{E}}$, μ, $\boldsymbol{\lambda}^0$, $\mathcal{E}$

1: **for** i in $\mathcal{S}$ ▷ First forward sweep
2: FK and initialize inertias and forces (Section IV-C1).
3: $\mathcal{N}_i \leftarrow \{\}$
4: **for** j in $\mathcal{E}$ ▷ Process constraints: update inertia, forces and neighbors
5: $H_{l_j^1,l_j^1} \overset{+}{\leftarrow} \mu K_j^{1T} K_j^1, \quad H_{l_j^2,l_j^2} \overset{+}{\leftarrow} \mu K_j^{2T} K_j^2;$
6: **if** $H_{l_j^1,l_j^2}$ is undefined ▷ Create new edges
7: $H_{l_j^1,l_j^2} \leftarrow 0_{6\times 6}$;
8: $\mathcal{N}_{l_j^1} \leftarrow \mathcal{N}_{l_j^1} \cup \{l_j^2\}$; $\mathcal{N}_{l_j^2} \leftarrow \mathcal{N}_{l_j^2} \cup \{l_j^1\}$
9: $H_{l_j^1,l_j^2} \overset{+}{\leftarrow} \mu K_j^{1T} K_j^2, \quad \mathbf{f}_{l_j^1} \overset{+}{\leftarrow} K_j^{1T} \left(\mu \mathbf{k}_j - \boldsymbol{\lambda}_j^k\right),$
10: $\mathbf{f}_{l_j^2} \overset{+}{\leftarrow} K_j^{2T} \left(\mu \mathbf{k}_j - \boldsymbol{\lambda}_j^k\right)$
11: **for** i in $\mathcal{S}^{\mathcal{E}}$ ▷ Backward sweep
12: Compute resultant torques, apparent inertia, inverse inertia and projection matrix (Section IV-C2).
13: **if** $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Update terms after i^{th} link's elimination
14: **for** j in $\mathcal{N}_i + \{\pi(i)\}$ ▷ Update connections
15: $\mathcal{N}_j \leftarrow \mathcal{N}_j - \{i\}$; ▷ Remove i^{th} link from neighbors
16: **for** k in $\mathcal{N}_i + \{\pi(i)\} - \{j\}$
17: $H_{k,j} \leftarrow 0_{6\times 6}$; if $H_{k,j}$ undefined
18: $\mathcal{N}_j \leftarrow \mathcal{N}_j \cup \{k\}$ ▷ Update $\mathcal{N}_j$
19: Compute Eqs. (30c) to (30e) ▷ Update coupling inertias and forces.
20: **if** $\pi(i) \in \mathcal{N}_i$ ▷ Re-symmetrize if parent is a neighbor
21: $H_{\pi(i),\pi(i)} \overset{+}{\leftarrow} \left(H_{i,\pi(i)} P_i^T\right)^T;$
22: **if** $\pi(i) > 0$
23: Backward inertia and forces propagation (Section IV-C3)
24: **for** i in $\mathcal{S}_r^{\mathcal{E}}$ ▷ Second forward sweep (Rollout)
25: $\mathbf{u}_i \overset{-}{\leftarrow} \left(\bar{H}_{i,\mathcal{N}_i} S_i\right)^T \bar{\mathbf{a}}_{\mathcal{N}_i};$
26: Compute joint and link accelerations (Section IV-C4)

Algorithm 2 LCABA reduced two-sweep algorithm

Require: $\Delta\mathbf{f}_i$s, K_is, $\left(\bar{H}_{\mathcal{N}_i} S_i\right)$s, U_is, D_is, S_is, $\mathcal{S}^{\mathcal{E}}$

for i in $\mathcal{S}^{\mathcal{E}}$ ▷ Reduced backward sweep
$\Delta\mathbf{u}_i \leftarrow S_i^T \Delta\mathbf{f}_i$ ▷ Resultant torque changes
if $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Update forces on neighbors due to $\Delta\mathbf{u}_i$
$\Delta\bar{\mathbf{f}}_{\mathcal{N}_i} \overset{+}{\leftarrow} \left(\bar{H}_{\mathcal{N}_i} S_i\right) \left(D_i^{-1} \Delta\mathbf{u}_i\right)$
if $\pi(i) > 0$ ▷ Update forces on parent due to $\Delta\mathbf{u}_i$
$\Delta\mathbf{f}_{\pi(i)} \overset{+}{\leftarrow} \Delta\mathbf{f}_i - U_i \left(D_i^{-1} \Delta\mathbf{u}_i\right)$
$\Delta\mathbf{a}_0 \leftarrow 0_{6\times 1}$
for i in $\mathcal{S}_r^{\mathcal{E}}$ ▷ Reduced forward sweep
if $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Update $\Delta\mathbf{u}_i$ due to neighbor accelerations
$\Delta\mathbf{u}_i \overset{-}{\leftarrow} \left(\bar{H}_{i,\mathcal{N}_i} S_i\right)^T \Delta\bar{\mathbf{a}}_{\mathcal{N}_i}$
$\Delta\dot{\boldsymbol{\nu}}_i^{(k+1)} = D_i^{-1} \Delta\mathbf{u}_i - \left(U_i D_i^{-1}\right)^T \Delta\mathbf{a}_{\pi(i)};$
$\Delta\mathbf{a}_i = \Delta\mathbf{a}_{\pi(i)} + S_i \Delta\dot{\boldsymbol{\nu}}_i^{(k+1)}$

Algorithm 3 LCABA

Require: $\mathbf{q}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, S_is, H_is, K_is, $\mathbf{k}_i$s, $\mathbf{f}_i^{\text{ext}}$, $\mathcal{S}^{\mathcal{E}}$, μ, $\boldsymbol{\lambda}^0$, $\mathcal{E}$, ϵ, max_iter

Execute the three-sweep algorithm in Algorithm 1.
for k in range(1, max_iter)
for i in in $\mathcal{S}$
$\Delta\mathbf{f}_i \leftarrow \mathbf{0}_6$;
for i in $\mathcal{E}$ ▷ ALM multiplier update
$\Delta\mathbf{k}_i \leftarrow K_i^1 \mathbf{a}_{l_i^1} + K_i^2 \mathbf{a}_{l_i^2} - \mathbf{k}_i$ ▷ Compute constraint residual;
$\Delta\mathbf{f}_{l_i^2} \overset{-}{\leftarrow} \mu K_i^{2T} \Delta\mathbf{k}_i; \quad \Delta\mathbf{f}_{l_i^1} \overset{-}{\leftarrow} \mu K_i^{1T} \Delta\mathbf{k}_i;$
$\boldsymbol{\lambda}_i^{k+1} \overset{+}{\leftarrow} \mu \Delta\mathbf{k}_i;$
if $\min(\|\dot{\boldsymbol{\nu}}^k - \dot{\boldsymbol{\nu}}^{k-1}\|_\infty, \|\Delta\bar{\mathbf{k}}\|_\infty) < \epsilon$
break ▷ Terminate if converged
Execute reduced-two sweep algorithm in Algorithm 2
$\dot{\boldsymbol{\nu}}^{k+1} \leftarrow \dot{\boldsymbol{\nu}}^k + \Delta\dot{\boldsymbol{\nu}};$

D. LCABA complexity analysis

The worst-case computational complexity of LCABA is now analyzed, starting with the three-sweep Algorithm 1. The first forward sweep from line 1 to line 10 requires $O(n+m)$ operations. In the second forward sweep, line 25 requires $O(\mathcal{C}(\mathcal{N}_i))$ operations per joint, while the next line require a fixed number of operations, bringing the second forward sweep's complexity to $O(n+\mathfrak{m}_c n)$. Note that $\mathfrak{m}_c$ is the maximum neighbor count among all links, given by $\max_{i\in\mathcal{S}} \mathcal{C}(\mathcal{N}_i)$. Similarly to ABA and constrainedABA, the backward sweep is the most computationally expensive part, with line 19 requiring $O(\mathcal{C}(\mathcal{N}_i)^2)$ operations at each joint to update the inertial coupling by computing Eq. (30d). This brings the total computational complexity of the three-sweep algorithm to $O(n+m+\mathfrak{m}_c^2 n)$. Note that loops are local and not coupled in many practical cases, e.g., the four-bar submechanisms of the Digit robot in Fig. 1b. Even when there is coupling among loops, typically only a few loops ($\mathfrak{m}_c \sim 3$) participate in such coupling, bringing the effective complexity of LCABA to the best-case complexity of $O(n+m)$.

V. PROXBBO

This section presents the proxBBO algorithm, that generalizes the state-of-the-art recursive algorithm BBO [18], [19] to the proximal dynamics formulation to handle singular constraints.

