Constrained Articulated Body Dynamics Algorithms

Ajay Sathya, Justin Carpentier

Constrained Articulated Body Dynamics Algorithms

Ajay Suresha Sathya[1,2] and Justin Carpentier[1]

***Abstract*—Rigid-body dynamics algorithms have played an essential role in robotics development. By finely exploiting the underlying robot structure, they allow the computation of the robot kinematics, dynamics, and related physical quantities with low complexity, enabling their integration into chipsets with limited resources or their evaluation at very high frequency for demanding applications (e.g., model predictive control, large-scale simulation, reinforcement learning, etc.) While most of these algorithms operate on constraint-free settings, only a few have been proposed so far to adequately account for constrained dynamical systems while depicting low algorithmic complexity. In this article, we introduce a series of new algorithms with reduced (and lowest) complexity for the forward simulation of constrained dynamical systems. Notably, we revisit the so-called articulated body algorithm (ABA) and the Popov-Vereshchagin algorithm (PV) in the light of proximal-point optimization and introduce two new algorithms, called constrainedABA and proxPV. These two new algorithms depict linear complexities while being robust to singular cases (e.g., redundant constraints, singular constraints, etc.). We establish the connection with existing literature formulations, especially the relaxed formulation at the heart of the MuJoCo and Drake simulators. We also propose an efficient and new algorithm to compute the damped Delassus inverse matrix with the lowest known computational complexity. All these algorithms have been implemented inside the open-source framework Pinocchio and depict, on a wide range of robotic systems ranging from robot manipulators to complex humanoid robots, state-of-the-art performances compared to alternative solutions of the literature.**

I. Introduction

Efficient rigid body dynamics algorithms [1] have played an essential role in robotics development. By finely exploiting the underlying robot structure to compute the robot kinematics, dynamics, and related physical quantities with low computational complexity, these algorithms enable dynamics evaluation in chip sets with limited resources and at high frequencies for demanding applications (e.g., computed torque control, model predictive control, large-scale simulation, reinforcement learning, etc.).

Most simulators [2]–[6] currently use such low-complexity algorithms only in constraint-free settings. Though a few low-complexity algorithms have been proposed for constrained dynamical systems, they suffer not only from being fairly complex to derive and implement. Still, they cannot adequately deal with singular cases (e.g., redundant constraints, singular constraints, etc.). Perhaps due to these issues, the usage of efficient constrained dynamics algorithms in simulators is currently low. Addressing these issues, we present a series of new constrained dynamics algorithms based on proximal algorithms in this paper that are simple and effectively handle singular cases. Our proximal perspective also presents a unified framework to interpret several existing disparate algorithms as special cases of the presented algorithms.

A. Related work

The first $O(n)$ complexity unconstrained forward dynamics algorithm, where n is the degrees of freedom (DoF) of the robot, was proposed by Vereshchagin in [7], but was unknown in west for decades. Their approach solved the linear quadratic regulator (LQR) problem associated with the Gauss' principle of least constraint (GPLC) [8], which is an optimization-based formulation of classical mechanics, using the dynamic programming (DP) principle [9]. Nearly a decade later, a practically identical algorithm [10] called the articulated body algorithm (ABA) was proposed, which was derived using a different perspective of cleverly propagating the solution of Newton-Euler equations through a kinematic tree. The ABA algorithm was further refined in [11] with insights such as improvements in computational efficiency through the use of local frames. Vereshchagin's LQR-ABA connection was also independently discovered by Western researchers in [12], where the similarities between the ABA algorithm and Kalman filter were recognized to derive ABA using filtering theory literature. However, the approach in [12] is fairly complex, and [7] (see [13] for an expository derivation) remains a significantly clearer connection between LQR and ABA.

These $O(n)$ complexity algorithms are efficient compared to computing the joint-space inertia matrix (JSIM) [14] and factorizing it, which has a complexity of $O(n^3)$. However, branching in kinematic trees induces sparsity in the JSIM, which can be exploited by the Cholesky decomposition (LTL algorithm) developed in [15] with $O(nd^2)$ complexity, where d is the depth of the kinematic tree. The LTL algorithm is faster than the $O(n)$ algorithms for small robots ($n < 7$) and can be competitive even for robots as large as quadrupeds (n 18), due to branching in the kinematic tree.

The unconstrained forward dynamics algorithm in [7] was quickly extended to account for motion constraints in the Popov-Vereshchagin algorithm (PV algorithm) [16], [17] with $O(n + m^2d + m^3)$ complexity, where m is the dimensionality of motion constraints, by encoding the constraints as equality constraints in the associated LQR problem. However, the PV algorithm remained sparsely known and used in the robotics community. Addressing this issue, a recent work [13] provides an accessible and expository derivation of the PV algorithm along with two original extensions, namely, PV-soft

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

[2]MECO Research Team, Department of Mechanical Engineering, KU Leuven and Flanders Make@KU Leuven, Belgium {ajay.sathya}@kuleuven.be

and PV-early algorithms, each with only $O(n+m)$ complexity. The PV-soft algorithm relaxes the motion constraints with a quadratic penalty, allowing it to obtain low complexity. However, it does not solve the constraints exactly. On the other hand, the PV-early algorithm exactly solves the motion constraints and yet achieves its low complexity by aggressively eliminating the Lagrange multipliers associated with motion constraints as early as possible. This prevents the accumulation of Lagrange multipliers during the DP recursions, that occurs in the original PV algorithm contributing to the $O(m^2 + m^3)$ complexity terms. The PV-early algorithm uses the singular value decomposition (SVD) [18] of a symmetric matrix at every joint, whose rank is equal to the respective joint's DoF, for early elimination of the Lagrange multipliers. While an efficient analytical formula exists for computing SVD of symmetric rank one (SR-1) matrices due to single DoF joints, the general applicability of the PV-early algorithm is impacted by the computational expense of SVD for multi-DoF joints. Moreover, the PV algorithm is sensitive to singular/redundant constraints, and the popular approach of Tikhonov regularization employed in [13] does not adequately solve constrained dynamics as it can bias solutions towards the origin and adversely affect constraint satisfaction.

The most widely used constrained dynamics algorithm for kinematic trees [19] solves the problem in joint space, resulting in a relatively higher computational complexity of $O(nd^2 + m^2d + md^2 + m^3)$ despite exploiting branching-induced sparsity. Recent work [20] generalized this approach to robustly handle singular/redundant constraints using proximal algorithms [21]. A similar proximal perspective can be applied to the low-complexity constrained dynamics algorithms to derive efficient and robust algorithms, forming the basis of this paper.

The main bottleneck in the joint-space constraint dynamics algorithms [19], [20] is the computation and factorization of the Delassus matrix [22], [23], also known as the inverse operational space inertia (OSIM) matrix [24], which itself has a computational complexity of $O(nd^2+m^2d+m^3)$ [19]. The Delassus matrix computation is crucial to many tasks, such as contact simulation, differentiating constrained dynamics, and operational space control, just to name a few. Therefore, unsurprisingly, several dedicated low-complexity algorithms have been proposed for computing it. One of the first such algorithms was the Kreutz-Jain-Rodriguez (KJR) algorithm [25], which used a three-sweep computation structure similar to ABA to achieve $O(n+m^2d+m^3)$. A further improvement on this propagation-based algorithm was proposed in EFPA [26] using extended force propagators (EFP) [27] to achieve a lower complexity of $O(n+md+m^3)$. Recent work [13] also showed that the PV algorithm computes the Delassus matrix as an intermediate quantity, which yielded another efficient algorithm, PV-OSIM, which requires only two computation sweeps, unlike EFPA or KJR. Despite its higher computational complexity of $O(n + m^2d + m^3)$, PV-OSIM was found to be more efficient [13] than the EFPA for most realistic robots. Finally, the PV-OSIM's computational complexity was further optimized in [28] by using the compositionality of EFPs to the lowest known complexity of $O(n + m^3)$. All the above algorithms have the m^3 complexity term due to the factorization of the Delassus matrix, which is generally dense. Specifically for robots with branching at the base (e.g., quadrupeds, humanoids), it was shown [13] that using matrix inversion lemma (MIL) [29] could accelerate the inverse Delassus matrix computation by up to 30%.

Though we restrict our scope in this paper to kinematic trees with motion constraints relative to the inertial ground frame, we mention constrained dynamics algorithms for closed loops for the sake of completeness. Most robot simulators do not cope with closed-loop mechanisms properly, and often, when they do, they resort to using inefficient methods like using general-purpose linear solvers to solve the contact KKT matrix. Recent work in [20] extended Featherstone's LTL algorithm [15], [19] to account for loop closure constraints with the same computational complexity of $O(n^2d+nd^2+md^2+m^3)$ as the original LTL algorithm and provided an implementation within the open source robot dynamics simulator library PINOCCHIO [6]. However, more efficient algorithms were proposed in the past, which can be an improvement over [20]. In the late 1980s, the PV algorithm was independently rediscovered and extended to a more general class of mechanisms, including those with internal closed loops in [30] and [31]. Both these algorithms are practically identical with the worst-case complexity of $O(n+m^2d+m^3)$ and employ loop-cutting with zero mass phantom links to be able to use a propagation-based algorithm similar in spirit to ABA. However, these algorithms are quite complex and are not yet supported in robotics simulators.

B. Contributions

Our paper contains five main contributions, three of which correspond to algorithmic contributions:

(i) **The constrainedABA algorithm:** We introduce a $O(n+m)$ complexity-constrained dynamics algorithm for trees called constrainedABA (cABA). Its derivation and implementation are significantly simpler than the existing $O(n+m)$ algorithms like the PV-early while being more efficient and more generally applicable to singular cases.

(ii) **The proxPV algorithm:** Revisiting the PV algorithm in the light of proximal-point optimization, we introduce proxPV that makes the PV algorithm robust to singular cases with minimal computational overhead.

(iii) **The cABA-OSIM algorithm:** We introduce the most efficient known algorithm with $O(n + m^2)$ complexity to compute the damped Delassus inverse matrix, called cABA-OSIM. In contrast, all the existing algorithms have an additional $O(m^3)$ complexity term in the general case due to the Delassus matrix factorization.

(iv) **Analysis and detailed benchmarking:** Our proximal point perspective presents a unifying framework highlighting connections between seemingly disparate solvers like constrainedABA, PV-early, and the soft Gauss' principle formulation at the heart of the MUJOCO simulator and PV-early algorithms. We present a detailed benchmarking of the different algorithms regarding the number of operations and computational speed for various robot models.

(v) **Open-source software implementation:** The constrainedABA algorithm proposed here is available to the community in the open-source software toolbox PINOCCHIO [6].

II. PROXIMAL REFORMULATION OF CONSTRAINED DYNAMICS

A. Notation

Lower-case symbols (x), bold-faced lower-case symbols ($\boldsymbol{x}$), and upper-case symbols (X) are scalars, vectors, and matrices, respectively. Let $\mathbb{S}^k_+$ and $\mathbb{S}^k_{++}$ denote the space of symmetric positive semidefinite and positive definite matrices of size $k \times k$.

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. $\mathcal{Q}$, $\mathcal{T}_{\mathbf{q}}\mathcal{Q}$ and n are the robot's configuration space, $\mathcal{Q}$'s tangent space at $\mathbf{q}$ and degrees of freedom (DoF) respectively. Let $\boldsymbol{\tau} \in \mathcal{T}^*_{\mathbf{q}}\mathcal{Q} \simeq \mathbb{R}^n$ be the generalized forces exerted on the robot. Let $\mathcal{S}$ be a topologically ordered list of link indices in a kinematic tree such that $i < j$ if the i^{th} link is an ancestor of the j^{th} link. By convention, we assume the 0^{th} link to be a global inertial frame. For floating base robots, 1^{st} link is the floating base.

We will use Featherstone's spatial algebra [1] to refer to rigid body quantities. The 6D spatial velocity and acceleration of a rigid body i is $\mathbf{v}_i \in \mathbb{M}^6$ and, $\mathbf{a}_i \in \mathbb{M}^6$ respectively, where $\mathbb{M}^6$ is a 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 $\mathbb{M}^6$. $H_i \in \mathbb{I}^{6\times6}$ is the spatial inertia (represented as a 6×6 symmetric positive definite matrix) that maps $\mathbb{M}^6$ to $\mathbb{F}^6$. $\times$ and $\times^*$ are the cross-product operators on motion vectors and force vectors. Refer [1] for more details on the spatial algebra. Table I gives a comprehensive list of the symbols used all over the paper.

B. Constrained dynamics in generalized coordinates

Constrained dynamics. According to Gauss' principle of least constraint [8], [32], [33], the acceleration $\dot{\boldsymbol{\nu}}$ that a robot system at state $(\mathbf{q}, \boldsymbol{\nu})$ undergoes, when constrained holonomically/non-holonomically by the equation,

$$\mathbf{f}_c(\mathbf{q}, \boldsymbol{\nu}) = 0, \tag{1}$$

and acted upon by $\boldsymbol{\tau}$ forces, is the minimizer of the following equality-constrained strongly convex quadratic program (QP):

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

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

where $M(\mathbf{q}) \in \mathbb{S}^n_{++}$ is the joint-space inertia matrix (JSIM) and $\dot{\boldsymbol{\nu}}_{\text{free}}(\mathbf{q}, \dot{\boldsymbol{\nu}}, \boldsymbol{\tau})$ is the joint-space acceleration in the absence of constraints

$$\dot{\boldsymbol{\nu}}_{\text{free}}(\mathbf{q}, \dot{\boldsymbol{\nu}}, \boldsymbol{\tau}) := M^{-1}(\mathbf{q})\left(\boldsymbol{\tau} - \mathbf{c}(\mathbf{q}, \boldsymbol{\nu})\right), \tag{3}$$

where $\mathbf{c}(\mathbf{q}, \boldsymbol{\nu})$ is the generalized force vector due to gravity, Coriolis and centripetal effects. Eq. (2b) is the result of differentiating Eq. (1) twice (once for non-holonomic constraints)

