Efficient Constrained Dynamics Algorithms Based on an Equivalent LQR Formulation Using Gauss' Principle of Least Constraint

Ajay Suresha Sathya, Herman Bruyninckx, Wilm Decré, *Member, IEEE*, and Goele Pipeleers

***Abstract*—We derive a family of efficient constrained dynamics algorithms by formulating an equivalent linear quadratic regulator (LQR) problem using Gauss' principle of least constraint and solving it using dynamic programming. Our approach builds upon the pioneering (but largely unknown) $O(n + m^2\ d + m^3)$ solver by Popov and Vereshchagin (PV), where n, m, and d are the number of joints, number of constraints, and the kinematic tree depth, respectively. We provide an expository derivation for the original PV solver and extend it to floating-base kinematic trees with constraints allowed on any link. We make new connections between the LQR's dual Hessian and the inverse operational space inertia matrix (OSIM), permitting efficient OSIM computation, which we further accelerate using matrix inversion lemma. By generalizing the elimination ordering and accounting for MuJoCo-type soft constraints, we derive two original $O(n + m)$ complexity solvers. Our numerical results indicate that significant simulation speed-up can be achieved for high dimensional robots like quadrupeds and humanoids using our algorithms as they scale better than the widely used $O(nd^2 + m^2\ d + d^2\ m)$ LTL algorithm of Featherstone. The derivation through the LQR-constrained dynamics connection can make our algorithm accessible to a wider audience and enable cross fertilization of software and research results between the fields.**

***Index Terms*—Direct/Inverse dynamics formulation, dynamics, optimization and optimal control, redundant robots.**

Manuscript received 4 April 2023; revised 26 August 2023; accepted 18 October 2023. Date of publication 24 November 2023; date of current version 21 December 2023. This paper was recommended for publication by Associate Editor M. Posa and Editor E. Yoshida upon evaluation of the reviewers' comments. This work was supported in part by the Flanders Make (Flanders Make is the Flemish Strategic Research Centre for the manufacturing industry), through the Flanders Make SBO project - MULTIROB, and in part by the Research Foundation Flanders (FWO) under Grant G0D1119 N. *(Corresponding author: Ajay Suresha Sathya.)*

Ajay Suresha Sathya and Wilm Decré are with the Division of Robotics, Automation and Mechatronics in the Department of Mechanical Engineering, KU Leuven, 3001 Leuven, Belgium, and also with the DMMS-M Lab, Flanders Make, 3001 Leuven, Belgium (e-mail: ajay.sathya@kuleuven.be; wilm.decre@kuleuven.be).

Herman Bruyninckx is with the Division of Robotics, Automation and Mechatronics in the Department of Mechanical Engineering, KU Leuven, 3001 Leuven, Belgium, and also with the TU Eindhoven, 5600 MB Eindhoven, The Netherlands (e-mail: herman.bruyninckx@kuleuven.be).

Goele Pipeleers is with the Division of Robotics, Automation and Mechatronics in the Department of Mechanical Engineering, KU Leuven, 3001 Leuven, Belgium, and also with the Materialise NV, 3001 Leuven, Belgium (e-mail: goele.pipeleers@kuleuven.be).

This article has supplementary downloadable material available at https://doi.org/10.1109/TRO.2023.3335652, provided by the authors.