A. ProxBBO derivation

ProxBBO is derived using the dualPPA discussed in Section III-D similarly to LCABA's derivation. ProxBBO uses DP to compute the following dual PPA iteration

$$\boldsymbol{\lambda}^{k+1} := \underset{\boldsymbol{\lambda}}{\mathbf{argmin}} \left\{ -\left(\min_{\dot{\boldsymbol{\nu}}} \mathcal{L}(\dot{\boldsymbol{\nu}}, \mathbf{a}, \boldsymbol{\lambda}) \right) + \frac{1}{2\mu} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^k\|^2 \right\}, \tag{37}$$

where

$$\mathcal{L}(\dot{\boldsymbol{\nu}}, \mathbf{a}, \boldsymbol{\lambda}) = \sum_{i=\mathcal{S}} \left\{ \frac{1}{2} \mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a}_i - \boldsymbol{\tau}_i^T \dot{\boldsymbol{\nu}}_i \right\} + \sum_{i \in \mathcal{E}} \left\{ \boldsymbol{\lambda}_i^T \left(K_i^1 \mathbf{a}_{l_i^1} + K_i^2 \mathbf{a}_{l_i^2} - \mathbf{k}_i \right) \right\}. \tag{38}$$

Note that the proximal regularization term in Eq. (37) does not depend on $\dot{\boldsymbol{\nu}}$, so the proximal term can be pushed inside the inner minimization problem to get

$$\boldsymbol{\lambda}^{k+1} := \underset{\boldsymbol{\lambda}^k}{\mathbf{argmax}} \left\{ \min_{\dot{\boldsymbol{\nu}}} \left(\mathcal{L}(\dot{\boldsymbol{\nu}}, \mathbf{a}, \boldsymbol{\lambda}) - \frac{1}{2\mu} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^k\|^2 \right) \right\}. \tag{39}$$

This max-min problem will be solved using DP.

From the Lagrangian's structure, we anticipate linear quadratic terms depending on $\mathbf{a}_i$ and the Lagrange multipliers associated with cut-joints of the loops supported by the i^{th} link. Therefore, we hypothesize the optimal cost-to-go *Lagrangian* for the DP step at a spanning-tree leaf-link to have the following quadratic form similarly to [15]

$$V_i^{\mathcal{L}*}\left(\mathbf{a}_i, \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i}\right) := \frac{1}{2} \begin{bmatrix} \mathbf{a}_i \\ \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i} \end{bmatrix}^T \begin{bmatrix} H_{i,i} & \bar{K}_{i,\mathcal{N}_i}^T \\ \bar{K}_{i,\mathcal{N}_i} & -\bar{L}_{\mathcal{N}_i,\mathcal{N}_i} \end{bmatrix} \begin{bmatrix} \mathbf{a}_i \\ \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i} \end{bmatrix} - \begin{bmatrix} \mathbf{f}_i \\ \bar{\mathbf{k}}_{\mathcal{N}_i} \end{bmatrix}^T \begin{bmatrix} \mathbf{a}_i \\ \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i} \end{bmatrix}, \tag{40}$$

where $\bar{\boldsymbol{\lambda}}_{\mathcal{N}_i}$, $\bar{K}_{i,\mathcal{N}_i}$, $\bar{L}_{\mathcal{N}_i,\mathcal{N}_i}$ and $\bar{\mathbf{k}}_{\mathcal{N}_i}$ aggregates the dual variables, constraint matrices, dual Hessian terms, and desired constraint accelerations for all the loop constraints supported by the link i such that $\mathcal{N}_i = \mathrm{LS}(i)$.

For every $j, k \in \mathcal{N}_i$ these terms are defined as follows

$$\bar{\boldsymbol{\lambda}}_{\mathcal{N}_i} = \begin{bmatrix} \dots & \boldsymbol{\lambda}_j^T & \dots \end{bmatrix}^T, \bar{\mathbf{k}}_{\mathcal{N}_i} = \begin{bmatrix} \dots & \mathbf{k}_j^T & \dots \end{bmatrix}^T \tag{41}$$

$$\bar{K}_{i,\mathcal{N}_i} = \begin{bmatrix} \vdots \\ K_{j,i} \\ \vdots \end{bmatrix}, \bar{L}_{\mathcal{N}_i,\mathcal{N}_i} = \begin{bmatrix} L_{k,k} & \dots & L_{k,j} & \dots \\ \vdots & \ddots & \vdots & \ddots \\ L_{k,j}^T & \dots & L_{j,j} & \dots \\ \vdots & \ddots & \vdots & \ddots \end{bmatrix},$$

At the start of the DP elimination, the terms above are initialized to zero and updated by iterating over the cut-joints $j \in \mathcal{E}$ as follows

$$K_{j,l_j^1} \leftarrow K_j^1, \quad K_{j,l_j^2} \leftarrow K_j^2, \tag{42a}$$

$$L_{j,j} \leftarrow \frac{1}{\mu} I_{m_j,m_j}, \tag{42b}$$

$$\mathbf{k}_j \leftarrow \frac{1}{\mu} \boldsymbol{\lambda}_j^k. \tag{42c}$$

For each joint $i \in \mathcal{S}$,

$$H_{i,i} \leftarrow H_i.$$

Similarly to LCABA, the DP recurrence relation for proxBBO is given by

$$V^{\mathcal{L}*}\left(\mathbf{a}_{\pi(i)}, \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i}\right) = \frac{1}{2} \mathbf{a}_{\pi(i)}^T H_{\pi(i),\pi(i)} \mathbf{a}_{\pi(i)} - \mathbf{f}_{\pi(i)}^T \mathbf{a}_{\pi(i)} + \bar{\boldsymbol{\lambda}}_{\mathcal{N}_{\pi(i)}}^T \bar{K}_{\pi(i),\mathcal{N}_{\pi(i)}} \mathbf{a}_{\pi(i)} + \min_{\dot{\boldsymbol{\nu}}_i, \mathbf{a}_i} \left\{ V_i^{\mathcal{L}*}\left(\mathbf{a}_i, \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i}\right) - \boldsymbol{\tau}_i^T \dot{\boldsymbol{\nu}}_i \right\}. \tag{43}$$

To solve the minimization problem above, link i's acceleration $\mathbf{a}_i$ is again eliminated via substitution using the acceleration recurrence relation to get following simplified unconstrained QP similarly to Eq. (28)

$$\underset{\dot{\boldsymbol{\nu}}_i}{\mathbf{minimize}} \quad \frac{1}{2} \dot{\boldsymbol{\nu}}_i^T D_i \dot{\boldsymbol{\nu}}_i - \left[\boldsymbol{\tau}_i + S_i^T \Big(\mathbf{f}_i - \bar{K}_{i,\mathcal{N}_i}^T \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i} - H_{i,i} \left(\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i} \right) \Big) \right]^T \dot{\boldsymbol{\nu}}_i, \tag{44}$$

optimizing which gives

$$\dot{\boldsymbol{\nu}}_i^* = D_i^{-1} \Big[\boldsymbol{\tau}_i + S_i^T \Big(\mathbf{f}_i - \bar{K}_{i,\mathcal{N}_i}^T \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i} - H_{i,i} \left(\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i} \right) \Big) \Big]. \tag{45}$$

Substituting the optimal $\dot{\boldsymbol{\nu}}_i^*$ expression back in to Eq. (43) gives a quadratic form for the function $V^*\left(\mathbf{a}_{\pi(i)}, \bar{\boldsymbol{\lambda}}_{\mathcal{N}_i}\right)$ with the following updates to the quadratic form coefficients

$$H_{\pi(i),\pi(i)} \overset{+}{\leftarrow} P_i H_{i,i}, \tag{46a}$$

$$\mathbf{f}_{\pi(i)} \overset{+}{\leftarrow} P_i \left(\mathbf{f}_i - H_{i,i} \mathbf{a}_{b,i} \right) - H_{i,i} S_i D_i^{-1} \boldsymbol{\tau}_i, \tag{46b}$$

$$\bar{K}_{\pi(i),\mathcal{N}_i} \overset{+}{\leftarrow} \bar{K}_{i,\mathcal{N}_i} P_i^T, \tag{46c}$$

$$\bar{L}_{\mathcal{N}_i,\mathcal{N}_i} \overset{+}{\leftarrow} \bar{K}_{i,\mathcal{N}_i} S_i D_i^{-1} S_i^T \bar{K}_{i,\mathcal{N}_i}^T, \tag{46d}$$

$$\bar{\mathbf{k}}_{\mathcal{N}_i} \overset{-}{\leftarrow} \bar{K}_{i,\mathcal{N}_i} \Big[P_i^T \mathbf{a}_{b,i} + S_i D_i^{-1} \big(\boldsymbol{\tau}_i + S_i^T \mathbf{f}_i \big) \Big]. \tag{46e}$$

The inertia and force recursions in Eq. (46a) and Eq. (46b) are identical to ABA and LCABA equations in Eq. (30). However, the $H_{i,i}$ and $\mathbf{f}_i$ terms computed by proxBBO differs from the corresponding terms in LCABA because proxBBO's Lagrangian (see Eq. (38)) does not contain the quadratic penalty terms of the ALF used in LCABA. The i^{th} link's constraint matrix $\bar{K}_{i,\mathcal{N}_i}$ is backpropagated to the parent link $\pi(i)$ in Eq. (46c). The set $\mathrm{LS}(\pi(i))$ for each link can be recursively computed using the following update rule

$$\mathcal{N}_{\pi(i)} \leftarrow \mathcal{N}_{\pi(i)} \cup \mathcal{N}_i. \tag{47}$$

Early elimination. The spanning-tree leaf-links can all be eliminated sequentially using the recursive formulae above before solving for the optimal Lagrange multipliers. However, this would introduce expensive coupling between all the loops,

eventually leading to $O(n+m^2n+m^3)$ operations. To counter this, both [18] and [19] propose eliminating loop constraints as soon as all links supporting the corresponding loop are eliminated. We adopt this approach for proxBBO as well.