Symbol	Meaning
d	Depth of the kinematic tree.
$\mathbf{q}$	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 $\mathbf{q}$.
$\mathcal{S}$	Topologically ordered list of tree link indices.
$\mathbf{v}_i$	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.
$\mathbf{f}_c$	Motion constraint function in generalized coordinate space.
M	Robot inertia matrix in generalized coordinates (JSIM).
$\dot{\boldsymbol{\nu}}_{\text{free}}$	Generalized accelerations when unconstrained.
$\boldsymbol{a}^*_c$	Desired constraint accelerations.
$\mathbf{J}_{\mathbf{f}_c}$	Geometric Jacobian of $\mathbf{f}_c$.
$\mathbf{c}$	Generalized forces due to gravity, centripetal and Coriolis effects.
$\boldsymbol{\gamma}_{\mathbf{f}_c}$	Constraint accelerations due to Coriolis effects.
$\boldsymbol{\lambda}$	Lagrange multipliers and constraint force magnitudes.
$\mathcal{L}$	Lagrangian of GPLC.
Λ	OSIM or inverse Delassus matrix.
μ	Proximal operator parameter.
$\mathbf{g}$	Dual function of GPLC.
Λ_μ	Damped Delassus inverse matrix.
M_μ	Constraint augmented Inertia matrix.
S_i	Spans i^{th} joint's motion subspace.
$\mathbf{a}_{b,i}$	i^{th} link's bias acceleration vector.
K_i	i^{th} link's constraint matrix.
$\mathbf{k}$	i^{th} link's desired constraint accelerations.
$\mathbf{a}_{\text{grav}}$	Spatial acceleration-due-to-gravity vector.
$\pi(i)$	i^{th} link's parent link.
$\gamma(i)$	Set of i^{th} link's children links.
$\tilde{H}_i$	i^{th} link's constraint augmented inertia tensor Eq. (29).
$\tilde{\mathbf{f}}_i$	i^{th} link's constraint augmented force vector Eq. (29).
$\text{desc}(i)$	Set of all descendants of the i^{th} link.
H^A_i	i^{th} link's constrained articulated body inertia.
$\mathbf{f}^A_i$	i^{th} link's resultant force due to all constraints and forces on descendant links.
D_i	Apparent constrained inertia felt at the i^{th} joint.
P_i	Backward force propagation matrix at the i^{th} joint.
$\text{ES}(i)$	Set of motion-constrained links among i^{th} link's descendants.
$\mathcal{S}_r$	List $\mathcal{S}$ reversed.
$\mathcal{E}$	Set of all links that are motion-constrained (end-effectors).
$\mathcal{C}(\cdot)$	Returns cardinality of a set.
$\mathcal{S}_\mathcal{E}$	Sublist of $\mathcal{S}$, whose links support an end-effector.
$\mathcal{S}_{\mathcal{E}r}$	List $\mathcal{S}_\mathcal{E}$ reversed.
$\Delta\mathbf{f}_i$	Force update vector in constrainedABA reduced sweep.
K^A_i	Constraint matrix felt at i^{th} link for all $\text{ES}(i)$ constraints.
L^A_i	Intermediate term to recursively accumulate Λ^{-1}.
R_E^{-1}	Diagonal constraint weighting matrix in soft Gauss' principle.
${}^jP_i$	EFP - backward propagates i^{th} link's forces to j^{th} link.
$\text{cca}(i,j)$	i^{th} and j^{th} links' closest common ancestor.
${}^j\Omega_i$	Spatial inverse inertia of the i^{th} link considering a sub-tree rooted at the j^{th} link.
$\mathcal{N}$	Set of branching links supporting strictly more of constraints than any child link.
${}^jK_i$	Constraint-space EFP.
${}^iL_{e_j,e_k}$	Delassus matrix block for e_j and e_k constraints for a sub-tree rooted at i^{th} link.

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

with respect to (w.r.t) to time. $J_{\mathbf{f}_c}(\mathbf{q}) \in \mathbb{R}^{m\times n}$ is the geometric Jacobian of the constraints, $\dot{J}_{\mathbf{f}_c}(\mathbf{q},\boldsymbol{\nu}) \in \mathbb{R}^{m\times n}$ is its time-derivative and $\boldsymbol{a}_c^* \in \mathbb{R}^m$ is the desired constraint acceleration. While $\boldsymbol{a}_c^* = \mathbf{0}_m$ for the constraint in Eq. (1) in theory, in practice the constraints accumulate numerical integration errors during simulation. They can be stabilized by including a constraint error (residual of Eq. (1)) negative feedback term in $\boldsymbol{a}_c^*$, with methods like Baumgarte's stabilization [34]. Let

$$\boldsymbol{\gamma}_{\mathbf{f}_c}(\mathbf{q},\boldsymbol{\nu}) := \dot{J}_{\mathbf{f}_c}(\mathbf{q},\boldsymbol{\nu})\boldsymbol{\nu}, \tag{4}$$

be the bias accelerations. The variable dependencies will be dropped for brevity whenever obvious from the context.

Constrained dynamics Lagrangian. The solution to the QP above is the primal-dual saddle point of the Lagrangian [35]

$$(\dot{\boldsymbol{\nu}}^*, \boldsymbol{\lambda}^*) = \mathbf{arg}\ \max_{\boldsymbol{\lambda}}\ \min_{\dot{\boldsymbol{\nu}}}\ \mathcal{L}(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}), \tag{5}$$

where

$$\mathcal{L}(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) := \frac{1}{2}\|\dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_{\text{free}}\|_M^2 + \boldsymbol{\lambda}^T(J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*), \tag{6}$$

and $\boldsymbol{\lambda} \in \mathbb{R}^m$ are the Lagrange multipliers (also called dual variables) of the QP in Eq. (2).

Since Eq. (2) is an equality-constrained QP, its primal-dual saddle point is immediately given by solving the Karush-Kuhn-Tucker (KKT) [35] system implied by the first-order necessary optimality conditions [36],

$$\begin{bmatrix} M & J_{\mathbf{f}_c}^T \\ J_{\mathbf{f}_c} & 0_{m\times m} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\nu}} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} M\dot{\boldsymbol{\nu}}_{\text{free}} \\ -\boldsymbol{\gamma}_{\mathbf{f}_c} + \boldsymbol{a}_c^* \end{bmatrix}. \tag{7}$$

Solving the KKT system. Because M is positive definite, the KKT system in Eq. (7) is most often solved by eliminating $\dot{\boldsymbol{\nu}}$ using

$$\dot{\boldsymbol{\nu}} = \dot{\boldsymbol{\nu}}_{\text{free}} - M^{-1}J_{\mathbf{f}_c}^T\boldsymbol{\lambda}, \tag{8}$$

back-substituting which in Eq. (5), gives the dual function (after ignoring constant terms)

$$g(\boldsymbol{\lambda}) = -\frac{1}{2}\boldsymbol{\lambda}^T\Lambda^{-1}\boldsymbol{\lambda} + \left(J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}}_{\text{free}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*\right)^T\boldsymbol{\lambda}, \tag{9}$$

where $\Lambda^{-1}(\mathbf{q}) := J_{\mathbf{f}_c}M^{-1}J_{\mathbf{f}_c}^T$ is the so-called Delassus matrix [22], [23], also known as the inverse operational space inertial matrix (inverse OSIM) [24].

Remark 1. *$\Lambda^{-1} \in \mathbb{S}_+^m$ and the dual function above is concave. Furthermore, if $J_{\mathbf{f}_c}$ is full row-rank, $\Lambda^{-1} \in \mathbb{S}_{++}^m$, the dual function $g(\boldsymbol{\lambda})$ is strongly concave, Λ^{-1} is invertible and the OSIM matrix Λ exists.*

The dual function is maximized to obtain the optimal Lagrange multipliers by solving the linear equation

$$\Lambda^{-1}\boldsymbol{\lambda}^* = J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}}_{\text{free}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*. \tag{10}$$

The frequently used approach to solve the equation above is computing M using the composite rigid body algorithm (CRBA) [14] and its Cholesky factorization using Featherstone's LTL algorithm [15], which exploits branching-induced sparsity. Then, the LTL algorithm is exploited to efficiently compute Λ^{-1} as well [19], which is finally factorized using dense Cholesky factorization. However, as mentioned in Section I-A, this approach is computationally expensive with a total computational complexity of $O(nd^2 + m^2d + md^2 + m^3)$, where d is the depth of the kinematic tree, n the number of joints and m is the number of constraints.

Nonuniqueness of solutions. In practice, $J_{\mathbf{f}_c}$ often does not have full rank or loses it due to redundant constraints or kinematic singularities, making the Λ^{-1} singular. This prevents the use of efficient algorithms like Cholesky factorization. Typically, Tikhonov regularization (adding a positive definite diagonal matrix to Λ^{-1}) is employed to address this issue, but Tikhonov regularization biases the constraint forces towards the origin, preventing constraint satisfaction and can pose numerical issues if the regularization parameter is too small. An alternate approach that does not introduce such bias is the truncated singular value decomposition (SVD) [18]. However, this is computationally more expensive than Cholesky factorization and is therefore not commonly used.

C. Proximal point algorithm

An exact and efficient alternative to Tikhonov regularization or SVD that we will leverage is the proximal point algorithm (PPA) [21], [37], which is effective for robotics problems [20], [38], [39], and most often requiring few iterations (each of which is efficient) to converge for robot dynamics problems [20].

We first introduce the proximal operator of a convex function, $f : \mathbb{R}^n \to \mathbb{R}$

$$\text{prox}_{\mu,f}(\mathbf{x}^k) := \quad \mathbf{arg}\ \min_{\mathbf{x}}\ \left\{ f(\mathbf{x}) + \frac{1}{2\mu}\|\mathbf{x} - \mathbf{x}^k\|^2 \right\}, \tag{11}$$

where $\mu > 0 \in \mathbb{R}$ is the proximal operator parameter, which can be interpreted as a step-size [21]. The PPA minimizes $f(\mathbf{x})$ via fixed-point iterations on the proximal operator until a termination criterion is met

$$\mathbf{x}^{k+1} = \text{prox}_{\mu,f}(\mathbf{x}^k). \tag{12}$$

D. Constrained dynamics using the two QP approaches

We now describe two different approaches to solve the QP in Eq. (2), that are mathematically equivalent but differ in their computational cost.

1) Dual proximal point method (proxLTL): Applying PPA to optimize the dual function in Eq. (9) gives

$$\boldsymbol{\lambda}^{k+1} = \text{prox}_{\mu,-g}(\boldsymbol{\lambda}^k) = \mathbf{arg}\ \min_{\boldsymbol{\lambda}}\ -g(\boldsymbol{\lambda}) + \frac{1}{2\mu}\|\boldsymbol{\lambda} - \boldsymbol{\lambda}^k\|^2, \tag{13a}$$

$$= \Lambda_\mu\left(J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}}_{\text{free}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^* + \frac{1}{\mu}\boldsymbol{\lambda}^k\right), \tag{13b}$$

where $\Lambda_\mu^{-1} := \Lambda^{-1} + \frac{1}{\mu}I$ is the damped Delassus matrix. $\Lambda_\mu^{-1} \in \mathbb{S}_{++}^m$ and can be factorized efficiently using Cholesky decomposition. This factorization is computed once and re-used, making each fixed point iteration fast. This algorithm,

which we will call proxLTL, has already been implemented in the PINOCCHIO library [20] and solves the motion constraints exactly unlike Tikhonov regularization.

Remark 2. *All the algorithmic developments in a paper can be trivially extended to the case $\Lambda_\mu^{-1} := \Lambda^{-1} + R_E^{-1}$, where R_E is a positive definite diagonal matrix. Such a choice can permit variable weights on different constraints.*

Assuming that the constraints are correctly modeled for physical systems, the primal problem in Eq. (2) is always feasible (meaning that the constraints are satisfiable). However, numerically, the primal problem in Eq. (2) can be infeasible, especially when there are redundant constraints and Baumgarte terms [34] are added to $\boldsymbol{a}_c^*$ for constraint stabilization. Then, the dual problem is unbounded above, and the PPA iterations do not converge. However, in this situation, it has been shown [40]–[42] that the primal residual converges to a desirable value that is optimal in the least squares sense during the PPA iterations. Therefore, the termination criteria for the convergence of the PPA should monitor the least-squares error given by

$$\|\Lambda^{-T}(\Lambda^{-1}\boldsymbol{\lambda} - [J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}}_{\text{free}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*])\|_\infty. \tag{14}$$

Advantageously, the PPA algorithm converges to the optimal value for feasible problems and to the least squares solution for numerically infeasible problems automatically without any modification to the algorithm.

2) Augmented Lagrangian method (proxLTLs): An alternative to proxLTL that can solve the QP in Eq. (2) exactly is the augmented Lagrangian method [43], [44] (ALM), where the augmented Lagrangian function is

$$\mathcal{L}^A(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) := \mathcal{L}(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) + \frac{\mu}{2}\|J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*\|^2. \tag{15}$$

ALM iterations alternately optimize $\mathcal{L}^A$ over its primal and dual variables

$$\dot{\boldsymbol{\nu}}^{k+1} = M_\mu^{-1}\left\{M\dot{\boldsymbol{\nu}}_{\text{free}} - J_{\mathbf{f}_c}^T\left(\boldsymbol{\lambda}^k + \mu\left(\boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*\right)\right)\right\}, \tag{16a}$$

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mu\left(J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}}^{k+1} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*\right), \tag{16b}$$

where $M_\mu := M + \mu J_{\mathbf{f}_c}^T J_{\mathbf{f}_c}$ is the augmented JSIM, with the influence of the constraints from the quadratic term in the augmented Lagrangian function. This formulation, which we will call proxLTLs, is interesting as M_μ can be efficiently computed with a small modification to the CRBA (see Section II-F) and efficiently factorized using the LTL algorithm, leading to an algorithm that has a lower computational complexity of $O(nd^2 + md)$, which is significantly faster than the $O(nd^2 + m^2d + md^2 + m^3)$ complexity of the proxLTL algorithm. This difference arises because proxLTLs does not compute or factorize the Λ^{-1} matrix. While the idea of proxLTLs was briefly mentioned in [20], it was neither implemented nor explored further, which we will do in this paper.

3) Both approaches are equivalent: ProxLTL and proxLTLs are mathematically equivalent because PPA applied to the dual problem and the ALM are equivalent [41], [45]. Theoretically, the sequence of $\boldsymbol{\lambda}^k$ they compute are equal. However, they differ numerically and computationally, with the difference arising due to the order in which the primal and dual variables are optimized. We now show the equivalence of proxLTL and proxLTLs by recovering the ALM iterations in Eq. (16a) and Eq. (16b) from the PPA iterations in Eq. (13)).