Digital Object Identifier 10.1109/TRO.2023.3335652

NOMENCLATURE

Symbol	Definition
${}^{\{j\}}X_i$	Spatial pose of ith link in jth link's frame.
$\mathbf{v}_i$	6-D spatial velocity of the ith link.
$\mathbf{a}_i$	6-D spatial acceleration of the ith link.
$\mathbf{f}_i$	6-D spatial force acting on the ith link.
$\mathbf{q_p}$	Vector of robot joint positions.
$\dot{\mathbf{q}}$	Vector of robot joint velocities.
$\ddot{\mathbf{q}}$	Vector of robot joint accelerations.
$\boldsymbol{\tau}$	Vector of robot joint torques.
n	DOF of the robot.
K_i	Acceleration constraint matrix on ith link.
$\mathbf{k}_i$	Desired constraint accelerations.
J_i	Geometric Jacobian of the ith link.
$\dot{J}_i$	Time derivative of J_i.
J	Joint-space constraint Jacobian.
$\dot{J}$	Time derivative of J.
$\mathbf{k}$	Concatenation of all $\mathbf{k}_i$.
m	Number of acceleration constraints on the robot.
M	Joint-space inertia matrix.
$\mathbf{c}$	Joint torques due to bias accelerations, forces, and gravity.
$\boldsymbol{\lambda}$	Lagrange multipliers of constraints.
Λ	Operational-space inertia matrix.
L	Lower triangular matrix in LTL decomposition [43].
Y	Intermediate quantity in LTL-OSIM [43], see Section II-C.
$\pi(i)$	Index of ith link's parent link.
$\gamma(i)$	Set of ith link's children links' indices.
S_i	Motion subspace of the ith joint.
T_i	Force subspace of the ith joint.
H_i	6×6 spatial inertia tensor of the ith link.
$\mathbf{a}_{b,i}$	ith link's bias acceleration.
$\mathcal{L}$	Lagrangian of the LQR problem.
V_i	Cost-to-go Lagrangian at ith link.
H_i^A	Articulated body inertia of ith link.
L_i^A	Constraint's coupling due ith and its descendant joints.
K_i^A	Constraint force propagated to the ith link.
$\mathbf{f}_i^A$	Resultant force on i link excluding constraint forces.
$\mathbf{l}_i^A$	Desired constraint accelerations propagated to the ith link.
D_i	Apparent articulated body inertia along the ith joint.
P_i	Backward force propagator through the ith joint.
$\mathbf{f}_i^{\text{ext}}$	Resultant external wrench acting on the ith link.

IEEE Transactions on Robotics (T-RO) paper, presented at ICRA 2024, Yokohama, Japan. Cite as T-RO paper.

b Index of the floating-base link.
$\boldsymbol{\lambda}_i^A$ Concatenated multipliers on ith and its descendant links.
r Number of branches from the floating-base link.

I. Introduction

RIGID body mechanics is a long-studied field with fundamental contributions already made in the 18th and 19th centuries. Since the 1970s, robotics research has focused on developing computationally efficient dynamics algorithms [1]. Initial motivation for this research was to enable real-time dynamic simulation and computed torque control on the slow computers of the 1970s. Despite significant processor clock-time improvements since then, computing dynamics efficiently remains a relevant problem because it can positively impact modern robotics applications involving model predictive control (MPC) and reinforcement learning. Faster computation enables MPC control designers to increase the prediction horizon, which usually improves optimality and stability properties of the MPC controller [2]. It can speed up contact-aware online trajectory optimization [3], [4] and also shorten long training times in reinforcement learning from simulations. Unsurprisingly, implementing efficient dynamics simulators remains an active research area [5], [6], [7], [8], [9].

A. Related Work

However, efficient dynamics algorithms are typically complex with "a steep learning curve" [10] and are not discussed in introductory robotics textbooks [11], [12]. Consequently, robotics researchers often use dynamics algorithms, especially constrained dynamics algorithms (CDA) [with motion constraints like the nonpenetration for the Go1 robot's feet in Fig. 1(a)], implemented in simulators as a black-box and are, therefore, unable to adapt or debug the algorithms to suit their applications. By deriving efficient CDA as the solution of an equivalent equality-constrained linear quadratic regulator (LQR) problem, we believe that this article makes efficient CDAs accessible to researchers with an optimization and control background. This includes many roboticists that are MPC practitioners due to the rising popularity of differential dynamic programming style [13] algorithms. The optimization-based perspective as well as the LQR connection opens up possibilities for transfer of software and recent research results between the fields, especially the recent data-driven methods for safe control of systems with uncertain dynamics [14]. Our derivation is also self-contained and does not assume prior knowledge of LQR derivation.