The last link to be eliminated among the links supporting loop j is the loop's root link $i = \mathfrak{r}_j$ from its definition in Section III. The set of loops supported by link i, $\mathcal{N}_i$, can be partitioned into two sets: i) the set of loops for which the link i is a root denoted as $\mathcal{R}_i$ and ii) the remaining neighbor loops $\mathcal{U}_i := \mathcal{N}_i - \mathcal{R}_i$. The optimal cost-to-go Lagrangian function from Eq. (40) is also expanded based on this partition

$$V_i^{\mathcal{L}*}\left(\mathbf{a}_i, \bar{\boldsymbol{\lambda}}_{\mathcal{U}}, \bar{\boldsymbol{\lambda}}_{\mathcal{R}}\right) := \frac{1}{2}\begin{bmatrix}\mathbf{a}_i\\ \bar{\boldsymbol{\lambda}}_{\mathcal{U}_i}\\ \bar{\boldsymbol{\lambda}}_{\mathcal{R}_i}\end{bmatrix}^T \begin{bmatrix} H_{i,i} & \bar{K}_{i,\mathcal{U}_i}^T & \bar{K}_{i,\mathcal{R}_i}^T \\ \bar{K}_{i,\mathcal{U}_i} & -\bar{L}_{\mathcal{U}_i,\mathcal{U}_i} & -\bar{L}_{\mathcal{U}_i,\mathcal{R}_i} \\ \bar{K}_{i,\mathcal{R}_i} & -\bar{L}_{\mathcal{U}_i,\mathcal{R}_i}^T & -\bar{L}_{\mathcal{R}_i,\mathcal{R}_i}\end{bmatrix}\begin{bmatrix}\mathbf{a}_i\\ \bar{\boldsymbol{\lambda}}_{\mathcal{U}_i}\\ \bar{\boldsymbol{\lambda}}_{\mathcal{R}_i}\end{bmatrix} - \begin{bmatrix}\mathbf{f}_i\\ \bar{\mathbf{k}}_{\mathcal{U}_i}\\ \bar{\mathbf{k}}_{\mathcal{R}_i}\end{bmatrix}^T \begin{bmatrix}\mathbf{a}_i\\ \bar{\boldsymbol{\lambda}}_{\mathcal{U}_i}\\ \bar{\boldsymbol{\lambda}}_{\mathcal{R}_i}\end{bmatrix}. \tag{48}$$

The Lagrange multipliers associated with the loops in $\mathcal{R}_i$ are now eliminated, leading to the following updates to the DP cost function and the i^{th} link's neighbor set

$$V_i^{\mathcal{L}*}\left(\mathbf{a}_i, \bar{\boldsymbol{\lambda}}_{\mathcal{U}}\right) \leftarrow \max_{\bar{\boldsymbol{\lambda}}_{\mathcal{R}}} V_i^{\mathcal{L}*}\left(\mathbf{a}_i, \bar{\boldsymbol{\lambda}}_{\mathcal{U}}, \bar{\boldsymbol{\lambda}}_{\mathcal{R}}\right), \tag{49}$$

$$\mathcal{N}_i \leftarrow \mathcal{U}_i. \tag{50}$$

The optimizer $\bar{\boldsymbol{\lambda}}_{\mathcal{R}}^*$ of Eq. (49) is given by the necessary first-order optimality conditions of the corresponding QP problem

$$\bar{\boldsymbol{\lambda}}_{\mathcal{R}_i}^* = \bar{L}_{\mathcal{R}_i,\mathcal{R}_i}^{-1}\left(\bar{K}_{i,\mathcal{R}_i}\mathbf{a}_i - \bar{L}_{\mathcal{U}_i,\mathcal{R}_i}^T\bar{\boldsymbol{\lambda}}_{\mathcal{U}_i} - \bar{\mathbf{k}}_{\mathcal{R}_i}\right), \tag{51}$$

where $\bar{L}_{\mathcal{R}_i,\mathcal{R}_i}$ is invertible because it is initialized as a positive definite diagonal matrix due to the proximal regularization (see Eq. (42)), which is followed by adding symmetric positive semi-definite matrices to its diagonal blocks in Eq. (46d). Substituting optimal Lagrange multipliers back into the original DP cost function in Eq. (48) gives the following recursive formulae for the coefficients of $V_i^{\mathcal{L}*}\left(\mathbf{a}_i, \bar{\boldsymbol{\lambda}}_{\mathcal{U}}\right)$

$$\begin{bmatrix} H_{i,i} & \bar{K}_{i,\mathcal{U}_i}^T \\ \bar{K}_{i,\mathcal{U}_i} & -\bar{L}_{\mathcal{U}_i,\mathcal{U}_i}\end{bmatrix} \xleftarrow{+} \begin{bmatrix} \bar{K}_{i,\mathcal{R}_i}^T \\ -\bar{L}_{\mathcal{U}_i,\mathcal{R}_i}\end{bmatrix} \bar{L}_{\mathcal{R}_i,\mathcal{R}_i}^{-1} \begin{bmatrix} \bar{K}_{i,\mathcal{R}_i}^T \\ -\bar{L}_{\mathcal{U}_i,\mathcal{R}_i}\end{bmatrix}^T, \tag{52a}$$

$$\begin{bmatrix}\mathbf{f}_i \\ \bar{\mathbf{k}}_{\mathcal{U}_i}\end{bmatrix} \xleftarrow{+} \begin{bmatrix} \bar{K}_{i,\mathcal{R}_i}^T \\ -\bar{L}_{\mathcal{U}_i,\mathcal{R}_i}\end{bmatrix} \bar{L}_{\mathcal{R}_i,\mathcal{R}_i}^{-1} \bar{\mathbf{k}}_{\mathcal{R}_i}. \tag{52b}$$

This way, all links and loop constraints are eliminated from the leaf links down to the root link. The second forward sweep then computes the numerical values of the joint accelerations and the optimal Lagrange multipliers using Eq. (45) and Eq. (51) respectively. Note that similarly to the LCABA algorithm, the subsequent proximal iterations are efficient because only relatively inexpensive acceleration and force computations in Eqs. 42c, 46b, 46e, 52b, 51 and 45 need to be evaluated.

B. ProxBBO illustrative example

ProxBBO is illustrated in Fig. 5 on the same mechanism used in LCABA illustration in Section IV-B. Compared to LCABA, proxBBO introduces additional nodes for the Lagrange multipliers of each cut-joint constraint, depicted as red nodes in Fig. 5a. Similarly to LCABA, elimination of each link or constraint introduces coupling between all the neighbors (including a link's parent) of the eliminated link or constraint. Eliminating links 6, 5, and 4 results in a graph where the constraint 12 is coupled with the link 3 as seen in Fig. 5b. Then eliminating links 11, 10 and 9 couple constraints 13 and link 3 as seen in Fig. 5c. Eliminating link 8 introduces coupling between constraints 12 and 13 as seen in Fig. 5d since link 8 supports both the loops. Upon eliminating the next leaf link, link 7, we arrive at link 3 in Fig. 5e, which is the root link of both loop 12 and loop 13. Both the constraints are then eliminated to get the graph in Fig. 5f, from where the elimination steps proceed identically to the ABA algorithm.

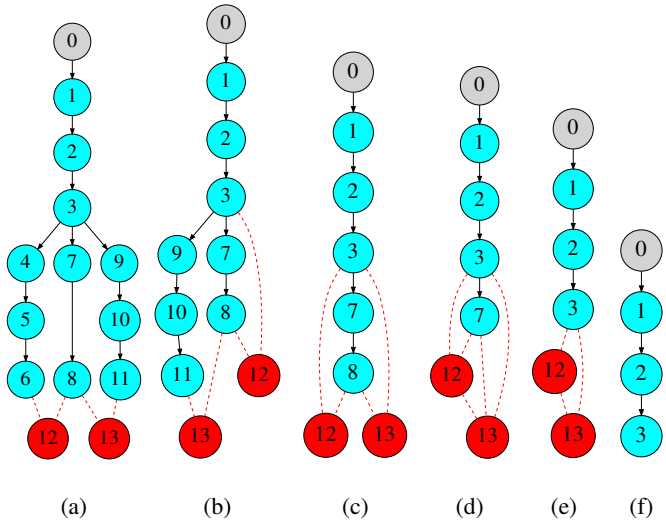


Fig. 5: Graphical illustration of the proxBBO's elimination steps. Compared to LCABA, additional red nodes are introduced to represent cut-joint Lagrange multipliers.

C. ProxBBO algorithm

This subsection presents proxBBO in an algorithmic form. Similarly to LCABA in Section IV-C, the three sweep algorithm corresponding to the first proximal iteration is presented in Algorithm 4, followed by the reduced sweep for the subsequent iterations in Algorithm 5 and the entire proxBBO algorithm in Algorithm 6. Again similarly to LCABA, the lines corresponding to ABA, proxPV and the proxBBO algorithms are colored in black, brown and blue respectively.