ProxLTL in Eq. (13) maximizes the dual function $g(.)$, that is obtained after minimizing the Lagrangian primal variables in the saddle point problem (see Eq. (5)). However, since each PPA iteration solves a feasible sub-problem that is strongly convex and strongly concave in primal and dual variables, respectively with a unique primal-dual solution, this elimination ordering can be reversed [46]

$$\text{prox}_{\mu,\mathcal{L}'}(\boldsymbol{\lambda}^k) := \underset{\boldsymbol{\lambda}}{\textbf{arg}}\ \underset{\dot{\boldsymbol{\nu}}}{\textbf{min}}\ \underset{\boldsymbol{\lambda}}{\textbf{max}}\ \mathcal{L}(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) - \frac{1}{2\mu}\|\boldsymbol{\lambda} - \boldsymbol{\lambda}^k\|^2, \tag{17}$$

The maximizer over $\boldsymbol{\lambda}$ is obtained by solving for gradient stationarity

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \mu\left(J_{\mathbf{f}_c}\dot{\boldsymbol{\nu}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*\right), \tag{18}$$

substituting which back in Eq. (17) in the place of $\boldsymbol{\lambda}$ and optimizing for the primal variables $\dot{\boldsymbol{\nu}}$ gives

$$\dot{\boldsymbol{\nu}}^{k+1} = M_\mu^{-1}\left\{M\dot{\boldsymbol{\nu}}_{\text{free}} - J_{\mathbf{f}_c}^T\left(\boldsymbol{\lambda}^k + \mu\left(\boldsymbol{\gamma}_{\mathbf{f}_c} - \boldsymbol{a}_c^*\right)\right)\right\}, \tag{19}$$

which can be substituted back in Eq. (18) to compute the numerical value of $\boldsymbol{\lambda}^{k+1}$. Comparing the equations Eq. (18) and Eq. (19) with Eq. (16b) and Eq. (16a) verifies that proxLTL iterations and the ALM iterations are identical.

E. Constrained dynamics in maximal coordinates

Both proxLTL and proxLTLs algorithms have lower complexity counterparts that can be derived by applying dynamic programming (DP) on the problem of Gauss' principle in the so-called maximal coordinates [33]

$$\underset{\dot{\boldsymbol{\nu}}, \boldsymbol{a}}{\textbf{minimize}} \quad \sum_{i=1}^{n_b}\left\{\frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a}_i\right\} \tag{20a}$$

$$\textbf{subject to} \quad \mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i\dot{\boldsymbol{\nu}}_i + \mathbf{a}_{b,i}, \quad i = 1, 2, \ldots, n_b, \tag{20b}$$

$$K_i\mathbf{a}_i = \mathbf{k}_i, \quad i = 1, 2, \ldots, n_b, \tag{20c}$$

$$\mathbf{a}_0 = -\mathbf{a}_{\text{grav}}, \tag{20d}$$

where $\mathbf{f}_i$ is the resultant spatial force acting on the i^{th} link including the bias forces ($-\mathbf{v}_i \times^* H_i\mathbf{v}_i$). All the spatial quantities are expressed in the inertial frame in our subsequent derivations for simplicity of notation. Since the GPLC is expressed in the maximal coordinates here, unlike the problem in Eq. (2), the joint constraints are also included in Eq. (20b). $\pi(i)$ is the parent link of the i^{th} link in the kinematic tree, $\boldsymbol{\nu}_i \in \mathbb{R}^{n_i}$ is the i^{th} joint's generalized velocities, $\dot{\boldsymbol{\nu}}_i \in \mathbb{R}^{n_i}$ is the i^{th} joint's generalized accelerations. S_i is the i^{th} joint's motion subspace matrix of size $6 \times n_i$, with n_i being the i^{th} joint's DoF. Each column vector of S_i is an element in $\mathbb{M}^6$.

$$\mathbf{a}_{b,i} := \mathbf{v}_i \times S_i\boldsymbol{\nu}_i, \tag{21}$$

is the i^{th} link's bias acceleration. The motion constraints are encoded in Eq. (20c), where K_i, expressed as a matrix in $\mathbb{R}^{m_i \times 6}$ and $\mathbf{k}_i \in \mathbb{R}^{m_i}$, are the constraint matrix and the desired

constraint accelerations. Each row vector of K_i is an element in $\mathbb{F}^6$. A uniform acceleration field of minus acceleration-due-to-gravity is added to all the links by fixing the root node acceleration in Eq. (20d). This trick [11] spares us from having to add the weight of each link to $\mathbf{f}_i$ thereby providing some computational speed up.

We now connect the constraints in Eq. (20c) to the constraints in the joint-space formulation in Eq. (2b). Connection in the other direction is not as interesting, since the constraints are more naturally formulated in the maximal coordinates. Suppose that $J_i \in \mathbb{R}^{6\times n}$ is the geometric Jacobian of the i^{th} link, mapping the generalized velocity $\boldsymbol{\nu}$ to the joint spatial velocity $\mathbf{v}_i$. Then,

$$J_{f_c}(\mathbf{q}) := \begin{bmatrix} K_1 J_1 \\ \vdots \\ K_{n_b} J_{n_b} \end{bmatrix}, \quad \boldsymbol{a}_c^*(\mathbf{q}, \boldsymbol{\nu}) := \begin{bmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_{n_b} \end{bmatrix} \tag{22}$$

F. Accelerating M_μ computation.

We now discuss the connection between the link inertias H_i, the JSIM M and the augmented JSIM M_μ. From [47, Eq. 8.57], we have

$$M = \begin{bmatrix} J_1 \\ \vdots \\ J_n \end{bmatrix}^T \begin{bmatrix} H_1 & & \\ & \ddots & \\ & & H_n \end{bmatrix} \begin{bmatrix} J_1 \\ \vdots \\ J_n \end{bmatrix}. \tag{23}$$

Considering the definition of M_μ and Eq. (22), we get the equation

$$M_\mu = \begin{bmatrix} J_1 \\ \vdots \\ J_n \end{bmatrix}^T \begin{bmatrix} H_1 + \mu K_1^T K_1 & & \\ & \ddots & \\ & & H_n + \mu K_n^T K_n \end{bmatrix} \begin{bmatrix} J_1 \\ \vdots \\ J_n \end{bmatrix} \tag{24}$$

The equation above implies that the inertia of every constrained link can be updated as above and the CRBA can be called to compute M_μ directly. This provides a minor speed-up compared to first separately computing M using CRBA and later adding the $\mu J_{f_c}^T J_{f_c}$ terms. In the rest of this paper, we will use this trick to speed up M_μ computation in the proxLTLs algorithm.

G. The PV algorithm

We now briefly review the PV algorithm [16], [17], an efficient constrained dynamics algorithm with its origin in the 1970s, see [13] for an expository derivation. It is a low-complexity counterpart to Featherstone's LTL algorithm, with a $O(n + m^2 d + m^3)$ complexity. Similarly to the LTL approach, it first eliminates the primal variables from the Lagrangian of the optimization problem in Eq. (20), but by using DP, to obtain the dual function $g(\boldsymbol{\lambda})$. The dual function is optimized to compute the optimal dual values, which is then used to compute the resulting accelerations $\mathbf{a}_i$ during a roll-out. It has a three-sweep structure analogous to ABA. However, the PV algorithm assumes the invertibility of Λ^{-1}, and Tikhonov regularization was used in [13] to make the algorithm applicable to singular cases. However, this biases the constraint forces to be closer to the origin, so the motion constraints will not be satisfied accurately.

III. Constrained ABA derivation

In this section, we will derive the constrainedABA algorithm, one of the main contributions of this paper. As further discussed later, constrainedABA is the low-complexity, recursive analog of the proxLTLs algorithm (see Section II-D2) derived from the maximal coordinate formulation of GPLC (see Eq. (20)). Therefore, similarly to proxLTLs, we will solve the QP in Eq. (20) using the ALM. To do this, we will closely follow the DP approach used in the PV algorithm derivation in [13], which applied DP on the Lagrangian, resulting in a non-iterative algorithm composed of three sweeps. In contrast, we will apply DP on the augmented Lagrangian, which gives a similar three-sweep algorithm, and propose two efficient reduced sweeps associated with the ALM iterations.

All the link acceleration terms $\mathbf{a}_i$s can be expressed as a function of the joint accelerations $\dot{\boldsymbol{\nu}}_i$s and the base link acceleration $\mathbf{a}_0$, and these constraints will be eliminated through substitution using Eq. (20b) and Eq. (20d). Therefore, we include only the motion constraints Eq. (20c) as the joint constraints in the Lagrangian of Eq. (20) defined as

$$\mathcal{L}_m(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) := \sum_{i=1}^{n_b} \left\{ \frac{1}{2} \mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a_i} + \boldsymbol{\lambda}_i^T \left(K_i \mathbf{a}_i - \mathbf{k}_i \right) \right\}, \tag{25}$$

and the augmented Lagrangian is

$$\mathcal{L}_m^A(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) := \mathcal{L}_m(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) + \frac{\mu}{2} \sum_{i=1}^{n_b} \| K_i \mathbf{a}_i - \mathbf{k}_i \|^2. \tag{26}$$

Similarly to Section II-D2, applying ALM to Eq. (20) provides the update equations

$$\dot{\boldsymbol{\nu}}^{k+1} = \min_{\dot{\boldsymbol{\nu}}} \mathcal{L}_m^A(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}^k), \tag{27a}$$

$$\boldsymbol{\lambda}_i^{k+1} = \boldsymbol{\lambda}_i^k + \mu \left\{ K_i \mathbf{a}_i^{k+1} - \mathbf{k}_i \right\}, \quad \forall i \in \mathcal{S}, \tag{27b}$$

where $\boldsymbol{\lambda}_i$ are the dual variables corresponding to the i^{th} link's motion constraints. We first solve Eq. (27a) leveraging the dynamic programming principle.

A. DP-based derivation

Eq. (27a) is an unconstrained QP over $\dot{\boldsymbol{\nu}}$. As a first step to solving it, we collect all the $\mathcal{L}_m^A$ terms (see Eq. (25) and Eq. (26)) that are quadratic and linear in $\mathbf{a}_i$ to obtain

$$\mathcal{L}_m^A \left(\dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}^k \right) = \sum_{i=1}^{n_b} \Bigg\{ \frac{1}{2} \mathbf{a}_i^T \left(H_i + \mu K_i^T K_i \right) \mathbf{a}_i - \left(\mathbf{f}_i + K_i^T \left(\mu \mathbf{k}_i - \boldsymbol{\lambda}_i^k \right) \right)^T \mathbf{a}_i + \text{const} \Bigg\}, \tag{28}$$

where we remind readers that all the $\mathbf{a}_i$s are functions of $\dot{\boldsymbol{\nu}}$, and 'const' consists of all the remaining terms in $\mathcal{L}_{\Updownarrow}^A$ that do not depend on $\mathbf{a}_i$. Considering the algebraic similarity of equation above with Eq. (20), let us 'update' the H_i and $\mathbf{f}_i$ terms

$$\tilde{H}_i := H_i + \mu K_i^T K_i, \quad \tilde{\mathbf{f}}_i := \mathbf{f}_i + K_i^T \left(\mu \mathbf{k}_i - \boldsymbol{\lambda}_i^k \right), \tag{29}$$

to get the unconstrained QP problem

$$\begin{aligned} \underset{\dot{\boldsymbol{\nu}},\boldsymbol{a}}{\textbf{minimize}} \quad & \sum_{i=1}^{n_b} \left\{ \frac{1}{2} \mathbf{a}_i^T \tilde{H}_i \mathbf{a}_i - \tilde{\mathbf{f}}_i^T \mathbf{a_i} \right\}, \\ \text{where} \quad & \mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i \dot{\boldsymbol{\nu}}_i + \mathbf{a}_{b,i}, \quad i = 1, 2, \ldots, n_b, \\ \text{and} \quad & \mathbf{a}_0 = -\mathbf{a}_{\text{grav}}. \end{aligned} \tag{30}$$

The problem above is algebraically identical to an unconstrained forward dynamics problem and can be readily solved using the ABA. For completeness, we now derive the ABA algorithm using Vereshchagin's DP approach [7], which is algebraically identical to the Riccati recursion derived for the standard discrete-time linear quadratic regulator (LQR) problem [13], [48]. Using the terminology from LQR literature, let us define the 'cost-to-go' function

$$V_i(\mathbf{a}_i, \dot{\boldsymbol{\nu}}_{\text{desc}(i)}) := \sum_{j=\{i\}\cup \text{desc}(i)} \left\{ \frac{1}{2} \mathbf{a}_j^T \tilde{H}_j \mathbf{a}_j - \tilde{\mathbf{f}}_j^T \mathbf{a_j} \right\}, \tag{31}$$

as the GPLC objective considering the i^{th} link and all its descendants. Then the optimal cost-to-go function is

$$V_i^*(\mathbf{a}_i) := \underset{\dot{\boldsymbol{\nu}}_{\text{desc}(i)}}{\textbf{min}} \quad V_i(\mathbf{a}_i, \dot{\boldsymbol{\nu}}). \tag{32}$$

With LQR being one of the few problems where DP is computationally tractable, we know that $V_i^*(\mathbf{a}_i)$ is exactly parametrized as quadratic form (see [48]), which we will hypothesize

$$V_i^*(\mathbf{a}_i) = \frac{1}{2} \mathbf{a}_i^T H_i^A \mathbf{a}_i - \mathbf{f}_i^{AT} \mathbf{a}_i. \tag{33}$$

From Bellman's recursion [9], the optimal cost-to-go function of a link and its children are related as

$$V_i^*(\mathbf{a}_i) = \underset{\dot{\boldsymbol{\nu}}_{\gamma(i)}}{\textbf{min}} \left\{ \frac{1}{2} \mathbf{a}_i^T \tilde{H}_i \mathbf{a}_i - \tilde{\mathbf{f}}_i^T \mathbf{a}_i + \sum_{j \in \gamma(i)} V_j^*(\mathbf{a}_j) \right\} \tag{34}$$

$$= \underset{\dot{\boldsymbol{\nu}}_j \in \gamma(i)}{\textbf{min}} \left\{ \frac{1}{2} \mathbf{a}_i^T \tilde{H}_i \mathbf{a}_i - \tilde{\mathbf{f}}_i^T \mathbf{a}_i + \sum_{j \in \gamma(i)} V_j^*(\mathbf{a}_i + S_j \dot{\boldsymbol{\nu}}_j + \mathbf{a}_{b,j}) \right\}. \tag{35}$$

Substituting the assumed quadratic form from Eq. (33) for all the $V_j^*(.)$ terms above yields

$$V_i^*(\mathbf{a}_i) = \underset{\dot{\boldsymbol{\nu}}_{\gamma(i)}}{\textbf{min}} \Bigg\{ \frac{1}{2} \mathbf{a}_i^T \tilde{H}_i \mathbf{a}_i - \tilde{\mathbf{f}}_i^T \mathbf{a}_i + \sum_{j \in \gamma(i)} \Big[(\mathbf{a}_i + S_j \dot{\boldsymbol{\nu}}_j + \mathbf{a}_{b,j})^T H_j^A (\mathbf{a}_i + S_j \dot{\boldsymbol{\nu}}_j + \mathbf{a}_{b,j}) - \mathbf{f}_j^{AT} (\mathbf{a}_i + S_j \dot{\boldsymbol{\nu}}_j + \mathbf{a}_{b,j}) \Big] \Bigg\}. \tag{36}$$

Collecting all the terms involving $\dot{\boldsymbol{\nu}}_j$ gives a positive definite quadratic function in $\dot{\boldsymbol{\nu}}_j$, which is minimized by solving for gradient stationarity condition

$$\dot{\boldsymbol{\nu}}_j^* = D_j^{-1} S_j^T \left\{ \mathbf{f}_j^A - H_j^A (\mathbf{a}_i + \mathbf{a}_{b,j}) \right\}, \tag{37}$$

where $D_j := S_j^T H_j^A S_j$ and $D_i \in \mathbb{S}_{++}^{n_i}$. Substituting $\dot{\boldsymbol{\nu}}_j^*$'s expression from above back in Eq. (36) results in a quadratic expression in $\mathbf{a}_i$ that is identical to the hypothesized quadratic form in Eq. (33), with quadratic form's coefficients given by the recursive update equations

$$H_i^A = \tilde{H}_i + \sum_{j \in \gamma(i)} P_j H_j^A, \tag{38a}$$

$$\mathbf{f}_i^A = \tilde{\mathbf{f}}_i + \sum_{j \in \gamma(i)} P_j \left\{ \mathbf{f}_j^A - H_j^A \mathbf{a}_{b,j} \right\}, \tag{38b}$$

where $P_j := I_{6\times 6} - H_j^A S_j D_j^{-1} S_j^T$ is the projection matrix [10] that propagates inertia and force quantities from a child link to its parent link after subtracting the component that causes the child joint's motion. For all leaf-nodes j in the kinematic tree, we set $H_j^A = \tilde{H}_j$ and $\mathbf{f}_j^A = \tilde{\mathbf{f}}_j$, following which we compute these terms for all the non-leaf links tree recursively using Eq. (38). Thus, we can inductively show that the cost-to-go function hypothesis in Eq. (33) is valid for all links.