The first efficient recursive algorithms, with $O(n)$ complexity in the number of joints, for computing the unconstrained forward dynamics were independently discovered by Vereshchagin [15] and Featherstone [16]. However, Vereshchagin's solver "was way ahead of its time and languished in obscurity for a decade" [17]. Featherstone's insight involved efficiently propagating the solution of the Newton–Euler equations through the links, whereas Vereshchagin's approach was based on optimizing the Gauss' principle of least constraint (GPLC) [18], using dynamic programming (DP) [19]. Vereshchagin's idea is analogous to the standard textbook approach for solving the discrete-time linear quadratic regulator (LQR) problem using DP [2, Ch. 1], which we will use in the rest of this article. Similar connection to the LQR problem was independently made in [20] by noting similarities between the Kalman filter and $O(n)$ recursive dynamics algorithms, and this connection was further developed within a *spatial operator algebra* (SOA) framework [21], [22], making efficient $O(n)$ dynamics algorithms accessible to researchers familiar with filtering theory. However, the SOA derivation is fairly complex, is performed over several papers and assumes strong familiarity with filtering theory literature and notation from 1960 s and 1970s. Moreover, the SOA derivation does not permit a straightforward extension to constrained dynamics. Unlike SOA, our LQR approach starts with the optimization problem arising from first principles, includes motion constraints and readers will find our derivation to be a significantly simpler and more direct connection to LQR than [20].

The simplicity arises from GPLC straightforwardly modeling motion constraints as constraints in the associated optimization problem. This ease of modeling allowed Popov and Vereshchagin (PV) to quickly extend their forward dynamics algorithm to an efficient $O(n)$ CDA [23], [24] for fixed-base kinematic chains with end-effector constraints. But this extension of the LQR connection to constrained dynamics remains largely unknown and unused by the robotics community despite its simplicity and efficiency. There have been a few robot control architectures using the PV solver [25], [26], including an implementation in OROCOS-KDL[1] for kinematic chains, but its wider usage remains limited. Shakhimardanov[25] also derived the PV solver by introducing the concept of "acceleration energy" and extended it to trees by assembling acceleration energies. For readers unfamiliar with acceleration energy, their derivation is hard to follow and verify, whereas in this article we provide an expository derivation purely using the mathematical perspective of the LQR problem.

Other independent contributions that can be used to solve constrained dynamics includes the well-known operational-space formulation [27]. However, Khatib [27] does not propose an efficient algorithm for computing the operational-space inertia matrix (OSIM), which has a computational complexity of $O(n^3)$ when computed naively in joint-space. A major contribution to computing OSIM efficiently came in the form of an $O(n + m^2\ d + m^3)$ complexity recursive algorithm in [28] and [29], where m is the number of constraints and d is the tree depth. An efficient formula for computing off-diagonal blocks of the inverse OSIM using extended force propagators (EFP) was proposed in [30]. However, they do not exploit this EFP idea in their proposed algorithm and instead used a recursive approach similar to [28] to obtain $O(n + mn + m^3)$ complexity [30]. The idea of EFP was fully exploited in the EFP algorithm (EFPA) [31] to obtain a reduced complexity of $O(n + md + m^3)$. In [32], Featherstone reported that exploiting the branching-induced sparsity in the joint-space inertia matrix (JSIM) and the kinematic Jacobian to compute the OSIM more efficiently than the existing recursive $O(n)$ algorithms despite having a worse $O(nd^2 + md^2 + dm^2)$ complexity (where d