D. ProxBBO complexity analysis

Let the maximum number of loops supported by a link be

$$\mathfrak{m}_b = \max_{i \in \mathcal{S}} \mathcal{C}(\mathcal{N}_i), \tag{53}$$

in the proxBBO algorithm. The three-sweep algorithm from Algorithm 4 corresponding to the first proximal iteration dominates the computational cost of the algorithm. Reasoning similarly to the LCABA analysis, the first forward sweep

Algorithm 4 proxBBO three-sweep algorithm

Require: $\mathbf{q}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, S_is, H_is, K_is, $\mathbf{k}_i$s, $\mathbf{f}_i^{\text{ext}}$, μ, $\boldsymbol{\lambda}^0$, $\mathcal{E}$

1: **for** i in $\mathcal{S}$ ▷ First forward sweep
2: FK and initialize inertias and forces (Section IV-C1).
3: $\mathcal{N}_i \leftarrow \{\}$
4: **for** j in $\mathcal{E}$ ▷ Initialize constraint terms and connections
5: ${}^{l_j^1}K_j \leftarrow K_j^1$; ${}^{l_j^2}K_j \leftarrow K_j^2$;
6: $L_{j,j} \leftarrow \frac{1}{\mu} I_{m_j,m_j}$; $\mathbf{k}_j \xleftarrow{+} \frac{1}{\mu}\boldsymbol{\lambda}_j^0$;
7: $\mathcal{N}_{l_j^1} \leftarrow \mathcal{N}_{l_j^1} \cup \{j\}$; $\mathcal{N}_{l_j^2} \leftarrow \mathcal{N}_{l_j^2} \cup \{j\}$;
8: **for** i in $\mathcal{S}_r$ ▷ Backward sweep
9: **if** $\mathcal{C}(\mathcal{R}_i) > 0$ ▷ Early eliminate constraints rooted at i
10: $\mathcal{U}_i \leftarrow \mathcal{N}_i - \mathcal{R}_i$; $\mathcal{N}_i \leftarrow \mathcal{U}_i$; ▷ Update neighbor set
11: Update $H_{i,i}$, $\bar{K}_{i,\mathcal{U}_i}$, $\bar{L}_{\mathcal{U}_i,\mathcal{U}_i}$, $\mathbf{f}_i$, and $\bar{\mathbf{k}}_{\mathcal{U}_i}$ after constraint elimination using Eq. (52);
12: Compute resultant torques, apparent inertia (its inverse) and projection matrix (Section IV-C2).
13: **if** $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Propagate remaining constraints to parent
14: $\mathcal{R}_{\pi(i)} \xleftarrow{\cup} (\mathcal{N}_i \cap \mathcal{N}_{\pi(i)})$; $\mathcal{N}_{\pi(i)} \xleftarrow{\cup} \mathcal{N}_i$;
15: Update $\bar{K}_{\pi(i),\mathcal{N}_i}$, $\bar{L}_{\mathcal{N}_i,\mathcal{N}_i}$ and $\bar{\mathbf{k}}_{\mathcal{N}_i}$ using Eqs. (46c) to (46e);
16: **if** $\pi(i) > 0$
17: Backpropagate inertia and force to parent link (Section IV-C3).
18: **for** i in $\mathcal{S}$ ▷ Second forward sweep (roll-out)
19: **if** $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Add constraint forces to joint torques
20: $\mathbf{u}_i \leftarrow \left(\bar{K}_{i,\mathcal{N}_i} S_i\right)^T \bar{\boldsymbol{\lambda}}^1_{\mathcal{N}_i}$
21: Compute resulting joint and link accelerations (Section IV-C4).
22: **if** $\mathcal{C}(\mathcal{R}_i) > 0$ ▷ Compute eliminated constraint forces
23: $\bar{\boldsymbol{\lambda}}^1_{\mathcal{R}_i} = \bar{L}^{-1}_{\mathcal{R}_i,\mathcal{R}_i}\left(\bar{K}_{i,\mathcal{R}_i}\mathbf{a}_i^1 - \bar{L}^T_{\mathcal{U}_i,\mathcal{R}_i}\bar{\boldsymbol{\lambda}}^1_{\mathcal{U}_i} - \bar{\mathbf{k}}_{\mathcal{R}_i}\right)$

Algorithm 5 proxBBO reduced two-sweep algorithm

Require: $\Delta\mathbf{f}_i$s, $\Delta\mathbf{u}$, K_is, $\left(\bar{K}_{i,\mathcal{N}_i}S_i\right)$s, U_is, D_is, S_is

1: **for** i in $\mathcal{S}_r$ ▷ Reduced backward sweep
2: **if** $\mathcal{C}(\mathcal{R}_i) > 0$ ▷ Updates from early eliminated constraints
3: $\begin{bmatrix}\Delta\mathbf{f}_i \\ \Delta\bar{\mathbf{k}}_{\mathcal{U}_i}\end{bmatrix} \xleftarrow{+} \begin{bmatrix}\bar{K}^T_{i,\mathcal{R}_i} \\ -\bar{L}_{\mathcal{U}_i,\mathcal{R}_i}\end{bmatrix} \bar{L}^{-1}_{\mathcal{R}_i,\mathcal{R}_i}\Delta\bar{\mathbf{k}}_{\mathcal{R}_i}$;
4: $\Delta\mathbf{u}_i \leftarrow S_i^T\Delta\mathbf{f}_i$ ▷ Compute change in joint torque;
5: **if** $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Update desired constraint accelerations
6: $\Delta\bar{\mathbf{k}}_{\mathcal{N}_i} \xleftarrow{+} \left(\bar{K}_{i,\mathcal{N}_i}S_i\right)\left(D_i^{-1}\Delta\mathbf{u}_i\right)$;
7: **if** $\pi(i) > 0$ ▷ Propagate force changes to parent
8: $\Delta\mathbf{f}_{\pi(i)} \xleftarrow{+} \Delta\mathbf{f}_i - U_i\left(D_i^{-1}\Delta\mathbf{u}_i\right)$;
9: $\Delta\mathbf{a}_0 \leftarrow \mathbf{0}_6$;
10: **for** i in $\mathcal{S}$ ▷ Reduced forward sweep
11: **if** $\mathcal{C}(\mathcal{N}_i) > 0$ ▷ Add constraint forces to joint torques
12: $\Delta\mathbf{u}_i \leftarrow -\left(\bar{K}_{i,\mathcal{N}_i}S_i\right)^T\Delta\bar{\boldsymbol{\lambda}}_{\mathcal{N}_i}$;
13: $\Delta\dot{\boldsymbol{\nu}}_i^{(k+1)} = D_i^{-1}\Delta\mathbf{u}_i - \left(U_iD_i^{-1}\right)^T\Delta\mathbf{a}_{\pi(i)}$;
14: $\Delta\mathbf{a}_i = \Delta\mathbf{a}_{\pi(i)} + S_i\Delta\dot{\boldsymbol{\nu}}_i^{(k+1)}$;
15: **if** $\mathcal{C}(\mathcal{R}_i) > 0$ ▷ Compute eliminated constraint forces
16: $\Delta\bar{\boldsymbol{\lambda}}^{k+1}_{\mathcal{R}_i} = \bar{L}^{-1}_{\mathcal{R}_i,\mathcal{R}_i}\big(\bar{K}_{i,\mathcal{R}_i}\Delta\mathbf{a}_i^{k+1} - \bar{L}^T_{\mathcal{U}_i,\mathcal{R}_i}\Delta\bar{\boldsymbol{\lambda}}^{k+1}_{\mathcal{U}_i} - \Delta\bar{\mathbf{k}}_{\mathcal{R}_i}\big)$;

Algorithm 6 proxBBO

Require: $\mathbf{q}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, S_is, H_is, K_is, $\mathbf{k}_i$s, $\mathbf{f}_i^{\text{ext}}$, μ, $\boldsymbol{\lambda}^0$, $\mathcal{E}$, ϵ, max_iter

1: Execute the three-sweep algorithm in Algorithm 4.
2: **for** k in range(1, max_iter)
3: **for** i in in $\mathcal{S}$
4: $\Delta\mathbf{f}_i \leftarrow \mathbf{0}_6$
5: **for** i in $\mathcal{E}$ ▷ Update right-hand side of dual PPA problem
6: $\Delta\mathbf{k}_i^k \leftarrow \frac{1}{\mu}\Delta\boldsymbol{\lambda}_i^k$;
7: **if** $\min(\|\Delta\dot{\boldsymbol{\nu}}\|_\infty, \|\Delta\bar{\mathbf{k}}_{\mathcal{E}}\|_\infty) < \epsilon$ ▷ Check convergence
8: break
9: Execute reduced-two sweep algorithm in Algorithm 5;
10: $\dot{\boldsymbol{\nu}}^{k+1} \leftarrow \dot{\boldsymbol{\nu}}^k + \Delta\dot{\boldsymbol{\nu}}$;
11: $\bar{\boldsymbol{\lambda}}^{k+1}_{\mathcal{E}} \leftarrow \bar{\boldsymbol{\lambda}}^k_{\mathcal{E}} + \Delta\bar{\boldsymbol{\lambda}}_{\mathcal{E}}$;

can be shown to require $O(n+m)$ operations. The second forward sweep is more expensive than LCABA due to the line 23 requiring $O(\mathcal{C}(\mathcal{N}_i)^2)$ operations per joint, bringing the second forward sweep's cost to $O(n+\mathfrak{m}_b^2 n)$ operations. The backward sweep is the most expensive part, where back-propagating the constraint matrices, constraint coupling and the constraint accelerations in the line 15 incur $O(\mathcal{C}(\mathcal{N}_i))$, $O(\mathcal{C}(\mathcal{N}_i)^2)$ and $O(\mathcal{C}(\mathcal{N}_i))$ operations respectively at each joint bringing their total worst-case cost to $O(\mathfrak{m}_b^2 n)$ operations. ProxBBO also factorizes the constraint coupling matrices in line 11 during early elimination, incurring a cubic cost in the number of eliminated constraints, upper-bounded by $O(\mathfrak{m}_b^3)$ at each joint. If all the loops are coupled, this cost can even reach $O(m^3)$ operations. This brings the backward sweep cost to $O(n+\mathfrak{m}_b^3 n)$. The reduced two-sweeps in Algorithm 5 reuse the factorization from the three-sweep algorithm and incur a lower cost of $O(n+\mathfrak{m}_b^2 n)$ operations. Therefore, the total worst-case complexity of the proxBBO three-sweep algorithm, being dominated by the backward sweep computations, is $O(n+\mathfrak{m}_b^3 n)$. While the worst-case computational complexity of LCABA required pathological cases to manifest, the proxBBO algorithm's worst-case complexity is likelier to be encountered in the common case when the loops are external and coupled, e.g., ground contact for a quadruped.