After the recursion in Eq. (38) reaches the root node 0 of the tree, the value of $-\mathbf{a}_{\text{grav}}$ is substituted in place of $\mathbf{a}_0$, and in a forward-sweep from root to leaves (also called roll-out in DP), the resulting joint accelerations and link accelerations are computed using Eq. (37) and Eq. (20b) respectively.

Unsurprisingly, the recursive equations in Eq. (38) are similar to the ABA [10], as constraint-related terms were absorbed into inertia and force terms in Eq. (29). Despite the algebraic similarity, the H_i^A and $\mathbf{f}_i^A$ terms in Eq. (38) may not be interpreted as the physical articulated body inertia and the resultant spatial forces because of the mathematical modification in Eq. (29) due to the augmented Lagrangian terms of the constraints.

Details on $\mathbf{f}_i$: As explained in [13]

$$\mathbf{f}_i = T_i \boldsymbol{\tau}_i - \mathbf{v}_i \times^* H_i \mathbf{v}_i + \mathbf{f}_i^{\text{ext}} - \sum_{j \in \gamma(i)} T_j \boldsymbol{\tau}_j,$$

and note that T_i does not need to be explicitly computed because the algorithm only requires inner product of spatial forces with S_i in Eq. (37), where T_i gets cancelled since $S_i^T T_i = I_{n_i}$. Note that, total resultant force due to $\boldsymbol{\tau}_j$ on its parent link i, is $-T_j \boldsymbol{\tau}_j + P_j T_j \boldsymbol{\tau}_j$ by including also the backpropagation term from Eq. (38), which evaluates to $-H_j^A S_j D_j^{-1} \boldsymbol{\tau}_j$, thereby not requiring T_j computation for backpropagation as well.

IV. Constrained ABA algorithm

We now present constrainedABA derived in the previous section in an algorithmic form. Introducing some notation, let $\text{ES}(i)$ be the set of indices of the end-effector supported by the i^{th} link, which implies that these end-effectors are descendants of the i^{th} link that are in $\mathcal{E}$. Let $\mathcal{S}_\mathcal{E}$ be the sublist of $\mathcal{S}$, consisting of only those links $i \in \mathcal{S}$ that support constraints, meaning that the cardinality of $ES(i)$ is greater than 0, denoted by $\mathcal{C}(ES(i)) > 0$. $\mathcal{S}_{\mathcal{E}r}$ is a reversed version of $\mathcal{S}_\mathcal{E}$.

We first present the three-sweep algorithm, which is computed only during the first ALM iteration. For subsequent

ALM iterations, we present a fast reduced two-sweep algorithm, that re-uses many of the terms computed in the three-sweep algorithm. Finally, we present the constraintABA algorithm itself and perform a complexity analysis. Please note that there are some additional computation optimizations to avoid unnecessary computations. Performing a second forward sweep to compute accelerations for links not supporting constraints ($\mathcal{S}-\mathcal{S}_\mathcal{E}$) is not useful during ALM iterations since they do not have any constraints that would require the ALM's Lagrange multiplier updates. For the same reason, no backward sweep is required for these links from the second ALM iteration onwards, as there are no updated constraint forces that need to be transmitted back. Therefore, to avoid unnecessary computation, the reduced sweeps are not performed for these links until the ALM iterations have converged.

A. Three-sweep constrainedABA algorithm

Algorithm 1 ConstrainedABA three-sweep algorithm

Require: $\mathbf{q}^\mathrm{p}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, S_is K_is, $\mathbf{k}_i$s, μ, $\boldsymbol{\lambda}^k$, robot model

First forward sweep

1: **for** i in $\mathcal{S}$ **do**
2: $\quad \mathbf{v}_i = \mathbf{v}_{\pi(i)} + S_i\boldsymbol{\nu}_i$
3: $\quad \mathbf{a}_{b,i} = \mathbf{v}_i \times S_i\boldsymbol{\nu}_i$
4: $\quad \mathbf{f}_i^A \leftarrow \mathbf{f}_i + T_i\boldsymbol{\tau}_i - \mathbf{v}_i \times^* H_i\mathbf{v}_i + \mathbf{f}_i^{\text{ext}}$; $\mathbf{f}_{\pi(i)}^A \leftarrow \mathbf{f}_{\pi(i)}^A - T_i\boldsymbol{\tau}_i$
5: $\quad H_i^A \leftarrow H_i + \mu K_i^T K_i$; $\quad \mathbf{f}_i^A \leftarrow \mathbf{f}_i^A + K_i^T(\mu k_i - \boldsymbol{\lambda}_i^k)$

Backward sweep

6: **for** i in $\mathcal{S}_r$ **do**
7: $\quad D_i = S_i^T H_i^A S_i$; $P_i = (I_{6\times 6} - H_i^A S_i (D_i)^{-1} S_i^T)$
8: $\quad \mathbf{f}_{\pi(i)}^A \leftarrow \mathbf{f}_{\pi(i)}^A + P_i(\mathbf{f}_i^A - H_i^A \mathbf{a}_{b,i})$
9: $\quad H_{\pi(i)}^A \leftarrow H_{\pi(i)}^A + P_i H_i^A$

Second forward sweep (roll-out)

10: **for** i in $\mathcal{S}_\mathcal{E}$ **do**
11: $\quad \dot{\boldsymbol{\nu}}_i^{(k+1)*} = (D_i)^{-1} S_i^T \left\{ \mathbf{f}_i^A - H_i^A(\mathbf{a}_{\pi(i)}^{k+1} + \mathbf{a}_{b,i}) \right\}$
12: $\quad \mathbf{a}_i^{k+1} = \mathbf{a}_{\pi(i)}^{k+1} + S_i\dot{\boldsymbol{\nu}}_i^{(k+1)*} + \mathbf{a}_{b,i}$

Algorithm 1 lists the three sweep part of constrainedABA. It is identical to the original articulated-body algorithm [1], [10] except for the line 5, where the inertia matrices and the spatial force terms are updated to account for the influence of the constraint. It requires modifying existing ABA implementations by adding only a few lines of code to obtain the presented algorithm, thereby facilitating its implementation in existing simulators and rigid-body dynamics libraries such as RBDL [49], Pinocchio [6] or Drake [50].

B. Reduced two-sweep algorithm

For the subsequent ALM iterations, we now derive a fast reduced two-sweep algorithm that computes fast sweeps by re-using as many terms computed in Algorithm 1 as possible. The main observation is that only the force update in Eq. (29) changes between the ALM iterations, and we compute only the modification to the force and acceleration terms due to this force update in the two-sweep algorithm. This force term also changes only for those links that support constraints $\mathcal{S}_\mathcal{E}$.

Even in Eq. (29), only the $\boldsymbol{\lambda}_i$ term changes between ALM iterations. These dual variables are updated using Eq. (27b). Substituting Eq. (27b) in Eq. (29), we see that the spatial force terms change by $\mu K_i^T \left(\mathbf{k_i} - K_i\mathbf{a}_i^{k+1}\right)$. We introduce an additional term $\Delta\mathbf{f}_i$ to keep track of these force updates. The reduced sweep algorithm is presented in Algorithm 2.

Algorithm 2 ConstrainedABA reduced two-sweep algorithm

Require: $\Delta\mathbf{f}_i$s, μ, K_is, $\mathbf{k}_i$s, $\mathbf{a}_i$s, P_is, D_is, S_is, $\mathbf{f}_i^A$s, H_i^As, robot model

Backward sweep

1: **for** i in $\mathcal{S}_{\mathcal{E}r}$ **do**
2: $\quad \Delta\mathbf{f}_i \leftarrow \Delta\mathbf{f}_i + \mu K_i^T(\mathbf{k}_i - K_i\mathbf{a}_i^k)$
3: $\quad \Delta\mathbf{f}_{\pi(i)} \leftarrow \Delta\mathbf{f}_{\pi(i)} + P_i\Delta\mathbf{f}_i$

Second forward sweep (roll-out)

4: **for** i in $\mathcal{S}_\mathcal{E}$ **do**
5: $\quad \dot{\boldsymbol{\nu}}_i^{(k+1)*} = D_i^{-1} S_i^T \left\{ \mathbf{f}_i^A + \Delta\mathbf{f}_i - H_i^A \left(\mathbf{a}_{\pi(i)}^{k+1} + \mathbf{a}_{b,i} \right) \right\}$
6: $\quad \mathbf{a}_i^{k+1} = \mathbf{a}_{\pi(i)}^{k+1} + S_i\dot{\boldsymbol{\nu}}_i^{(k+1)*} + \mathbf{a}_{b,i}$
7: $\quad \Delta\mathbf{f}_i = \mathbf{0}_6$

C. ConstrainedABA algorithm

Using the three-sweep and reduced two-sweep algorithms, we present the entire constrainedABA in Algorithm 3. The algorithm takes as input the numerical tolerance $\epsilon \in \mathbb{R}, \epsilon > 0$, and the maximum number of iterations to determine the convergence of the ALM method.

Algorithm 3 ConstrainedABA

Require: $\mathbf{q}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, μ, max_iter, K_is, $\mathbf{k}_i$s, ϵ, robot model

1: $\boldsymbol{\lambda}^0 \leftarrow \mathbf{0}_m$
2: Execute the three-sweep algorithm in Algorithm 1.
3: **for** i in in $\mathcal{S}$ **do**
4: $\quad \Delta\mathbf{f}_i \leftarrow \mathbf{0}_6$
5: **for** k in range(1, max_iter) **do**
6: $\quad$ **if** $\min(\|\dot{\boldsymbol{\nu}}^k - \dot{\boldsymbol{\nu}}^{k-1}\|_\infty, \|K\mathbf{a}^k - \mathbf{k}\|_\infty) < \epsilon$ **then**
7: $\quad\quad$ break
8: $\quad$ Execute reduced-two sweep algorithm in Algorithm 2
9: **for** i in $\mathcal{S} - \mathcal{S}_\mathcal{E}$ **do**
10: $\quad \dot{\boldsymbol{\nu}}_i^{(k+1)*} = (D_i)^{-1} S_i^T \left\{ \mathbf{f}_i^A - H_i^A(\mathbf{a}_{\pi(i)}^{k+1} + \mathbf{a}_{b,i}) \right\}$
11: $\quad \mathbf{a}_i^{k+1} = \mathbf{a}_{\pi(i)}^{k+1} + S_i\dot{\boldsymbol{\nu}}_i^{(k+1)*} + \mathbf{a}_{b,i}$

D. Complexity analysis

We now analyze the computational complexity of Algorithm 1, where each line has a fixed computational cost except for the line 5, which is proportional to number of rows in K_i, corresponding to the number of constraints acting on the i^{th} link m_i. The line 5 is executed at all constrained links. Therefore, this line requires $O(m)$ number of operations in total. All the other lines are executed at most n number of times (once per joint) and therefore require $O(n)$ number of operations. Combining these costs, we get $O(m+n)$ as the computational complexity for the three-sweep algorithm.

Very similar analysis can be made for the reduced sweep Algorithm 2, where the line 2 requires $O(m)$ operations while the rest of the lines have a fixed cost and require $O(n)$ operations, bringing the total computational complexity of the two-sweep algorithm to $O(m+n)$ as well.

Within constrainedABA (see Algorithm 3), the three-sweep algorithm is computed once, and the two-sweep algorithm is computed at most max_iter-1 number of times, which is a fixed number supplied by the user. Empirically, we find that constrainedABA requires only a few iterations (nearly always under 5), as we shall see in the benchmarking (see Sec. VIII) on a diverse set of mechanisms. In practice, this brings the total computational complexity of constrainedABA to $O(m+n)$.

V. Proximal PV algorithm

As mentioned in Section II-G, the PV algorithm in literature [13] requires linear independence of the motion constraints and in practice employs Tikhonov regularization to handle the singular cases. However, this prevents the motion constraints from being exactly satisfied even when feasible. This drawback of the PV algorithm can be straightforwardly resolved by employing the PPA (see Section II-D1) to obtain a proximal algorithm version of the PV algorithm, that we shall call proxPV. We only discuss the modifications to the original PV algorithm that will provide us with the proxPV algorithm. For a detailed derivation of the PV algorithm, the readers are referred to [13].

The PV algorithm computes the dual function $\mathbf{g}(\boldsymbol{\lambda})$ (both Λ^{-1} and the affine terms) at the end of its backward sweep. Similarly to proxLTL, the weighted regularization $\frac{1}{\mu}I$ is added to Λ^{-1} to get Λ_μ^{-1}, which is always invertible and can be efficiently factorized using the Cholesky factorization. Then the dual function is maximized using the PPA until convergence and the resulting robot accelerations are computed in the third sweep. Effectively, only three lines are changed in the PV algorithm to get the proxPV algorithm.