[1][Online]. Available: https://www.orocos.org/kdl.html

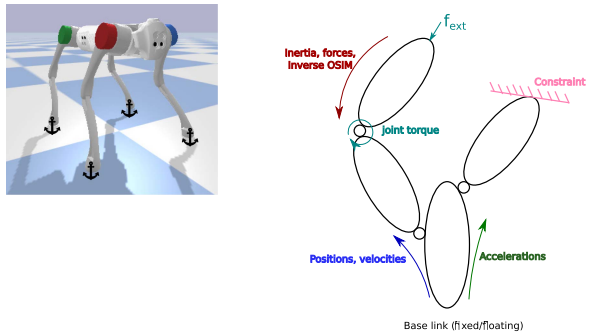
is the depth of the tree) even for the Honda Asimo robot, a complex robot with $n = 40$. This result has led to a much wider usage of Featherstone's higher complexity method in simulators like MuJoCo, Pinocchio [6], Raisim and RBDL, to name a few, compared with the lower complexity algorithms like EFPA [31]. Carpentier et al. [6] derived Featherstone's OSIM algorithm [32] from the perspective of factorizing the contact karush-kuhn-tucker (KKT) matrix and utilizes proximal-point iterations to solve for systems with redundant constraints.

Independent efforts to extend the efficient articulated body algorithm (ABA) algorithm to internal kinematic closed-loop constraints were realized in [33] and [34]. With the loop-closure constraint being a more general constraint model than the simpler desired acceleration-relative-to-ground constraint model considered in the PV solver, these more general algorithms include the PV solver computations as a subset of their computations. These algorithms can be straightforwardly adapted to kinematic trees with acceleration-relative-to-ground constraints to obtain an algorithm virtually identical to the PV solver. The derivation in [33] relies heavily on the physical insight of the readers, whereas the derivation in [34] is relatively more formal by algebraically solving the d'Alembert's equations. Otter et al. [33] further proposed a form of early constraint elimination that provides an $O(m + n)$ complexity algorithm for certain kinematic mechanisms. Similar ideas were also used in [35] to obtain an $O(n + m)$ algorithm based on Kane's equations [36]. However, our PV solver derivation approach is different, and we will discuss in detail the comparison with these algorithms in Section IX-E. Moreover, we are not aware of any open-source implementation of [33] and [34] or its computational comparison with the popular Featherstone's sparsity exploiting algorithms.

Another line of research for accelerating dynamics computations includes the divide-and-conquer type of algorithms that aim to exploit parallel computing [37], [38], [39], [40], achieving an $O(\log(n))$ complexity using $O(n)$ computational cores. These algorithms compute constrained dynamics by placing handles on the constrained bodies. The PV solver derived in Section V can also be interpreted as an algorithm that computes the relative inertia of these handles. Bhalerao et al. [41] presents a distributed algorithm specifically for computing the OSIM. Comparison of the PV solver with these parallel algorithms is further discussed in Section IX-E.

The efficient algorithms discussed so far have complex derivations, a third simple approach pioneered in [10], involves constructing the KKT matrix in "maximal" coordinates and solving it using a sparse linear solver. Despite having a favorable $O(n + md + m^3)$ complexity, Barraf's [10] algorithm, does not exploit as much structure as possible (for example, it computes joint constraint forces, which are avoided in other methods) and requires joint constraint stabilization. It is generally not considered to be competitive with the recursive or sparse factorization methods mentioned above [1].

The PV solver derivation using the LQR connection has the elegance and simplicity of Baraff's derivation, with a three-sweep structure that is analogous to forward simulation, backward DP recursion, and rollout in LQR control, as shown in Fig. 1(b).



(a) (b)

Fig. 1. Structure-exploiting constrained dynamics algorithm. (a) Environment imposes constraints on a robot, which must be accounted for in dynamics equations. (b) Three sweep structure of the PV solver: Paralleling forward simulation, backward DP recursion, and rollout.

We also found it to be more efficient than the state-of-the-art algorithms as we will show in the rest of this article.

B. Contributions

1) Expository Derivation of the Original PV Solver and Extensions: We provide an expository derivation of the original PV solver by adapting the textbook approach for solving the LQR problem [2], highlighting its connection to constrained dynamics more clearly than in existing literature. We then derive extensions to the original PV solver to support: 1) floating-base robots, 2) constraints potentially on any link, and 3) kinematic trees and show its computational complexity to be $O(n + m^2\ d + m^3)$.