VI. Experiments

This section discusses the C++ implementation of LCABA and proxBBO, presents the computational benchmarking of the algorithms on various robot setups and investigates the scaling of the algorithms for different topologies.

A. Implementation

The recursive algorithms LCABA and proxBBO are implemented in C++ using the efficient open-source dynamics library Pinocchio [17], and computationally benchmarked with the joint-space algorithm proxLTL, whose state-of-the-art version [5] is implemented in Pinocchio. All these three algorithms, being implemented in C++ and identically leveraging Pinocchio's efficient rigid-body dynamics functions,

contribute to benchmarking fairness. The proxLTL implementation in Pinocchio is particularly mature with implementation improvements since [5], and leverages vectorization, which empirically gives it quadratic scaling as opposed to the theoretically expected cubic scaling [16], making the comparison between the recursive and the joint-space algorithms particularly fair towards the joint-space algorithms. All timings were benchmarked on a laptop running Ubuntu 22.04 LTS with an `Intel® Core™ Ultra 7 165H CPU`, and the code was compiled using the `Clang 21.1.2` compiler.

To simulate the constrained system, our implementation supports the semi-implicit Euler method,

$$\boldsymbol{\nu}_{k+1} = \boldsymbol{\nu}_k + \Delta t\, \dot{\boldsymbol{\nu}}_k, \tag{54}$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k \oplus \Delta t\, \boldsymbol{\nu}_{k+1}, \tag{55}$$

where $\oplus$ denotes integration on manifold, supporting configuration spaces including Lie groups due to floating-base. Numerical integration can accumulate constraint drift over time. To stabilize the constraints against this drift, we support Baumgarte stabilization technique [54], where a proportional-derivative feedback term on the position and velocity terms of the constraint is added to the desired constraint acceleration term $\mathbf{k_i}$ in Eq. (1c).

We note that there is diverse and rich literature on integration schemes for constrained systems [55], [56]. Users can readily integrate our algorithms within any of the explicit integration schemes. However, implicit or half-explicit DAE schemes [56][Sec.VII] typically rely on Newton iterations, whose Jacobian may not match the problem structure assumed in Eq. (1), except for special cases such as [57]. Therefore, the presented algorithms require non-trivial extensions to be integrated within such schemes. Investigating this is left for future work.

B. Benchmarking on robot setups

We benchmark the presented algorithms on five robot scenarios consisting of internal closed-loops. We start with the 16 DoF Allegro Hand (AH) [1] holding a cube with its four fingertips. `Connect`-type 3D constraints, which allow relative rotation but not relative translation, are imposed at the contact point between the fingertips and the cube. The next example involves two Allegro hands collaboratively holding a cube. Subsequently, the two AHs holding the cube are attached to a humanoid[2] robot's wrists to investigate the scaling of the algorithms. Then, 6D `weld`-type constraints are imposed between the humanoid robot's feet and the ground, testing the algorithms on a combination of internal and external loops. Then we consider the Digit humanoid robot, which has three closed loops on each leg, standing with `Weld`-type constraints on the feet. This is followed by a Digit robot standing and holding a box with its wrists. Finally, we consider the example of two Digit platforms collaboratively holding a box with their wrists while standing.

Table II lists the benchmarking results. The robot name is listed on the left, with the superscript indicating the number of robots. The constraints are listed as T, F, or H, depending on whether the constraint is imposed on the fingertip, feet, or hand. The three closed-loops in each of Digit's legs are modeled by cutting a link involved in the loop and introducing a fixed joint, denoted by C. The subscript on the constraint indicates the type of constraint, and the superscript indicates the number of such imposed constraints. For each example, the timings are listed for one proximal/ALM iteration and three iterations to indicate how the cost might scale for a higher number of iterations. The computation timings are averaged over 100,0000 samples and are reported in μs. The benchmarking was done in Ubuntu's terminal mode to avoid interference with the background processes affecting benchmarking results. Intel's `Turbo Boost` is left turned on since we did not observe appreciable differences in computation timings between different runs. For reference, the computation timings of the vanilla ABA algorithm for unconstrained dynamics are also reported in the last column. The ALM parameter was set to an unrealistically low value of $\mu = 10^1$ to ensure that all the algorithms will execute three iterations without converging to obtain the timings for three iterations.

For a single Allegro Hand grasping a cube, both proxLTL and LCABA are competitive. Note that the Allegro Hand's topology, with its extensive branching and short depth, best suits the proxLTL algorithm. The resulting JSIM matrix enjoys a favorable sparsity structure that is even block-diagonal. ProxBBO algorithm is more expensive than the other algorithms for this contact-rich Allegro Hand example because all the loops are coupled through the free-floating cube leading to its worst-case (more on the worst case in Section VI-C) cubic complexity. LCABA emerges as the most efficient algorithm and remains so for the rest of the cases with larger robots due to its lower computational complexity. For the Digit robot, both the recursive algorithms scale better than the higher-complexity proxLTL algorithm, with LCABA being the fastest among the recursive algorithms. ProxBBO's improved performance on the Digit platform is due to limited coupling among constraints due to the closed loops in the legs being local. For the case of two Digits together holding a box, LCABA is even over 6X faster than the proxLTL algorithm.

C. Scaling results

This subsection studies how the algorithms scale for different robot topologies, starting with a single loop with a varying number of links, followed by a chain of loops, a worst-case mechanism topology where every loop is coupled with every other loop, and finally, a topology that particularly favors LCABA over proxBBO.

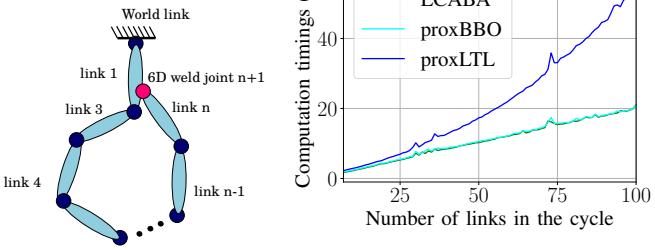
Single loop. A schematic diagram of a cycle of n links is shown in Fig. 6a, with the last link connected to the first link through a 6D *weld* joint (shown in red) and Fig. 6b shows the computation timings in μs for the first prox/ALM

[1]https://github.com/Gepetto/example-robot-data/tree/8d899847c8e7531a3d723b9647a79748056b0414/robots/allegro_hand_description/urdf accessed on Aug 19, 2024

[2]https://github.com/stack-of-tasks/pinocchio/blob/25714c7d738b08e98201871757811525db74f2aa/models/simple_humanoid.urdf

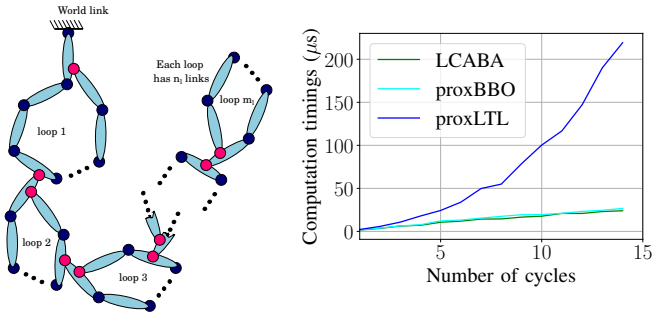
TABLE II: Computational timings of the proposed algorithms LCABA and proxBBO compared with proxLTL [5] up to two significant digits in μs is averaged over 100,000 samples. Timings of ABA [1] is provided for reference. The number of proximal/ALM iterations executed are indicated in the parentheses, system DoF within curly braces, and constraint dimension within the box brackets. Note that 6D constraints used for each loop in Digit's leg leads to a redundant constraint formulation.

System	LCABA	BBO	LTL	ABA
AH-cube-T^4_{3D}(1){22}[12]	**4.1**	5.1	4.4	2.0
AH-cube-T^4_{3D}(3)	5.6	7.3	5.9	-
AH^2-cube-T^8_{3D}(1){38}[24]	**7.6**	15	13	4.7
AH^2-cube-T^8_{3D}(3)	**9.7**	21	16	-
Hum-AH^2-cube-T^8_{3D}(1){73}[24]	**13**	26	25	9.0
Hum-AH^2-cube-T^8_{3D}(3)	**15**	33	28	-
Hum-AH^2-cube-$F^2_{6D}T^8_{3D}$(1) {73}[36]	**13**	31	32	9.0
Hum-AH^2-cube-$F^2_{6D}T^8_{3D}$ (3)	**16**	41	37	-
Digit-$F^2_{6D}C^6_{6D}$ (1){44}[48]	**9.1**	12	25	5.3
Digit-$F^2_{6D}C^6_{6D}$ (3){44}[48]	**12**	17	30	-
Digit-cube-$F^2_{6D}C^6_{6D}T^2_{3D}$ (1) {50}[54]	**11**	14	31	5.6
Digit-cube-$F^2_{6D}C^6_{6D}T^2_{3D}$ (3)	**14**	20	38	-
Digit^2-cube-F^4_{6D} C^{12}_{6D} T^4_{3D} (1) {94}[108]	**22**	37	160	11
Digit^2-cube-F^4_{6D} C^{12}_{6D} T^4_{3D} (3)	**29**	50	180	-



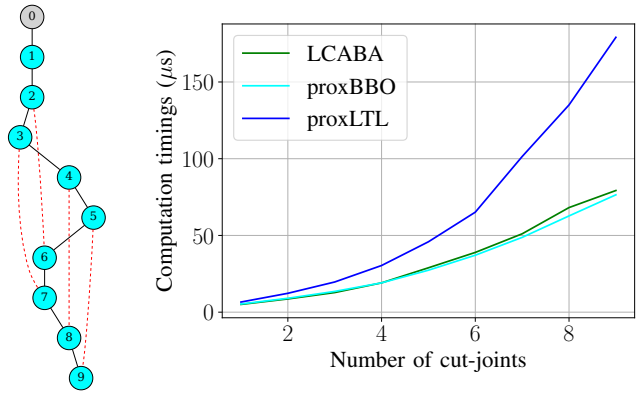
(a) A cycle with n links. (b) Computation timings in μs for the first prox/ALM iteration for the cycle mechanism.