Though we do not include a full derivation, Algorithm 4 presents the entire proxPV algorithm to make this paper self-contained. Every term in Algorithm 4 also has a physical interpretation, and we refer readers to [13] for a detailed discussion. Algorithm 4 requires additional notations that we now define as follows: for $\forall j \in \gamma(i)$, let

$$\boldsymbol{\lambda}_i^A = \begin{bmatrix}\dots, & \boldsymbol{\lambda}_j^{AT}, & \dots\end{bmatrix}^T, K_i^A = \begin{bmatrix}\vdots \\ K_j^A \\ \vdots\end{bmatrix},$$

and:

$$L_i^A = \begin{bmatrix}\ddots & & \\ & L_j^A & \\ & & \ddots\end{bmatrix}.$$

These three quantities collect all the terms related to motion constraints on the subtree rooted at the i^{th} link to make the recursive formulae in the backward sweep work.

Algorithm 4 Proximal PV solver

Require: $\mathbf{q}$, $\boldsymbol{\nu}$, $\boldsymbol{\tau}$, K_is, $\mathbf{k}_i$s, ϵ, max_iter, $\boldsymbol{\lambda}_0^0$, robot model

First forward sweep

1: **for** i in $\mathcal{S}$ **do**
2: $\quad\mathbf{v}_i = \mathbf{v}_{\pi(i)} + S_i\boldsymbol{\nu}_i$
3: $\quad\mathbf{a}_{b,i} = \mathbf{v}_i \times S_i\boldsymbol{\nu}_i$
4: $\quad\mathbf{f}_i^A \leftarrow \mathbf{f}_i^A + T_i\boldsymbol{\tau}_i - \mathbf{v}_i \times^* H_i\mathbf{v}_i + \mathbf{f}_i^{\text{ext}};\ K_i^A \leftarrow K_i;\ \mathbf{l}_i \leftarrow -\mathbf{k}_i;\ L_i^A \leftarrow \mathbf{0}_{m_i\times m_i}\ H_i^A \leftarrow H_i;$
$\quad\mathbf{f}_{\pi(i)}^A \leftarrow \mathbf{f}_{\pi(i)}^A - T_i\boldsymbol{\tau}_i$

Backward sweep

5: **for** i in $\mathcal{S}_r$ **do**
6: $\quad D_i = S_i^T H_i^A S_i;\ P_i = (I_{6\times6} - H_i^A S_i (D_i)^{-1} S_i^T)$
7: $\quad\mathbf{f}_{\pi(i)}^A \leftarrow \mathbf{f}_{\pi(i)}^A + P_i(\mathbf{f}_i^A - H_i^A\mathbf{a}_{b,i})$
8: $\quad H_{\pi(i)}^A \leftarrow H_{\pi(i)}^A + P_i H_i^A$
9: $\quad K_{\pi(i)}^A \leftarrow \begin{bmatrix}K_{\pi(i)}^A \\ K_i^A P_i^T\end{bmatrix}$
10: $\quad L_{\pi(i)}^A \leftarrow \begin{bmatrix}L_{\pi(i)}^A & \\ & L_i^A + K_i^A S_i (D_i)^{-1} S_i^T K_i^{AT}\end{bmatrix}$
11: $\quad\mathbf{l}_{\pi(i)} \leftarrow \begin{bmatrix}\mathbf{l}_{\pi(i)} \\ \mathbf{l}_i + K_i^A\{\mathbf{a}_{b,i} + S_i D_i^{-1} S_i^T(\mathbf{f}_i^A - H_i^A\mathbf{a}_{b,i})\}\end{bmatrix}$
12: **for** k in range(1, max_iter) **do**
13: $\quad\boldsymbol{\lambda}_0^k = (L_0^A + \frac{1}{\mu}I)^{-1}(\frac{\boldsymbol{\lambda}_0^{k-1}}{\mu} + K_0\mathbf{a}_0 + \mathbf{l}_0)$
14: $\quad$**if** $\left\| L_0^{AT}\left(L_0^A\boldsymbol{\lambda}_0^k - K_0\mathbf{a}_0 - \mathbf{l}_0\right)\right\|_\infty < \epsilon$ **then**
15: $\quad\quad$break
16: $\boldsymbol{\lambda}_0^{A*} = \boldsymbol{\lambda}_0^k$

Second forward sweep (roll-out)

17: **for** i in $\mathcal{S}$ **do**
18: $\quad\dot{\boldsymbol{\nu}}_i^* = D_i^{-1} S_i^T\left\{\mathbf{f}_i^A - H_i^A(\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i}) - K_i^{AT}\boldsymbol{\lambda}_i^{A*}\right\}$
19: $\quad\mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i\dot{\boldsymbol{\nu}}_i^* + \mathbf{a}_{b,i}$

VI. Efficient algorithms to compute Λ_μ

In this section, we introduce a new algorithm, named cABA-OSIM, to compute the damped Delassus matrix inverse Λ_μ with the lowest computational complexity of $O(n+m^2)$ compared to the next best algorithm of $O(n+m^3)$ complexity from literature [28]. Λ_μ^{-1} is the Hessian of the dual function of the proximal formulation of constrained dynamics (see Eq. (9)), and is an important quantity used in constrained dynamics simulation [13], [20] and operational space control [24].

The cABA-OSIM algorithm exploits a linear algebra identity known as the matrix inversion lemma [29] (also called Morrison-Sherman-Woodbury formula), which states that

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}, \tag{39}$$

where $A \in \mathbb{R}^{n\times n}$, $U \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{m\times n}$, and A and C matrices are assumed to be invertible. The identity is useful for computing the inverse of the left-hand-side in Eq. (39), when it is easy to compute the inverses of A and C.

We recall that

$$\Lambda_\mu^{-1} := \frac{1}{\mu}I_{m\times m} + J_{\mathbf{f}_c}M^{-1}J_{\mathbf{f}_c}^T,$$

to which applying the matrix inversion lemma gives,

$$\Lambda_\mu = \mu I_{m\times m} - \mu J_{\mathbf{f}_c}(M_\mu)^{-1} J_{\mathbf{f}_c}^T \mu, \tag{40}$$

where we recall that

$$M_\mu := M + \mu J_{\mathbf{f}_c}^T J_{\mathbf{f}_c}.$$

The second term in the right-hand side of Eq. (40) is algebraically identical to the Delassus matrix expression, which is of the form $\Lambda^{-1} := J_{\mathbf{f}_c}(M)^{-1} J_{\mathbf{f}_c}^T$. Therefore, we can augment the inertia matrix and leverage existing efficient algorithms for computing the Delassus matrix [19], [26], [28] followed by scalar multiplications and subtraction from a diagonal matrix to compute the damped Delassus matrix inverse. We recall from Section II-F that the augmented inertia matrix in the maximal coordinates takes the form of updating all the constrained links' inertias $H_i \leftarrow H_i + \mu K_i^T K_i$. This implies that one of the two efficient maximal coordinates-based Delassus matrix algorithms, namely EFPA [26] or PV-OSIMr [28] can be used within cABA-OSIM.

Between the two options, we select the PV-OSIMr algorithm due to its lower computational complexity of $O(n + m^2)$ compared to $O(m + md + m^2)$ of EFPA. Since the inertia augmentation and the evaluation of Eq. (40) require at most $O(m^2)$ number of additional operations, the total complexity of cABA-OSIM is $O(n + m^2)$. This contrasts with all the existing (that we are aware of) Λ_μ computation algorithms that require an additional $O(m^3)$ number of operations to factorize the Delassus matrix. We present the cABA-OSIM algorithm in the coming subsection, where we also include the computations within PV-OSIMr for completeness.

A. cABA Delassus inverse algorithm

Algorithm 5 presents the cABA-OSIM algorithm. We now review the terms from [28] needed to parse this algorithm and follow that with a brief explanation. To make the notation easier later, let us introduce a list of dummy links, $\mathcal{E}' = \{n_b+1, n_b+2, \ldots, n_b+m_b\}$, as a child link to each link that is constrained and call these dummy links, end-effector links. These dummy links are assumed to be connected to their parents with a fixed joint and their frame corresponding to the frame of their parent link. For every $j \in \mathcal{E}'$,, there exists an $i \in \mathcal{E}$, such that $\pi(j) = i$. Let $\mathrm{ES}(i)$ be the set of end-effector indices supported by the i^{th} link, meaning that it consists of all descendant links of i that are in $\mathcal{E}'$. Let $\mathcal{N}$ be the set of all the indices of the branching links, that support more strictly more constraints than all their children links. $\mathcal{N} = \{i \in \mathcal{S} | \mathrm{ES}(j) \subset \mathrm{ES}(i), \forall j \in \gamma(i)\}$. If i is a leaf node, note that $\gamma(i) = \varnothing$ and we define $\mathrm{ES}(\varnothing) = \varnothing$.

Let ${}^jP_i$ denote the extended force propagator (EFP), first discussed in [51], which directly propagates spatial forces acting on the i^{th} link backward in the kinematic tree to an ancestor link j. ${}^iP_i = I_6$. EFP's transpose ${}^jP_i^T$ directly propagates the spatial accelerations of the j^{th} link to the descendant link i (see [13], [51] for a derivation). Let ${}^j\Omega_i$ be the spatial inverse inertia matrix, which maps spatial forces acting on the i^{th} link to the i^{th} link's spatial acceleration considering the motion of all the joints connecting the j^{th} and the i^{th} links. Let $\mathrm{cca}(i,j) = \mathbf{max}\{k \in \mathcal{S} | k \in \mathrm{anc(i)} \cap \mathrm{anc(i)}\}$ be the closest common ancestor link of the i^{th} and the j^{th} links, which could possibly be the i^{th} or the j^{th} links themselves.

The only additions compared to the PV-OSIMr algorithm [28] are in line 4, where the inertia and constraint forces are updated and in line 21, where the Delassus matrix is computed using the matrix inversion lemma. The PV-OSIMr achieves its low complexity by computing only the required Ω and EFP terms compared to other algorithms like PV-OSIM [13] or EFPA [26].

Line 9 recursively computes the extended force propagator (EFP) from a link $i \in \mathcal{N}$ to its closest proper ancestor in $\mathcal{N}$, denoted as $\mathcal{N}_{\mathrm{anc}}(i) = \mathbf{max}\{j \in \mathcal{N}, j \neq i, j \in \mathrm{anc}(i)\}$. Similarly, line 10 recursively computes the spatial inverse inertia apparent at a link $i \in \mathcal{N}$ considering the joints connecting the i^{th} link and $\mathcal{N}_{\mathrm{anc}}(i)^{\text{th}}$ link. Then, in a limited outward sweep over $i \in \mathcal{N}$, the spatial inverse inertia considering the motion of all ancestor joints ${}^0\Omega_i$ is computed in line 13. Finally, the constraint space EFP ${}^jK_i^T$, which maps $\boldsymbol{\lambda}_i$ to the spatial forces acting on the j^{th} link is computed for all end-effector links $i \in \mathcal{E}'$ and for all links $j \in \mathcal{N} \cap \mathrm{anc}(i)$ in line 16. Note that ${}^iK_i = K_i$, and ${}^{\pi(i)}P_i = I_6$, $\forall i \in \mathcal{E}'$.

This gives all the quantities required for assembling the Delassus matrix for M_μ, given by

$$L_0^A = \begin{bmatrix} {}^0L_{1,1}^A & \cdots & {}^0L_{1,m_b}^A \\ \vdots & \ddots & \vdots \\ {}^0L_{m_b,1}^A & \cdots & {}^0L_{m_b,m_b}^A \end{bmatrix}$$

whose blocks are computed in line 18, for $i, j \in \mathcal{E}'$, where $m_b = \mathcal{C}(\mathcal{E}')$. That L_0^A is the Delassus matrix, was shown in [13]. Each block ${}^0L_{i,j}^A$ encodes the coupling between the i^{th} and i^{th} links. The right-hand side term in line 18 can be interpreted as follows, it first maps $\boldsymbol{\lambda}_j$ to spatial forces on the $\mathrm{cca}(i,j)^{\text{th}}$ link, then computes the spatial acceleration of the $\mathrm{cca}(i,j)^{\text{th}}$ link resulting from these forces which actually causes the the constraints on the i^{th} and j^{th} link to get coupled, before finally transmitting this acceleration to the constraint accelerations of the i^{th} link. Finally, the damped Delassus inverse matrix, the output of the cABA-OSIM algorithm, is computed in line 21 using the matrix inversion lemma.

B. Accelerating cABA for floating-base robots (cABA-OSIMf)

We now propose a minor structure-exploiting acceleration for cABA-OSIM for certain robot structures, namely floating-base robots with branching at the base. This structure is found commonly, e.g., in quadrupeds and humanoids. For these mechanisms, the Delassus matrix (as well as its inverse) is dense, without any structural zeros because all the constraints acting on different branches get coupled due to the floating-base's motion. However, L_1^A, where 1 is the floating base's index, would have a block-diagonal structure, with each block corresponding to constraints from a branch. It is computationally more efficient to exploit this block-diagonal sparsity and not explicitly compute the dense L_0^A matrix. This idea is implemented by the cABA-OSIM-fast (cABA-OSIMf) that we introduce in the rest of this subsection.