2) Connections to the OSIM: We show that the dual Hessian, that is computed as an intermediate step of the PV solver, is equal to the inverse OSIM. This connection is new in literature, to the best of the authors' knowledge, and provides an efficient $O(n + m^2\ d + m^2)$ algorithm, that is, as of yet unexploited to compute the inverse OSIM. This algorithm is structurally different from the currently known $O(n)$ family algorithms kreutz-jain-rodriguez (KJR) [28] and EFPA [30], [31], by requiring only two sweeps over the kinematic tree instead of three and is found to be more efficient in practice for most robots of interest despite having a worse complexity than the $O(n + md + m^2)$ EFP algorithm. We further accelerate OSIM computation for floating-base robots with branching structure at the base.

3) $O(n + m)$ Algorithms: Building upon our expository PV solver treatment, we derive two efficient and new (to the best of the authors' knowledge) CDA with only $O(n + m)$ computational complexity. The first algorithm solves the so-called "soft Gauss principle" used in the popular robot dynamics simulator MuJoCo [7], [42], that relaxes the hard motion constraints with quadratic penalties. The second algorithm solves the original problem with hard motion constraints, by incorporating early elimination of Lagrange multipliers, thereby limiting their backward propagation, which provides the improved computational complexity.

4) Benchmarking: Despite the PV solver and Otter et al.'s [33] contributions being over 35 years old, their computational performance is untested against the

state-of-the-art algorithms, that are currently recognized to be fast in literature. We provide a comprehensive benchmarking of the PV solver against Featherstone's sparsity-exploiting algorithms [32], [43] (currently used most widely in high-performance robot simulators, including the PINOCCHIO and MUJOCO toolboxes), the lower order EFPA [30], [31] algorithm, as well as our $O(n+m)$ extensions to the PV solver. These numerical results are new in literature to the best of the authors' knowledge.

The source code of the solver is made available publicly.[2]

C. Organization

The rest of this article is organized as follows. We first discuss background material and preliminaries in Section II and derive the PV solver for a kinematic chain with a fixed-base and motion constraints only on the end-effector in Section III. We then discuss the physical interpretation of the terms of this relatively simple algorithm and also show the equality of the dual Hessian of the constrained LQR problem and the inverse OSIM in Section IV. Later, we generalize the derivation to the more complex case of floating-base robots with a kinematic tree structure and constraints on any link in Section V. This separation of the PV solver derivation into two sections was made for clarity of exposition as it is easier to first follow the derivation for fixed-base kinematic chains before the generalization to trees. We then present an efficient extension of the PV solver to "soft" motion constraints in Section VI. We expand upon the dual Hessian-OSIM connection in Section VII and finish our derivations with a fast $O(n+m)$ algorithm for the original problem with hard motion constraints in Section VIII. Section IX presents algorithm benchmarking and discussions. Finally, Section X concludes this article.

II. BACKGROUND

A. Notation and Convention

Nomenclature lists the notation used in this article. Bold-faced lower case letters or symbols are vectors and upper case letters or symbols are matrices. A^T is the transpose of a matrix A. $I_{n\times n}$ and $0_{n\times n}$ are the identity matrix and zero matrix of dimension $n\times n$, respectively. The $:=$ operator defines the left-hand side (LHS) symbol with the right-hand side (RHS) expression. The $\leftarrow$ operator assigns the RHS expression to a LHS variable in an algorithm.

We use the popular Featherstone's spatial algebra notation [1] throughout the article. For a robot's ith rigid body, $X_i \in SE(3)$, $\mathbf{v}_i \in \mathbb{M}^6$, and $\mathbf{a}_i \in \mathbb{M}^6$ denote the spatial pose, velocity, and acceleration, respectively. $SE(3)$ is the special Euclidian group in three dimensions represented as a 6×6 spatial transformation matrix. $\mathbf{f}_i \in \mathbb{F}^6$ is the spatial force acting on the ith body. For notational simplicity of the upcoming derivations, all motion/force vectors $\mathbf{v}_i, \mathbf{a}_i$, and $\mathbf{f}_i$ are with respect to a common inertial frame. $\times$ and $\times^*$ are the spatial cross-product operators for motion and force vectors, respectively.