Fig. 6: Computational scaling of the different CDAs for cyclic mechanisms.



(a) A chain of m_l loops with each loop consisting of n_l links. (b) Computation timings in μs for the first prox/ALM iteration for the cycle mechanism.

Fig. 7: Computational scaling of the different CDAs for cyclic mechanisms.



(a) (b)

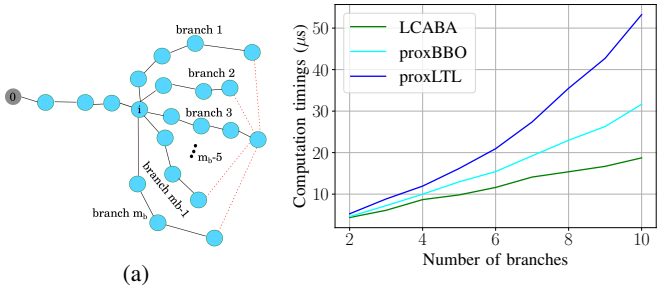
Fig. 8: Computational scaling of the different CDAs for cyclic mechanisms. a) A simple worst-case mechanism topology. b) Computation timings in μs for the first prox/ALM iteration for the cycle mechanism.

iteration. The proxLTL algorithm scales superlinearly with the number of links, as expected. Note that its cost is empirically observed to be quadratic and not the theoretically expected cubic cost due to the efficient implementation that leverages vectorization. Both the recursive algorithms display similar scaling since the cost of propagating a single constraint through a loop are similar for both the algorithms.

Chain of loops. Next, we benchmark the algorithms on a chain of loops as shown in Fig. 7a, with each loop consisting of seven links. The computation timings plotted in Fig. 7b follow a similar trend as the single loop case for the same reasons, with proxBBO being slightly faster than LCABA and proxLTL being the slowest.

Worst-case mechanism topology. The algorithms are next benchmarked on a mechanism topology shown in Fig. 8a, where all the loops are coupled. Each loop formed contains at least one joint from every other loop. The computation timings in Fig. 8b show that all the algorithms scale superlinearly, and the results indicate that the speed-up provided by the recursive algorithms is not as high in the worst-case scenario, where there are subject to a similar cubic computational complexity. is fortunately not encountered in practice.

Favorable mechanism topology. We now consider a topology where there is branching arising from a single link, and the branch tips are connected with each other with a 6D constraint as seen in Fig. 9a. LCABA's minimum degree heuristic ensures that the neighbor count for a leaf link being eliminated does not exceed one. The computation timings shown in Fig. 9b demonstrate that LCABA scaling is much better than prox-BBO, which, in turn, scales better than proxLTL for this mechanism topology.



(a) (b)

Fig. 9: Computational scaling for a mechanism topology that favors LCABA over proxBBO. a) the favorable mechanism topology with branching from a single link i, b) computation timings in μs for the first prox/ALM iteration as the number of branches m_b increases.

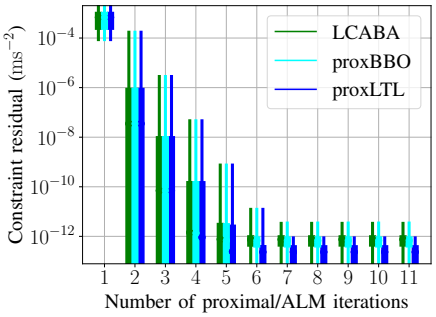


Fig. 10: Benchmarking convergence of the different algorithms on the two Allegro hands grasping a cube (second case in Table II with 24 constraints T^8_{3D}). This leads to a redundant constraint formulation that requires proximal methods. The constraint residual's inf norm is shown for 11 proximal/ALM iterations for $\mu = 10^5$ for 10,000 randomly generated examples. The box depicts mean and standard deviation, while line depicts the maximum and minimum values.

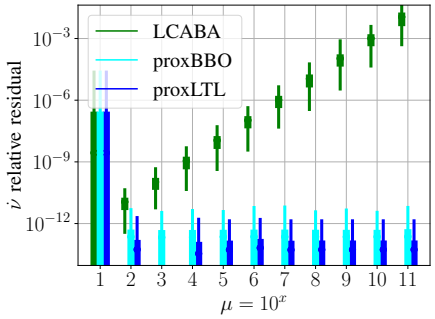


Fig. 11: Relative residual ($\frac{\|\dot{\nu} - \dot{\nu}_{\text{ref}}\|}{\|\dot{\nu}_{\text{ref}}\|}$) is plotted for the algorithms as μ increases, using the relatively numerically stable proxLTL algorithm with $\mu = 10^3$ as the reference over 10,000 randomly generated samples. The box depicts mean and standard deviation, while line depicts the maximum and minimum values.

D. Convergence of the algorithms

This section investigates the convergence of the constraint residuals (the ℓ_∞ norm of the residuals to be precise) and the numerical stability of the presented algorithms. Benchmarking is done for the second case in Table II with two Allegro hands grasping a cube with their fingertips, that leads to a singular case due to redundant constraint formulation. The results, plotted in Fig. 10 for $\mu = 10^5$, depict mean and standard deviation of the constraint residual with the boxes and maximum and minimum values of the constraint residual with the lines. These statistics, generated over 10,000 randomly sampled robot positions and control inputs, indicate rapid convergence for all three algorithms within a few iterations. Faster convergence was observed for higher values of μ. However, the augmented Lagrangian method behind LCABA, is known to be numerically sensitive to large quadratic penalty parameters μ values. Therefore, the numerical sensitivity of the different algorithms were investigated by comparing the solution ($\dot{\boldsymbol{\nu}}$) of different algorithms against the relatively numerically stable solution of proxLTL with $\mu = 10^3$ and the results are plotted in Fig. 11 over different values of μ. As expected, the ALM-based LCABA algorithm's solution deviated from the reference solution with increasing μ values, while both proxBBO and proxLTL demonstrated numerical stability up to $\mu = 10^{11}$.

Values of μ between 10^5 and 10^7 appear to provide a good balance between fast convergence and numerical stability for all the algorithms. We observed similar behavior across other setups and tasks, where the values of lead to convergence within a tolerance of 10^{-6} typically within 3 iterations. Note that an effective value of μ needs to be $> 10^3$ times the largest eigenvalue of the JSIM, since the relative magnitude of the penalty parameter and the objective function of the QP determines the convergence of the ALM. This eigenvalue can estimated using power iteration, which can be computed efficiently in a low-complexity manner using RNEA-based sweeps. However, we do not pursue this direction of automatic selection of the μ parameter because it is fairly easy to tune. If users of the presented algorithms find themselves with a mechanism whose JSIM spectrum is significantly different enough for the recommended range of μ to not work, they can easily tune μ by going high enough that the convergence is fast without encountering numerical issues.

VII. Discussions

This section critically discusses the presented algorithms, LCABA and proxBBO, their connections to existing literature, and potential directions for extensions. We first discuss how they generalize Riccati recursion to graphs. We then discuss the impact of the choice of spanning tree on the computational efficiency of the algorithms. We then highlight the connections between proxBBO and LCABA and how they can be combined to form a unified algorithm before connecting the algorithms to factor graphs and probabilistic inference. Finally, we discuss the choice of implicit versus explicit constraint formulations and its implications for the presented algorithms.

A. Generalizes Riccati recursion to graphs

The ABA [31], [32] and the PV algorithms [13] are known [58], [15] to generalize the celebrated Riccati recursion to tree-structured unconstrained and constrained equivalent LQR problems respectively. Such tree-structured Riccati recursions, parallelly developed in control and optimization literature [59], [60], are useful for solving stochastic optimal control problems with scenario trees [61]. The presented recursive algorithms eliminate links from leaves to root for a spanning tree using the joint acceleration recurrence relation similar to dynamics equation elimination via substitution in LQR solvers, effectively making them generalization of Riccati recursion beyond tree-structure to general graphs with loops. A straightforward occurrence of loops in an LQR problem is in periodic optimal control problems, where the periodicity constraint imposes the initial and terminal states to be equal. Efficient numerical algorithms for such periodic optimal control have been well studied, see [62], [63] and references therein. However, they do not appear to have been generalized to general graph structures.