Algorithm 5 The cABA-OSIM algorithm

Require: $\mathbf{q}$, K_is, robot model, μ

First forward sweep

1: **for** i in $\mathcal{S}$ **do**
2: $H_i^A \leftarrow H_i$;
3: **if** $i \in \mathcal{E}$ **then**
4: $H_i^A \leftarrow H_i^A + \mu K_i^T K_i$; $K_i^A \leftarrow \mu K_i$

Backward sweep

5: **for** i in $\mathcal{S}_r$ **do**
6: **if** $i \in \mathcal{N}$ **then**
7: $d \leftarrow i$; ${}^{i}P_i = I_{6\times 6}$; ${}^{i}\Omega_i^A \leftarrow 0_{6\times 6}$
8: $D_i = S_i^T H_i^A S_i$; $P_i = (I_{6\times 6} - H_i^A S_i (D_i)^{-1} S_i^T)$
9: $H_{\pi(i)}^A \leftarrow H_{\pi(i)}^A + P_i H_i^A$; ${}^{\pi(i)}P_{d(i)} = P_i \ {}^{i}P_{d(i)}$
10: ${}^{\pi(i)}\Omega_{d(i)} = {}^{i}\Omega_{d(i)} + {}^{i}P_{d(i)}^T \ S_i (D_i^{-1}) S_i^T \ {}^{i}P_{d(i)}$
11: $\mathrm{d}(\pi(i)) \leftarrow d(i)$
12: **for** i in $\mathcal{N}$ **do**
13: ${}^{0}\Omega_i = {}^{\mathcal{N}\mathrm{anc}(i)}\Omega_i + {}^{\mathcal{N}\mathrm{anc}(i)}P_i^T \ {}^{0}\Omega_{\mathcal{N}\mathrm{anc}(i)} \ {}^{\mathcal{N}\mathrm{anc}(i)}P_i$
14: **for** $e_i \in \mathcal{E}$ **do**
15: **for** i in $\mathcal{N}_r \cap \mathrm{anc}(e_i)$ **do**
16: ${}^{\mathcal{N}\mathrm{anc}(i)}K_{e_i} = {}^{i}K_{e_i} \ {}^{\mathcal{N}\mathrm{anc}(i)}P_i^T$
17: **for** $i, j \in \mathcal{E}$ if $i \leq j$ **do**
18: ${}^{0}L_{i,j}^A = {}^{\mathrm{cca}(i,j)}K_i^A \ ({}^{0}\Omega_{\mathrm{cca}(i,j)}) \ {}^{\mathrm{cca}(i,j)}K_j^{AT}$
19: **if** $i \neq j$ **then**
20: ${}^{0}L_{j,i}^A = {}^{0}L_{i,j}^{AT}$
21: $\Lambda_\mu = \mu I_{m\times m} - L_0^A$

Using the notation from the PV algorithm, L_1^A is related to L_0^A by the equation

$$L_0^A = L_1^A + K_1^A (H_1^A)^{-1} K_1^{AT}. \tag{41}$$

, where accounting for the coupling introduced between the constraints on different branches due to the spatial inverse inertia of the floating base link destroys block-diagonal sparsity pattern of L_1^A. The key idea behind cABA-OSIMf is to not explicitly evaluate the right-hand side of the equation above but to directly multiply this right-hand side with vectors. The cABA-OSIM shown in Algorithm 5 can be easily modified to compute L_1^A instead of L_0^A. This modification involves computing ${}^{1}\Omega_i$ instead of ${}^{0}\Omega_i$ for all links $i \in \mathcal{N}$ in line 13 and similarly ${}^{1}L_{i,j}^A$ instead of ${}^{0}L_{i,j}^A$ in line 18.

Let us consider that the dimensionality of constraints on the links of any descendant branch of the floating base is constant. We have that m is proportional to the number of branches. Then computing each term of the right-hand-side of Eq. (41) has a total computational complexity of $O(n)$ because the computation of each ${}^{1}L_{i,j}$ for constraints has a computational complexity of the number of joints in that branch (because we are considering that constraint dimensionality per branch is a constant). Combining this cost for all branches yields the $O(n)$ complexity for computing the right-hand size of Eq. (41). Multiplying the right-hand-side of Eq. (41) with a vector clearly has a total computational complexity of only $O(m)$.

VII. Connections to MuJoCo and the PV algorithms

In this section, we discuss the connections between the algorithms presented in this paper and the ones implemented in MuJoCo [52] corresponding to PV-soft and the PV-early algorithms developed in [13].

A. Connection to MuJoCo and PV-soft

MuJoCo [5], [52], a popular simulator in robotics and RL communities, with a wide user base, solves a soft version of the Gauss' principle. In MuJoCo, the hard motion constraints are relaxed with a quadratic penalty. For a simulation problem with equality constraints, the optimization problem solved by MuJoCo is of the form

$$\begin{aligned}\underset{\dot{\boldsymbol{\nu}}}{\textbf{minimize}} \ & \frac{1}{2}\|\dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_{\mathrm{free}}\|^2_{M(\mathbf{q})} + \\ & \|J_{\mathbf{f}_c}(q)\dot{\boldsymbol{\nu}} + \boldsymbol{\gamma}_{\mathbf{f}_c} - \mathbf{a}_c^*\|^2_{R_E^{-1}},\end{aligned} \tag{42}$$

where $R_E \in \mathcal{S}^m_{++}$ is a diagonal matrix chosen by the user, and which can be interpreted as a compliance term.

Comparing the optimization problem in Eq. (42) with the proxLTLs formulation in Eq. (15), we see that for $R_E^{-1} = \mu I_{m\times m}$ and $\boldsymbol{\lambda}^k = \mathbf{0}_m$, one iteration of proxLTLs is identical to the soft-Gauss problem solved in MuJoCo. Please note from Remark 2 that proxLTLs is trivially extendable to support a positive definite diagonal matrix R_E^{-1} instead of the weighted identity matrix $\mu I_{m\times m}$ we have chosen as the compliance term. Therefore, MuJoCo's algorithm can be considered to be a special case of proxLTLs. A practical consequence of this insight is that codebases using soft Gauss principle such as MuJoCo can be extended with minor changes to obtain the proxLTLs algorithm that can solve for the motion constraints exactly. Computationally, even our first iteration of the proxLTLs differs from MuJoCo's implementation by using the trick mentioned in Section II-F to speed up M_μ computation.

Analogously in the maximal coordinates, the problem solved by the PV-soft algorithm is minimizing

$$\underset{\dot{\boldsymbol{\nu}}}{\textbf{minimize}} \ \sum_{i=1}^{n_b} \left\{ \frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a}_\mathbf{i} + \frac{\mu}{2}\|K_i\mathbf{a}_i - \mathbf{k}_i\|^2 \right\},$$

which can be seen to be a special case of constraintABA's objective function (see Eq. (26)), when $\boldsymbol{\lambda}_k = \mathbf{0}_{m_k}$. So, PV-soft can be interpreted as the first ALM iteration of the constraintABA.

B. The proximal PV-early solver

ConstrainedABA is closely related to the $O(n+m)$ complexity PV-early algorithm proposed in [13]. ConstrainedABA can be derived by applying the PV-early algorithm to solve the proximal function in Eq. (13), but this time in maximal coordinates. This involves adding proximal ℓ-2 regularization to the maximal coordinate Lagrangian in Eq. (25) to get

$$\begin{aligned}\mathrm{prox}_{\mu,\mathcal{L}_m}(\dot{\boldsymbol{\nu}},\boldsymbol{\lambda}^k) := \underset{\boldsymbol{\lambda}}{\textbf{arg}} \ \underset{\dot{\boldsymbol{\nu}}}{\textbf{min}} \ \underset{\boldsymbol{\lambda}}{\textbf{max}} \ & \sum_{i=1}^{n_b} \left\{ \frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \right. \\ & \left. \mathbf{f}_i^T \mathbf{a}_\mathbf{i} + \boldsymbol{\lambda}_i^T (K_i\mathbf{a}_i - \mathbf{k}_i) - \frac{1}{2\mu}\|\boldsymbol{\lambda}_i - \boldsymbol{\lambda}_i^k\|^2 \right\},\end{aligned} \tag{43}$$

which makes the dual problem strongly convex. In the PV-early algorithm, the dual variables are aggressively eliminated as early as possible. Applying this idea to the problem above in Eq. (43), all the dual variables can be immediately eliminated since the problem is now strongly convex in the dual variables. Imposing gradient stationarity KKT condition w.r.t the dual variables of (43) gives

$$-\frac{1}{\mu}\boldsymbol{\lambda}_i^* + \frac{1}{\mu}\boldsymbol{\lambda}_i^k + (K_i\mathbf{a}_i - \mathbf{k}_i) = 0. \tag{44}$$

Re-arranging the terms gives the expression for optimal $\boldsymbol{\lambda}_i$

$$\boldsymbol{\lambda}_i^* = \boldsymbol{\lambda}_i^k + \mu(K_i\mathbf{a}_i - \mathbf{k}_i). \tag{45}$$

Finally, substituting them back into the problem in Eq. (43) gives the purely primal objective function

$$\min_{\dot{\boldsymbol{\nu}}} \sum_{i=1}^{n_b} \Big\{ \frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a_i} + \frac{\mu}{2}\|K_i\mathbf{a}_i - \mathbf{k}_i\|^2 + \boldsymbol{\lambda}_i^{kT}(K_i\mathbf{a}_i - \mathbf{k}_i) \Big\}.$$

The objective function above is identical to the maximal coordinate ALM formulation (see Eq. (25)) used to derive the constrainedABA algorithm. Further computations by the PV-early solver would mirror the three-sweep algorithm of constrainedABA. Therefore, evaluating the proximal operator using the PV-early solver is an alternative derivation of the constrainedABA. Compared to this approach, our ALM-based derivation in Section III is more accessible.

Previous work [13] had presented PV-soft and PV-early as two separate algorithms. However, from a proximal perspective, we have shown that PV-soft is identical to one iteration of constraintABA and that constraintABA can be derived by applying the PV-early algorithm on the proximal problem in Eq. (43). This reveals an interesting connection between the PV-soft and PV-early algorithms, namely that PV-soft can be derived from the PV-early algorithm using the proximal formulation.

VIII. Computational Benchmarking

We now present the benchmarking results of this paper's algorithms on diverse robots, namely Kuka Iiwa (7 DoF chain), Solo (18 DoF tree) [53], Talos (50 DoF tree) [54], and Atlas with two shadow-hands attached to each wrist (84 DoF tree). We also study the computational scaling on a chain mechanism and a balanced binary tree mechanism of varying numbers of links. This studies the impact of different robot topologies on the presented algorithms.

Constraints on a hand, fingertip, or feet in the benchmarks are represented as H_{m_i}, T_{m_i} or F_{m_i} respectively where m_i is the constraint dimension. $m_i = 3$ for `connect`-type constraints where the constrained frame's translation is restricted, but rotation is. This can model a quadruped's foot contact constraints. $m_i = 6$ for `weld`-type constraints, where the constrained link's motion is restricted in all six directions. Notations such as $F^2_{m_i}$ would indicate that F_{m_i} constraint is imposed on each of the two robot feet. T^i_3 would imply that `connect`-type constraints were imposed on i different fingertips of the Shadow /hand. We also consider the typical case of modeling a humanoid's foot contact with 4 `connect`-type constraint, which is denoted by F^4_3 (note that this symbol for a quadruped would instead imply a `connect`-type constraint on each of its four feet).

We first discuss our implementation of the algorithms in Section VIII-A, followed by benchmarking the constrained dynamics algorithms and the damped Delassus inverse algorithms in Section VIII-B and, Section VIII-C respectively.

A. Efficient C++ implementation

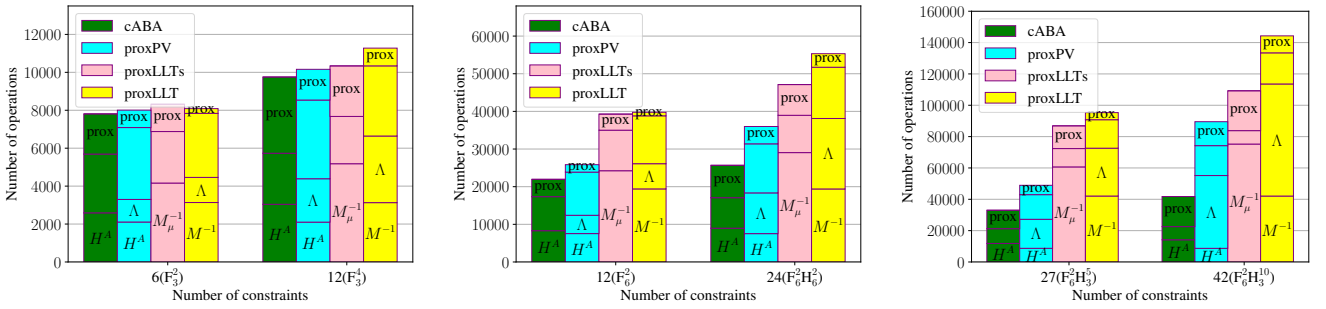
We first prototyped the algorithms in a `MATLAB` environment using CasADi [55] followed by a `C++` implementation within the Pinocchio library [6]. With CasADi's `SX` expressions, we can conveniently count the number of atomic operations such as additions, divisions, multiplications, sin/cos computations required by each algorithm. As explained in [13], this allows comparing the algorithms in the classical manner based on the number of operations, which is both implementation and computer architecture agnostic. However, on modern CPU architectures, vectorization, and memory access can significantly influence computation timings, therefore we also compare based on the computation timings of a `C++` implementation of the algorithms within the Pinocchio library. However, this timing comparison is admittedly influenced by code implementation details, compilers as well as on CPU architecture. The LTL implementation [20] in the Pinocchio library is particularly efficient due to its usage of CPU vectorization. Similarly, vectorizing constrainedABA and proxPV in a general manner is not straightforward and is deemed to be outside this paper's scope.

All timings were benchmarked on a `13th Gen Intel® Core™ i9-13950HX` laptop CPU running `Ubuntu 22.04LTS` operating system. We present both results with Intel's `Turbo Boost` both turned off to minimize CPU frequency variability and with `Turbo Boost` turned on to indicate the speeds achievable on a modern laptop in operational scenarios, when `Turbo Boost` is typically active. The code was compiled using `Clang-14` compiler with the usual optimized compilation flags `-O3 -march=native`.

B. Constrained dynamics

We now benchmark constrainedABA, proxPV, proxLTLs, and proxLTL algorithms in terms of the number of operations and the computation timings of their `C++` implementations.

1) Number of operations: Fig. 1 lists the number of operations, with their internal breakdown for proximal iterations, for the different algorithms on different robots. H^A refers to the cost of computing the articulated body inertia matrices in constrainedABA and proxPV algorithms. This operation is more expensive in constrainedABA due to the inertia matrix and constraint forces updates in Eq. (29). M_μ^{-1} and M^{-1} blocks for proxLTLs and proxLTL denote the cost of computing and factorizing M_μ and M blocks, respectively. Computing M_μ^{-1} is more expensive than M^{-1} due to the JSIM augmentation step in Eq. (24). The Λ block for proxPV



(a) Unitree Go1 robot. (b) Talos robot. (c) Atlas equipped with two Shadow Hands.

Fig. 1: Benchmarking the number of operations of the constrained dynamics algorithms with three proximal iterations.

and proxLTL denotes the cost of factorizing the Λ^{-1} matrix. Finally, the proximal block for all the algorithms denotes the number of operations required by three proximal iterations.

The proximal iterations for constrainedABA and proxLTLs are more expensive than for proxPV and proxLTL respectively, as the latter two algorithms only solve the smaller dual problem during proximal iterations. ConstrainedABA's proximal iterations are more expensive than that of proxLTLs because the former performs a reduced backward and forward sweep over all the joints supporting constraints, while proxLTLs computes this using the constraint Jacobian and Cholesky decomposition of M_μ. Proximal iterations of proxPV and proxLTL should cost the same, but proxPV's proximal iterations are slightly more expensive because of an implementation detail. We have employed an early elimination at the base trick [13], which speeds up factorization of Λ^{-1} (as seen in the reduced size of the Λ block for proxPV), but increases the cost of solving for the dual variables.

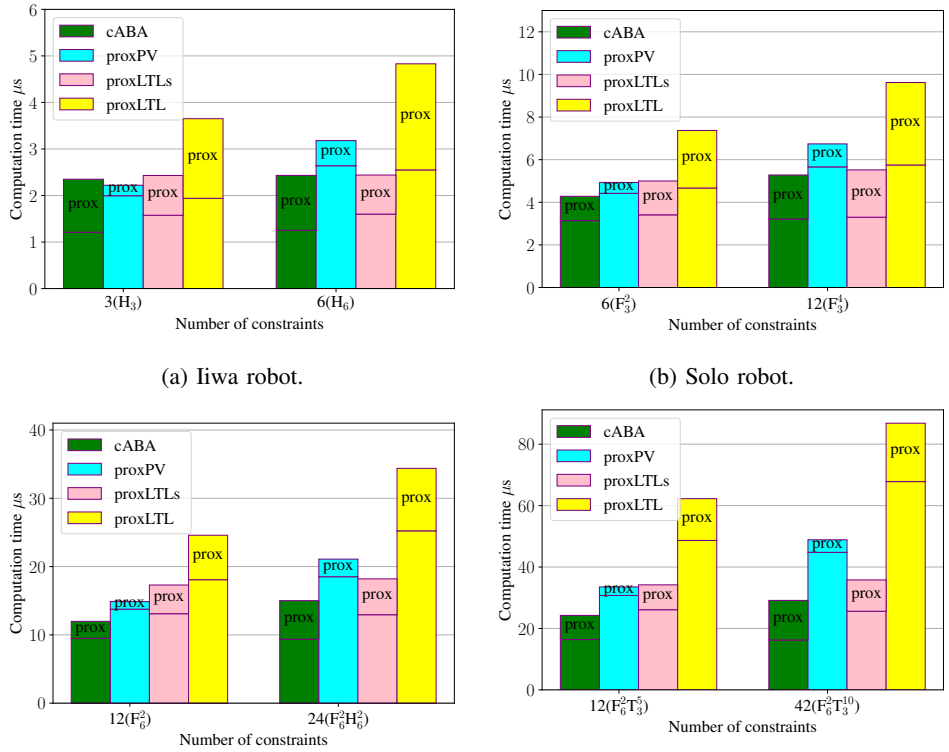
For the Go1 robot, the operation count of all the algorithms is similar, with constrainedABA requiring approximately 20% fewer operations than proxLTL. For larger robots as well as an increased number of constraints, the lower complexity algorithms clearly demonstrate their superiority.

2) Computation timings: Fig. 2 lists the computational timings benchmarking of the `C++` implementations of the algorithms with the `TurboBoost` disabled. We find constrainedABA to be faster than the other algorithms for all the considered robots and constraint scenarios, except for one case involving the Kuka iiwa robot arm with seven DoFs. Both constrainedABA and proxPV algorithms become relatively more efficient than the proxLTL algorithm for bigger robots due to their linear complexity w.r.t n. ConstrainedABA and proxLTLs were found to scale gracefully with an increasing number of constraints compared to the proxLTL or the proxPV algorithms due to their linear complexity w.r.t m. However, as expected from the previous section, the computational cost of each proximal iteration is greater for constrainedABA, which performs a reduced backward and forward sweep, compared to proxPV or proxLTL, which only solves the dual problem iteratively. Since `TurboBoost` is enabled in practice while simulating robotic systems, Table II includes the timings with `TurboBoost` enabled, for reference.

Computational scaling for chains. For an end-effector constrained chain (with 6D `weld`-type constraint), we now study the computational scaling of the algorithms w.r.t increasing chain sizes. The timings results are plotted for one proximal iteration and three proximal iterations in Fig. 3a and, Fig. 3b respectively. As expected, we observe a linear increase in computational cost for constrainedABA and proxPV and a superlinear increase for the proxLTL algorithm as the number of links increases. ConstrainedABA is over 4x faster than proxLTL for a chain of 100 links if we allow only one proximal iteration. However, when three proximal iterations are allowed, constrainedABA becomes nearly as expensive as the proxPV algorithm. As a testament to the efficient vectorized implementation of LTL in PINOCCHIO, the cost of proxLTL and proxLTLs increased only by 4x for 100 links compared to 50 links in Fig. 3a. Alternatively, if we used the dense Cholesky decomposition in the EIGEN library in place of PINOCCHIO's LTL, the cost for 100 link chain was found to be 8x higher than the cost for 50 link chain. This is expected due to the cubic complexity of LTL for chains.

Computational scaling for balanced binary trees. Finally, we benchmark the various algorithms on a balanced binary tree mechanism, a structure that most favours the LTL-based algorithms since the kinematic tree depth d is minimal for a balanced binary tree with n links if we permit a branching factor of 2. Each branch in the tree has three links before it splits into two descendant branches. We constrain all the leaf links with a 3D `connect`-type constraint. Therefore, a balanced binary tree with n links has $(n \bmod 6 + 1)3$ constraints and the results are plotted in Fig. 3c and Fig. 3d for one and three proximal iterations respectively. Therefore, we can also see the algorithms' scaling w.r.t. constraints in this test case. As expected, we observe constrainedABA to scale linearly. However, surprisingly, proxLTLs is also particularly efficient. This is because its complexity for the favorable binary tree example is $O(n\log_2^2(n) + md)$ linearly, and it benefits from the efficient vectorized implementation of LTL [20] in the PINOCCHIO library. Both proxPV and proxLTL scale poorly due to their cubic complexity w.r.t. to the number of constraints.

3) Convergence results: Empirically, the proximal method was found to converge quite reliably in about three iterations in



(a) Iiwa robot.

(b) Solo robot.

(c) Talos robot.

(d) Atlas robot with two Shadow Hands attached to each wrist.

Fig. 2: Computational benchmarking of the `C++` implementation of the constrained FD algorithms with `TurboBoost` disabled. The 'prox' component in the bar graphs corresponds to the computational cost of three proximal iterations for each algorithm.

TABLE II: Computational timings of the algorithms in μs with `Turbo Boost` enabled. Each algorithm is allowed three proximal iterations.

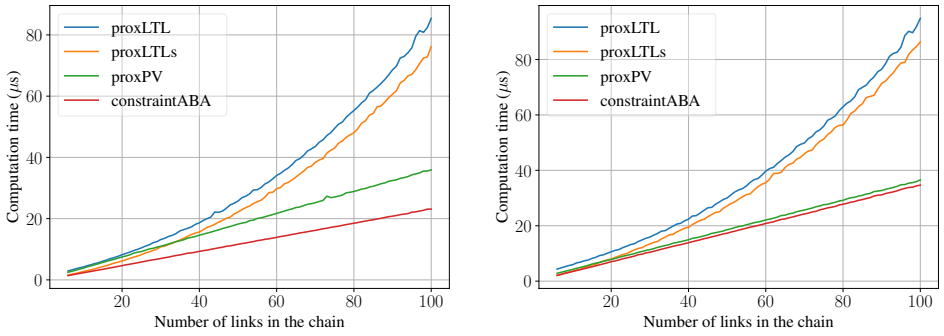
System	cABA	PV	LTLs	LTL
Iiwa - H$_3$	0.97	**0.95**	0.99	1.51
Iiwa - H$_6$	1.02	1.33	**1.0**	2.0
Solo - F$_3^2$	**1.81**	2.11	2.06	3.08
Solo - F$_3^4$	**2.25**	2.84	2.27	4.06
Talos - F$_6^2$	**5.02**	6.27	7.14	10.2
Talos - F$_6^2$H$_6^2$	**6.21**	8.84	7.57	14.5
Atlas SR - F$_6^2$	**7.90**	9.56	13.4	17.7
Atlas SR - F$_6^2$T$_3^5$	**9.95**	14.0	14.7	26.5
Atlas SR - F$_6^2$T$_3^{10}$	**11.7**	20.8	15.0	36.8

most cases. This should not be surprising since we are solving small equality-constrained strongly convex QPs. Fig. 4 plots a representative convergence whisker plot for 1000 randomly generated examples for a Talos robot with a challenging redundant constraint formulation. μ was chosen to be 10^6 as it provided a reasonable trade-off between fast convergence and numerical issues. Interestingly, we find the proxLTLs and constrainedABA to converge to smaller errors compared to proxLTL and proxPV algorithms. This is because, in the former algorithms, the primal error is fed back in the ALM formulation, effectively providing iterative refinement for free, while in the latter algorithms, the dual problem is solved till convergence before forward-sweep to get the primal variables.

C. Benchmarking cABA-OSIM

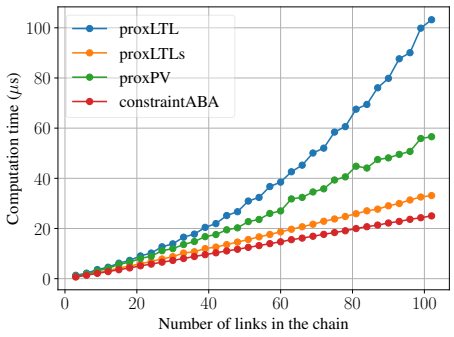
We now benchmark the damped Delassus inverse algorithms, cABA-OSIM and cABA-OSIMf (an accelerated version of cABA-OSIM tailored for floating-base robots with branching at the base) introduced in Section VI. We compare our cABA-OSIM and cABA-OSIMf with the existing state-of-the-art OSIM algorithms PV-OSIMr [28] and PV-OSIMf [13], along with the standard higher-complexity LTL-OSIM [19] algorithm. Similarly to the constrained dynamics algorithms, we first compare the algorithms by the number of operations before moving on to the computation timings of their `C++` implementations in the PINOCCHIO library.

Number of operations Fig. 5 lists the number of operations required by different OSIM algorithms for the Unitree Go1 robot, Talos robot and the Atlas robot with a Shadow Hand attached to each wrist in Fig. 5a, Fig. 5b, Fig. 5c respectively. In the plots, H^A refers to the cost of computing the articulated body inertias, which is slightly more expensive for the cABA algorithms compared to the PV algorithms due to additional inertia updates performed in line 4 of the Algorithm 5. The LTL-OSIM algorithm is in general the most expensive algorithm and scales poorly to higher dimensional robots as well. Both the PV-OSIMr and the LTL-OSIM algorithms scale poorly to the increasing number of constraints compared to cABA-OSIM, due to the cubic complexity they incur during

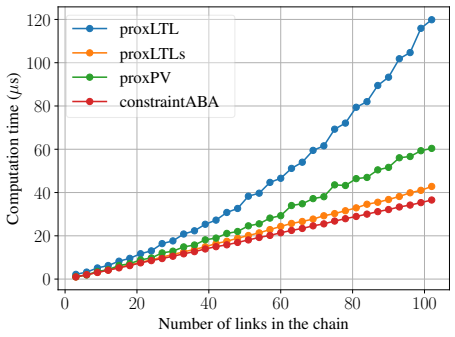


(a) One proximal iteration for serial chains.

(b) Three proximal iterations for serial chains.



(c) One proximal iteration for binary trees.



(d) Three proximal iterations for binary trees.

Fig. 3: Benchmarking the computational scaling of the different algorithms for serial chains and binary trees.

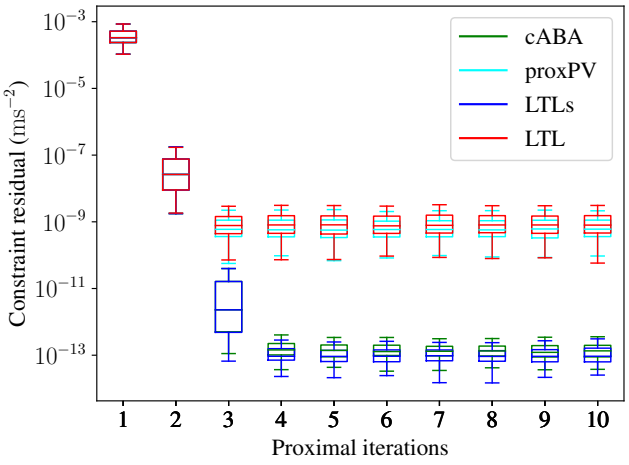


Fig. 4: Benchmarking convergence of the different proximal dynamics algorithms on a Talos robot with 24 constraints ($F_6^2 H_6$ and 3D constraint on elbow). This leads to a redundant constraint formulation that requires proximal methods. A whisker plot of convergence of the constraint residual is plotted for 10 proximal iterations $\mu = 10^6$ for 1000 randomly generated examples.

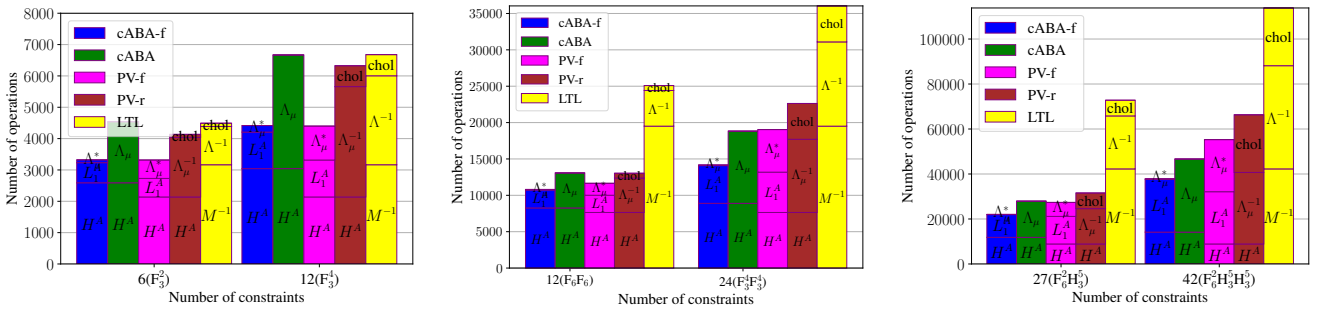
the Delassus matrix factorization.

Since cABA-OSIMf and PV-OSIMf are both tailored for floating-base robots with branching at the base, they outperform the other algorithms in benchmark cases, where the robots conform to this structure. However, for a higher number of constraints, the lower complexity cABA-OSIMf algorithm scales better because it does not require the factorization of L_1^A, unlike the PV-OSIMf algorithm.

TABLE III: Computational timings in μs of the damped Delassus inverse algorithms with `Turbo Boost` enabled. * LTL-chol computes the Cholesky decomposition of Λ_μ^{-1} unlike the other algorithms which actually compute Λ_μ.