Applying the presented algorithms to control problems for combining scenario trees and periodicity constraints, as well as studying the convergence and stabilization properties of such controllers, is a promising avenue for future work. Permitting graph structure in optimal control further enables interesting applications such as periodicity constraints at different frequencies for different subsets of states or for enforcing synchronization constraints in multi-agent systems at different time instants.

B. Spanning tree selection

It is important to choose the spanning tree that is assumed as an input to the presented algorithms appropriately, especially since it can significantly impact their computational efficiency. In practical scenarios, the spanning-tree choice is often straightforward. External contact constraints, e.g., finger-cube contact constraints from Fig. 1a, are modeled as cut joints, and the robot joints are included in the spanning tree to prioritize the mechanism's kinematic consistency. Even for robots with kinematic loops like the Digit robot, the actuated joints and the floating-base joint are typically chosen to get a spanning tree, and the closed-loop constraints and the sub-mechanism constraints are modeled as cut joints. This also often yields a favorable spanning tree for the presented algorithms.

It may be desirable to algorithmically automate the optimal spanning tree selection by solving a secondary discrete-optimization problem for a given mechanism and algorithm. However, it is well-known that finding an optimal elimination order, even without the restriction of conforming to a spanning-tree ordering, is an NP-complete problem [50]. Developing an effective algorithm for this problem is non-trivial and is unlikely to provide significant speed-ups for many existing robot topologies over manual spanning-tree selection. Therefore, this aspect is decidedly considered out of the scope of this paper so as not to overload it.

Relaxing spanning-tree elimination order: The spanning-tree based elimination ordering can be relaxed to fully leverage heuristics from numerical linear algebra such as minimum degree or minimum fill-in. However, such an approach does not exploit 'term-level' sparsity inherent in joint acceleration recurrence relations. It is likely to be less efficient than the presented algorithms for most practical robots. It also results in a significantly more complex algorithm, for example, eliminating a joint and the corresponding link in the middle of a chain of one DoF joints results in constructing a new fictitious two DoF joint between the link's parent and child link after tedious calculations and the new constraint moreover does not, in general, have term-level sparsity. Overall, this approach is sensitive to singularities necessitating expensive pivoting methods and corresponds to general-purpose sparse linear solvers, which are known to be less efficient than specialized algorithms [1].

C. Relation to factor graphs and probabilistic inference

The GPLC problem for a mechanism with loops can also be interpreted as a probabilistic inference problem over a Bayesian network [64]. Consider the link accelerations, joint accelerations, and joint torques as random variables, with a Bayesian prior on each link's acceleration given by a Gaussian distribution whose covariance is the link's inverse inertia matrix and mean is the link's acceleration in the absence of joint constraints. Each joint imposes a deterministic constraint between the accelerations of different links. Being an equivalent representation of the GPLC problem, solving for the maximum likelihood solution of this network, conditioned upon the applied joint torques, provides the solution to constrained dynamics problem. The factor graph perspective is not merely theoretically interesting, since non-serial DP and the variable elimination perspective used to derive this paper's algorithms are the same ideas underpinning inference algorithms [28] over factor graphs. This implies that the dynamics algorithms from this paper are suitable for probabilistic inference over factor graphs whose priors and constraints conform to the GPLC problem structure.

D. Connections between proxBBO and LCABA

LCABA can be derived from proxBBO's proximal formulation by eliminating Lagrange multipliers before eliminating any primal variable. Starting with the proximal formulation in Eq. (39), swap the order in which $\boldsymbol{\lambda}$ and $\dot{\boldsymbol{\nu}}$ are eliminated to get:

$$\dot{\boldsymbol{\nu}}^{(k+1)} = \operatorname*{\mathbf{argmin}}_{\dot{\boldsymbol{\nu}}} \max_{\boldsymbol{\lambda}} \left\{ \mathcal{L}(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) - \frac{1}{2\mu} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{(k)}\|^2 \right\}, \tag{56}$$

where solving the inner minimization problem yields LCABA's ALM formulation from Eq. (20) for the outer minimization problem. The equivalence between ALM and dual PPA is well-known in optimization [65] and has also been observed between constrainedABA and proxPV in the context of CDAs for kinematic trees and external loops [16].

Therefore, the main difference between LCABA and proxBBO is the elimination ordering of the variables from the proximal formulation. ProxBBO can be more efficient for

constraints involving a high number of links compared to the LCABA algorithm, making it suitable for constraints like center-of-mass (CoM) constraints. This suggests that a unified algorithm that can switch to proxBBO or LCABA based on the type of constraints and the connectivity graph structure can offer some speed-up. Like spanning-tree selection, this second-order enhancement requires discrete optimization and is left for future work. Similarly to constrainedABA [16], LCABA also generalizes the compliant constraint model in MuJoCo[8], with the first ALM iteration corresponding to solving MuJoCo's soft-Gauss principle (if dual variable initial guess $\boldsymbol{\lambda}^0 = \mathbf{0}$) and the subsequent iterations converging to rigid constraint model.

E. Explicit versus implicit constraint formulations

This paper focused exclusively on the implicit constraint formulation. Explicit constraint formulations are readily specified only for a limited set of constraints like four-bar linkages with single DoF joints and gear submechanisms. In other cases, it needs to be derived from the implicit formulation, leading to an additional cost that can get particularly expensive for large loops. Another common strategy to obtain an explicit constraint formulation is to assign a subset of joints as *independent*, which is, however, prone to kinematic singularities. Compared to these approaches, the implicit formulation is more readily specified, can handle a wider variety of constraints efficiently, and is less prone to singularities.

There exists Jain's linear constraint embedding (LCE) [41] approach that exploits explicit constraint formulation to propose efficient recursive algorithms. LCE, while particularly suited for local loops, can get expensive for larger loops (like the dual arm manipulation constraint or the feet-ground contact constraint for legged robots) because its cost increases cubically with the number of joints supporting the largest loop. While not as optimized for local loops as LCE, the presented algorithms can handle a wider variety of constraints efficiently. Investigating LCE's speedup compared to the presented algorithms for mechanisms with only local loops is interesting. Still, due to the complexity of implementing LCE within Pinocchio for fair comparison, this is considered outside the scope of this paper. Moreover, the LCE approach being compatible with the GPLC derivation [43], and can be embedded in the derivation of proxBBO or LCABA to obtain a hybrid algorithm that uses implicit formulation for larger loops and explicit formulation for local loops. However, it is non-trivial to combine these approaches, and the computational benefit of such a hybrid algorithm compared to LCABA/proxBBO is unclear. Answering these questions must be left for future work so as not to overload this paper.

VIII. Conclusion

This paper culminates the development of low-complexity CDAs, proxBBO and LCABA, in the context of proximal dynamics formulation [5], by efficiently extending proxPV and constrainedABA [16] respectively to a wide class of mechanisms with internal closed loops. The presented algorithms leverage proximal/ALM iterations, enabling them to account for singular cases due to redundant constraints and singular configurations in a straightforward manner. It also contributes to revisiting and reviving recursive low-complexity CDAs, that had fallen out of favor in modern simulators compared to the joint-space algorithms, by demonstrating compelling computational speed-ups compared to the state-of-the-art joint-space algorithm proxLTL.

LCABA matches proxLTL's performance for lower-dimensional robots while providing over 6x speed-ups for higher-dimensional robots like humanoids with several internal closed loops. LCABA also typically outperforms proxBBO, because it does not require additional factorizations to eliminate constraints and benefits from additional flexibility in choosing elimination orderings. It is also simpler to implement. However, proxBBO may be preferable for constraints that involve a large number of bodies, such as center-of-mass or momentum constraints, and is numerically more robust to high penalty parameters.

Limitations of the presented algorithms were also identified such as LCABA being numerically sensitive to high penalty parameters μ. The presented algorithms also likely require non-trivial modifications to support implicit integration schemes as well as some half-explicit schemes. Moreover, the proposed recursive algorithms assumed that a spanning tree is provided as input. They are also limited to a spanning tree elimination order and implicit constraint formulations. Finally, the proposed CDAs are currently limited to equality constraints, while many practical applications involve motion constraints arising from contacts, which are inequality constraints or frictional contact constraints.

Future Work: Future work will involve addressing these limitations. Most importantly, we will next explore strategies to support inequality constraints and frictional contact constraints leveraging the proposed algorithms. A natural avenue for such an extension is to use the ADMM-style approach to handle frictional contact proposed in [7]. The inner subproblem solved at each ADMM iteration in this approach directly matches the equality-constrained proximal dynamics problem solved in this paper. Another promising avenue for this extension would be to explore interior-point based approaches [66], which also require solving a sequence of equality-constrained problems. However, the contributions of the barrier term to the KKT conditions does not directly match the proximal dynamics problem, and therefore, will require additional, but minor, modifications to the presented algorithms.

The assumption of a spanning tree can be relaxed by developing a discrete-optimization framework to compute the optimal spanning tree or even to relax the spanning-tree elimination ordering constraint when beneficial. The proposed algorithms can also be extended to support explicit constraint formulations, by exploring combination of the proposed algorithm with the linear constraint embedding [41] approaches. Whether implicit integration schemes, with their energy dissipative properties, are beneficial in the context of robot control needs to be explored. If found to be relevant, we can also

explore extending our algorithms to support such schemes. However, this extension is expected to be non-trivial. Finally, differentiability of the proximal operator also makes it suitable for gradient-based optimization methods.