System	cABA-f	cABA	LTL-chol*	LTL-inv
Iiwa - H_3	0.7	0.70	0.53	0.63
Iiwa - H_6	0.77	0.77	0.63	0.81
Solo - F_3^2	1.57	1.68	1.38	1.57
Solo - F_3^4	1.94	2.28	1.86	2.48
Talos - F_6^2	4.05	4.21	5.73	6.42
Talos - $F_6^2 H_6^2$	4.57	4.95	8.42	10.9
Atlas SR - F_6^2	6.37	6.5	11.4	12.1
Atlas SR - $F_6^2 T_3^5$	8.8	9.3	17.8	21.2
Atlas SR - F_6^2	12.6	13.3	26.2	35.9

Computation timings Table III lists the computation timings for a subset of the algorithms from the previous section, that we implemented in `C++`. Similarly to the constrained dynamics benchmarking, we enabled `TurboBoost` since this is more reflective of the actual usage of these algorithms. However, there are two main differences compared to the previous benchmarks. Firstly, we also include the cost for computing the damped Delassus inverse using LTL in the



(a) Unitree Go1 robot. (b) Talos robot. (c) Atlas equipped with two Shadow Hands.

Fig. 5: Benchmarking the number of operations of the damped Delassus inverse/factorization algorithms.

last column, to make the comparison fairer with our cABA-OSIM algorithms, which compute the inverse and not just the Cholesky decomposition of Λ_μ^{-1}. Computing the inverse is interesting because it makes computing analytical derivatives faster because of vectorization effects [56]. Secondly, the cABA-OSIM and cABA-OSIMf algorithms are implemented in a global frame instead of a local frame compared to the previous sections. Using global frame negatively affects cABA-OSIM and cABA-OSIMf more than the LTL methods because of the loss of sparsity in the computation of D_i and P_i (note that S_i typically has only one non-zero element for a local frame implementation, while it is dense for a global frame implementation). This choice was made to facilitate our future work on using the Delassus inverse matrix to compute analytical derivatives of constrained dynamics, which is faster with global frames.

As expected cABA-OSIM and cABA-OSIMf significantly outperform the LTL methods for larger robots. However, for smaller robots including an 18 DoF quadruped, the LTL method remains competitive.

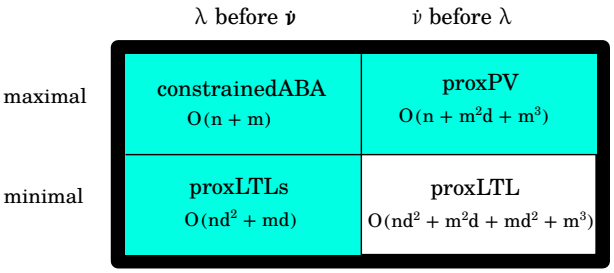


Fig. 6: Overview of the proximal dynamics algorithms.

IX. Discussion

The proximal formulation of the equality-constrained dynamics problem results in four different algorithms, namely proxLTL, proxLTLs, proxPV, and constrainedABA depending on the usage of maximal or minimal coordinates during derivation and the order of elimination of the primal and dual variables, as shown in the overview figure Fig. 6. These algorithms generalize several existing algorithms. For instance, MuJoCo's equality-constrained dynamics algorithm and PV-soft [13] can be considered to be the first iteration of proxLTLs and constrainedABA respectively. Our analysis also revealed that PV-soft can be recovered by applying PV-early on the proximal formulation, thereby connecting two existing algorithms that were considered to be disparate.

Though applying PV-early's idea of aggressively eliminating dual variables to the proximal dynamics formulation immediately gives us the first iteration of constrainedABA, constrainedABA generalizes this idea by introducing proximal iterations. Moreover, PV-early's superior performance in [13] relied on an efficient formula to compute the SVD of a symmetric rank-1 (SR1) 6×6 matrix arising due to one DoF joints and would become inefficient for more general higher DoF joints. However, constrainedABA avoids this limitation by eliminating all dual variables, thereby not requiring any SVD computation, and is therefore readily applicable to robots with higher DoF joints without compromising performance. However, we must note that we expect the vanilla PV-early algorithm (without proximal regularization) to marginally outperform constrainedABA for robots that predominantly possess single DoF joints and no singular constraints, as it does not require an iterative procedure. However, constrainedABA is significantly easier to implement than the vanilla PV-early algorithm, remains efficient for multi-DoF joints, and is valid even in singular cases.

ConstrainedABA, the main contribution of this paper, is the first algorithm with a $O(n+m)$ complexity that can reliably deal with singular constraints without biasing solutions towards the origin or resorting to expensive SVD computation. We also derived cABA-OSIM, a $O(n+m^2)$ complexity algorithm, to compute the damped Delassus inverse algorithm, also known as the OSIM. All existing OSIM algorithms have an extra m^3 term in the general case due to the factorization of the Delassus matrix. Since the damped Delassus inverse matrix is an $m \times m$ matrix and its computation requires considering all the joints, the asymptotic complexity achieved by us is optimal.

In addition to deriving these algorithms, we implemented them in C++, benchmarked them for various robots, and studied their scaling for chains and binary trees. The comparison between the algorithms was made based on the number of operations and the computation timings of an efficient C++ im-

plementation. The lower-complexity algorithms, constrainedABA, and proxPV were significantly more efficient than proxLTLs and proxLTL regarding the number of operations for larger robots like humanoids. However, the performance difference between the low-complexity and high-complexity algorithms was reduced for the C++ implementation. This is because the high-complexity algorithms enjoy CPU vectorization benefits from an efficient implementation in [20]. Devising a similarly efficient vectorized implementation for the low-complexity algorithms is far from straightforward as there is an inherent sequentiality in the recursive algorithm, so this aspect is not considered within the scope of this article. Nevertheless, constrainedABA emerged to be the fastest of the four algorithms for quadrupeds and humanoids even for the C++ implementation, being over 2x faster than proxLTL, the previously existing state-of-the-art C++ implementation. However, the proxLTLs algorithm turned out to be surprisingly competitive with constrainedABA by requiring only 20% more time for a heavily constrained Talos robot.

Our optimal complexity algorithm, cABA-OSIM, was also benchmarked with existing algorithms in terms of the number of operations required and computation timing of a C++ implementation. While it was found to require over 3x fewer operations compared to the LTL-OSIM methods, this difference was reduced for the C++ timings because of the efficient vectorized implementation of LTL-OSIM in the PINOCCHIO library. Despite this reduction, the cABA-OSIM remained over 2x faster than the LTL method for a large robot-like Talos.

X. CONCLUSION

We have applied the proximal point algorithm from optimization literature to equality-constrained robot dynamics problems for kinematic trees. The proximal formulation enables handling challenging singular cases efficiently and solves motion constraints exactly. Focusing on the algorithmic aspect of proximal constrained dynamics algorithms, our extensive treatment derived four different algorithms, proxLTL, proxLTLs, proxPV, and constrainedABA depending on the choice of maximal/minimal coordinate formulation and the primal/dual elimination order. Two of these (constrainedABA and proxPV) are original contributions of this article, while the third (proxLTLs) is implemented and benchmarked for the first time, though previously mentioned in [20].

ConstrainedABA, this article's main contribution, is the first linear complexity algorithm that deals with singular cases without resorting to expensive SVD computation, that we are aware of, and perhaps the simplest. The proximal formulation and the matrix inversion lemma also led to a beautiful result, where any Delassus matrix algorithm could be used to directly compute the damped Delassus inverse matrix, avoiding the expensive matrix factorization step. By using PV-OSIMr algorithm, we obtained cABA-OSIM algorithm with an optimal complexity of $O(n + m^2)$. The algorithms are implemented in C++ as a part of the Pinocchio library and are extensively benchmarked. ConstrainedABA emerged as the fastest out of the four algorithms for larger robots like quadrupeds and humanoids. The higher complexity proxLTLs surprisingly remained competitive even for larger robots, when heavily constrained, due to linear complexity in constraint dimension and an efficient vectorized C++ implementation.

The proximal formulation generalizes and establishes connections between existing algorithms like MUJOCO's solver, PV-soft, and PV-early. It is a powerful formulation that enables efficient trading-off between MUJOCO-style compliance and an expensive SVD-style rigid contact during singular cases (and also in general) through fast proximal iterations. ConstrainedABA and proxLTLs, in particular, are fairly straightforward to implement, by introducing only a few new lines of additional code compared to Featherstone's original ABA and LTL algorithms. This facilitates their implementation in existing simulators like MUJOCO, DRAKE, or RAISIM, to name a few of the most popular within the robotics community.

The favorable properties of the proximal point algorithm such as fast proximal iterations, differentiability of the proximal operator, and the cABA-OSIM set us up for several interesting future directions. We expect constrainedABA and cABA-OSIM, in particular, to serve as the computational backbone for our planned future work on extending proximal methods to efficiently deal with inequality constraints [38], [57], frictional contacts [4], [58]–[61], and analytical derivatives [6], [42], [56], [62].

ACKNOWLEDGEMENTS

The authors warmly thank Antoine Bambade for helpful discussions on the proximal-point method and augmented Lagrangian algorithms.

This work was supported by the Belgium government through the FWO project G0D1119N of the Research Foundation - Flanders (FWO - Flanders), 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 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] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[3] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu, "DART: Dynamic animation and robotics toolkit," *The Journal of Open Source Software*, vol. 3, no. 22, p. 500, Feb 2018. [Online]. Available: https://doi.org/10.21105/joss.00500

[4] 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

[5] 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.

[6] 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.

[7] A. Vereshchagin, "Computer simulation of the dynamics of complicated mechanisms of robot-manipulators," *Eng. Cybernet.*, vol. 12, pp. 65–70, 1974.

[8] C. F. Gauß, "Über ein neues allgemeines grundgesetz der mechanik." 1829.

[9] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[10] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias," *Int. J. Robot. Res.*, vol. 2, no. 1, pp. 13–30, 1983.

[11] 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.

[12] 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.

[13] A. S. Sathya, H. Bruyninckx, W. Decré, and G. Pipeleers, "Efficient constrained dynamics algorithms based on an equivalent lqr formulation using gauss' principle of least constraint," *IEEE Transactions on Robotics*, vol. 40, pp. 729–749, 2024.

[14] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," 1982.

[15] 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.

[16] J. P. Popov, A. F. Vereshchagin, and S. L. Zenkevič, *Manipuljacionnyje roboty: Dinamika i algoritmy*. Nauka, 1978.

[17] 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.

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

[19] R. Featherstone, "Exploiting sparsity in operational-space dynamics," *Int. J. Robot. Res.*, vol. 29, no. 10, pp. 1353–1368, 2010.

[20] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and sparse resolution of constrained dynamic equations," in *Proc. Robot., Sci. Syst.*, 2021.

[21] N. Parikh, S. Boyd *et al.*, "Proximal algorithms," *Foundations and trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[22] É. Delassus, "Mémoire sur la théorie des liaisons finies unilatérales," in *Annales scientifiques de l'École normale supérieure*, vol. 34, 1917, pp. 95–179.

[23] B. Brogliato, *Nonsmooth mechanics*. Springer, 1999, vol. 3.

[24] 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.

[25] K. Kreutz-Delgado, A. Jain, and G. Rodriguez, "Recursive formulation of operational-space control," *Int. J. Robot. Res.*, vol. 11, no. 4, pp. 320–328, 1992.

[26] P. Wensing, R. Featherstone, and D. E. Orin, "A reduced-order recursive algorithm for the computation of the operational-space inertia matrix," in *Proc. IEEE Int. Conf. Robot. Autom.* IEEE, 2012, pp. 4911–4917.

[27] K.-S. Chang and O. Khatib, "Efficient recursive algorithm for the operational space inertia matrix of branching mechanisms," *Advanced Robotics*, vol. 14, no. 8, pp. 703–715, 2001.

[28] A. S. Sathya, W. Decré, and J. Swevers, "Low order complexity algorithm for computing the inverse operational space inertia matrix," Oct. 2023, in revision for IEEE RA-L. [Online]. Available: https://doi.org/10.48550/arXiv.2310.03676

[29] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[30] 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.

[31] 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.

[32] F. E. Udwadia and R. E. Kalaba, *Analytical dynamics : a new approach*. Cambridge: Cambridge University press, 1996.

[33] 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.

[34] 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.

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

[36] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[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] 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.

[39] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, "Constrained differential dynamic programming: A primal-dual augmented lagrangian approach," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 13 371–13 378.

[40] O. Güler, "On the convergence of the proximal point algorithm for convex minimization," *SIAM journal on control and optimization*, vol. 29, no. 2, pp. 403–419, 1991.

[41] 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.

[42] A. Bambade, F. Schramm, A. Taylor, and J. Carpentier, "QPLayer: efficient differentiation of convex quadratic optimization," in *2024 International Conference on Learning Representations (ICLR)*, 2023.

[43] M. R. Hestenes, "Multiplier and gradient methods," *Journal of optimization theory and applications*, vol. 4, no. 5, pp. 303–320, 1969.

[44] M. J. Powell, "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298, 1969.

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

[46] R. Rockafellar, "Minimax theorems and conjugate saddle-functions," *Mathematica Scandinavica*, vol. 14, no. 2, pp. 151–173, 1964.

[47] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

[48] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, 2017, vol. 2.

[49] M. L. Felis, "Rbdl: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, vol. 41, no. 2, pp. 495–511, 2017.

[50] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[51] K. W. Lilly, *Efficient dynamic simulation of multiple chain robotic systems*. The Ohio State University, 1989.

[52] 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.

[53] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols *et al.*, "An open torque-controlled modular robot architecture for legged locomotion research," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3650–3657, 2020.

[54] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux *et al.*, "Talos: A new humanoid research platform targeted for industrial applications," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 689–695.

[55] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "Casadi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[56] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Proc. Robot., Sci. Syst.*, 2018.

[57] W. Jallet, A. Bambade, E. Arlaud, S. El-Kazdadi, N. Mansard, and J. Carpentier, "PROXDDP: Proximal Constrained Trajectory Optimization," Dec. 2023, working paper or preprint. [Online]. Available: https://inria.hal.science/hal-04332348

[58] P. C. Horak and J. C. Trinkle, "On the similarities and differences among contact models in robot simulation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 493–499, 2019.

[59] A. M. Castro, F. N. Permenter, and X. Han, "An unconstrained convex formulation of compliant contact," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1301–1320, 2022.

[60] Q. L. Lidec, W. Jallet, L. Montaut, I. Laptev, C. Schmid, and J. Carpentier, "Contact Models in Robotics: a Comparative Analysis," *arXiv preprint arXiv:2304.06372*, 2023.

[61] Q. Le Lidec and J. Carpentier, "Reconciling RaiSim with the Maximum Dissipation Principle," Feb. 2024, working paper or preprint. [Online]. Available: https://hal.science/hal-04438175
[62] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu, "Fast and Feature-Complete Differentiable Physics Engine for Articulated Rigid Bodies with Contact Constraints," in *Proceedings of Robotics: Science and Systems*, Virtual, July 2021.