Through this future research agenda, this paper's algorithms have the potential to serve as the algorithmic foundation for speeding up contact-rich simulation and computation-intensive control applications such as MPC. Beyond mechanics, the presented algorithms can also find applications in control and estimation by effectively generalizing Riccati recursion to general graphs and through the factor graph connection.

ACKNOWLEDGEMENTS

This work was supported by the French government under the management of Agence Nationale de la Recherche through the project INEXACT (ANR-22-CE33-0007-01) and as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute), by the European Union's Horizon Europe research and innovation programme through a Marie Skłodowska-Curie Postdoctoral Fellowship (101211945 — ExTRAORDiNary) and through the AGIMUS project (GA no.101070165) and the Louis Vuitton ENS Chair on Artificial Intelligence. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

REFERENCES

[1] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.
[2] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, 2017, vol. 2.
[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.
[4] R. Firoozi, J. Tucker, S. Tian, A. Majumdar, J. Sun, W. Liu, Y. Zhu, S. Song, A. Kapoor, K. Hausman *et al.*, "Foundation models in robotics: Applications, challenges, and the future," *The International Journal of Robotics Research*, vol. 44, no. 5, pp. 701–739, 2025.
[5] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and sparse resolution of constrained dynamic equations," in *Proc. Robot., Sci. Syst.*, 2021.
[6] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 895–902, 2018. [Online]. Available: www.raisim.com
[7] J. Carpentier, Q. L. Lidec, and L. Montaut, "From Compliant to Rigid Contact Simulation: a Unified and Efficient Approach," in *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, July 2024.
[8] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco," in *Proc. IEEE Int. Conf. Robot. Autom.* IEEE, 2014, pp. 6054–6061.
[9] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.
[10] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Int. Robots. Syst.* IEEE, 2012, pp. 5026–5033.
[11] R. Featherstone, "Efficient factorization of the joint-space inertia matrix for branched kinematic trees," *Int. J. Robot. Res.*, vol. 24, no. 6, pp. 487–500, 2005.
[12] ——, "Exploiting sparsity in operational-space dynamics," *Int. J. Robot. Res.*, vol. 29, no. 10, pp. 1353–1368, 2010.
[13] A. F. Vereshchagin, "Modeling and control of motion of manipulational robots," *Soviet Journal of Computer and Systems Sciences*, vol. 27, no. 5, pp. 29–38, 1989.
[14] J. P. Popov, A. F. Vereshchagin, and S. L. Zenkevič, *Manipuljacionnyje roboty: Dinamika i algoritmy*. Nauka, 1978.
[15] A. S. Sathya, H. Bruyninckx, W. Decré, and G. Pipeleers, "An Efficient Constrained Dynamics Algorithm based on an Equivalent LQR Formulation using Gauss' Principle of Least Constraint," Oct. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.7243050
[16] A. S. Sathya and J. Carpentier, "Constrained articulated body dynamics algorithms," *IEEE Transactions on Robotics*, 2024.
[17] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, 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)*. IEEE, 2019, pp. 614–619.
[18] M. Otter, H. Brandl, and R. Johanni, "An algorithm for the simulation of multibody systems with kinematic loops," in *Proceedings of the 7th World Congress on Theory of Machines and Mechanisms, IFToMM, Sevilla, Spain*, 1987.
[19] D.-S. Bae and E. J. Haug, "A recursive formulation for constrained mechanical system dynamics: Part ii. closed loop systems," *Journal of Structural Mechanics*, vol. 15, no. 4, pp. 481–506, 1987.
[20] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.
[21] C. F. Gauß, "Über ein neues allgemeines grundgesetz der mechanik." 1829.
[22] H. Bruyninckx and O. Khatib, "Gauss' principle and the dynamics of redundant and constrained manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 3. IEEE, 2000, pp. 2563–2568.
[23] F. E. Udwadia and R. E. Kalaba, *Analytical dynamics : a new approach*. Cambridge: Cambridge University press, 1996.
[24] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.
[25] R. Aris, *Discrete Dynamic Programming: an introduction to the optimization of staged processes*. Blaisdell Publishing Company, 1964.
[26] U. Bertele and F. Brioschi, "Contribution to nonserial dynamic programming," *Journal of Mathematical Analysis and Applications*, vol. 28, no. 2, pp. 313–325, 1969.
[27] ——, *Nonserial dynamic programming*. Academic Press, Inc., 1972.
[28] R. Dechter, "Bucket elimination: A unifying framework for reasoning," *Artificial Intelligence*, vol. 113, no. 1-2, pp. 41–85, 1999.
[29] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 137–146.
[30] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, no. 3, pp. 205–211, 09 1982. [Online]. Available: https://doi.org/10.1115/1.3139699
[31] A. Vereshchagin, "Computer simulation of the dynamics of complicated mechanisms of robot-manipulators," *Eng. Cybernet.*, vol. 12, pp. 65–70, 1974.
[32] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *Int. J. Robot. Res.*, vol. 2, no. 1, pp. 13–30, 1983.
[33] H. Brandl, R. Johanni, and M. Otter, "A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix," *IFAC Proceedings Volumes*, vol. 19, no. 14, pp. 95–100, 1986.
[34] B. Brogliato, *Nonsmooth mechanics*. Springer, 1999, vol. 3.
[35] C. Duriez, F. Dubois, A. Kheddar, and C. Andriot, "Realistic haptic rendering of interacting deformable objects in virtual environments," vol. 12, no. 1, pp. 36–47, 2005.
[36] 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.
[37] R. T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM journal on control and optimization*, vol. 14, no. 5, pp. 877–898, 1976.
[38] M. J. Powell, "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298, 1969.
[39] M. R. Hestenes, "Multiplier and gradient methods," *Journal of optimization theory and applications*, vol. 4, no. 5, pp. 303–320, 1969.
[40] N. Parikh, S. Boyd *et al.*, "Proximal algorithms," *Foundations and trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.
[41] 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, 2009, pp. 139–147.
[42] ——, "Multibody graph transformations and analysis: part ii: closed-chain constraint embedding," *Nonlinear dynamics*, vol. 67, pp. 2153–2170, 2012.

[43] M. Chignoli, N. Adrian, S. Kim, and P. M. Wensing, "Recursive rigid-body dynamics algorithms for systems with kinematic loops," *arXiv preprint arXiv:2311.13732v1*, 2023.

[44] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[45] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[46] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[47] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "Prox-qp: Yet another quadratic programming solver for robotics and beyond," in *RSS 2022-Robotics: Science and Systems*, 2022.

[48] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[49] A. Chiche and J. C. Gilbert, "How the augmented lagrangian algorithm can deal with an infeasible convex quadratic optimization problem," *Journal of Convex Analysis*, vol. 23, no. 2, 2016.

[50] M. Yannakakis, "Computing the minimum fill-in is np-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[51] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, vol. 3, no. 3, pp. 255–269, 1957.

[52] A. George and J. W. Liu, *Computer solution of large sparse positive definite*. Prentice Hall Professional Technical Reference, 1981.

[53] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, 1969, pp. 157–172.

[54] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer methods in applied mechanics and engineering*, vol. 1, no. 1, pp. 1–16, 1972.

[55] M. Haddouni, V. Acary, S. Garreau, J.-D. Beley, and B. Brogliato, "Comparison of several formulations and integration methods for the resolution of daes formulations in event-driven simulation of nonsmooth frictionless multibody dynamics," *Multibody System Dynamics*, vol. 41, no. 3, pp. 201–231, 2017.

[56] G. Wanner and E. Hairer, *Solving ordinary differential equations II*. Springer Berlin Heidelberg New York, 1996, vol. 375.

[57] V. Brasey and E. Hairer, "Symmetrized half-explicit methods for constrained mechanical systems," *Applied numerical mathematics*, vol. 13, no. 1-3, pp. 23–31, 1993.

[58] G. Rodriguez, "Kalman filtering, smoothing, and recursive robot arm forward and inverse dynamics," *IEEE Journal on Robotics and Automation*, vol. 3, no. 6, pp. 624–639, 1987.

[59] M. C. Steinbach, "Tree-sparse convex programs," *Mathematical methods of operations research*, vol. 56, no. 3, pp. 347–376, 2003.

[60] G. Frison, D. Kouzoupis, M. Diehl, and J. B. Jørgensen, "A high-performance riccati based solver for tree-structured quadratic programs," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 399–14 405, 2017.

[61] R. Marti, S. Lucia, D. Sarabia, R. Paulen, S. Engell, and C. de Prada, "Improving scenario decomposition algorithms for robust nonlinear model predictive control," *Computers & Chemical Engineering*, vol. 79, pp. 30–45, 2015.

[62] J. J. Hench and A. J. Laub, "Numerical solution of the discrete-time periodic riccati equation," *IEEE transactions on automatic control*, vol. 39, no. 6, pp. 1197–1210, 1994.

[63] Y. Yang, "An efficient lqr design for discrete-time linear periodic system based on a novel lifting method," *Automatica*, vol. 87, pp. 383–388, 2018.

[64] M. Xie and F. Dellaert, "A unified method for solving inverse, forward, and hybrid manipulator dynamics using factor graphs," *arXiv preprint arXiv:1911.10065*, 2019.

[65] R. T. Rockafellar, "A dual approach to solving nonlinear programming problems by unconstrained optimization," *Mathematical programming*, vol. 5, no. 1, pp. 354–373, 1973.

[66] V. Acary, P. Armand, H. Minh Nguyen, and M. Shpakovych, "Second-order cone programming for frictional contact mechanics using interior point algorithm," *Optimization Methods and Software*, pp. 1–30, 2024.