[2][Online]. Available: https://github.com/AjSat/spatial_V2

The whole robot's state is $(\mathbf{q_p}, \dot{\mathbf{q}})$, where $\mathbf{q_p} \in \mathcal{Q}$ is its pose in the configuration space $\mathcal{Q}$, $\dot{\mathbf{q}} \in \mathcal{T}_{\mathbf{q_p}}\mathcal{Q} \simeq \mathbb{R}^n$ is its generalized velocity in $\mathcal{Q}$'s tangent space at $\mathbf{q}_p$, and n is the robot's degrees of freedom (DOF). Let $\boldsymbol{\tau} \in \mathcal{T}^*_{\mathbf{q_p}}\mathcal{Q} \simeq \mathbb{R}^n$ be the generalized force acting on the robot in the dual tangent space of $\mathcal{Q}$ and $\ddot{\mathbf{q}} \in \mathbb{R}^n$ be $\dot{\mathbf{q}}$'s time derivative. This Lie algebraic notation allows a unified representation of floating-base robots and multiDOF joints where a singularity-free representation of position may require $n_p \geq n$. For a fixed-base manipulator with single DOF joints, $\mathbf{q_p} = \mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\boldsymbol{\tau}$ are simply the joint positions, velocities, accelerations, and torques, respectively.

B. Preliminaries

We will now briefly summarize forward dynamics, inverse dynamics, and constrained dynamics problems. Forward dynamics computes $\ddot{\mathbf{q}}$, that result from applying $\boldsymbol{\tau}$ on a given robot at state $(\mathbf{q_p}, \dot{\mathbf{q}})$, to simulate the robot state forward in time. Conversely, inverse dynamics computes the $\boldsymbol{\tau}$ required to obtain a desired $\ddot{\mathbf{q}}$ at state $(\mathbf{q_p}, \dot{\mathbf{q}})$. Constrained dynamics is the forward dynamics problem with motion constraints in addition to joint constraints and will be formalized in the next paragraph. Inverse dynamics is, in general, easier to compute than forward dynamics, which is, in turn, significantly easier to compute than constrained dynamics.

Let the acceleration constraint on the ith link be

$$K_i(\mathbf{q_p})\mathbf{a}_i = \mathbf{k}_i(\mathbf{q_p}, \dot{\mathbf{q}}) \tag{1}$$

with $K_i \in \mathbb{R}^{m_i\times 6}$, $\mathbf{k}_i \in \mathbb{R}^{m_i}$, and m_i the constraint dimensionality. Without loss of generality, we scale the constraints, such that each row of K_i has unit norm. Both holonomic and nonholonomic motion constraints can be converted to this form by differentiation [11]. The acceleration constraints can be transformed to the generalized coordinates using

$$\mathbf{a}_i = J_i(\mathbf{q_p})\ddot{\mathbf{q}} + \dot{J}_i(\mathbf{q_p}, \dot{\mathbf{q}})\dot{\mathbf{q}} \tag{2}$$

where $J_i(\mathbf{q_p}) \in \mathbb{R}^{6\times n}$ is ith link's geometric Jacobian and $\dot{J}_i(\mathbf{q_p}, \dot{\mathbf{q}}) \in \mathbb{R}^{6\times n}$ is its total time derivative. Substituting (2) in (1) and stacking all the links' constraints gives

$$J(\mathbf{q_p})\ddot{\mathbf{q}} + \dot{J}(\mathbf{q_p}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{k}(\mathbf{q_p}, \dot{\mathbf{q}}) \tag{3}$$

$$\text{where} \quad J(\mathbf{q_p}) := \begin{bmatrix} K_1(\mathbf{q_p})J_1(\mathbf{q_p}) \\ \vdots \\ K_i(\mathbf{q_p})J_i(\mathbf{q_p}) \\ \vdots \\ K_n(\mathbf{q_p})J_n(\mathbf{q_p}) \end{bmatrix} \in \mathbb{R}^{m\times n}, \quad \dot{J}(\mathbf{q_p}, \dot{\mathbf{q}}) := \begin{bmatrix} K_1\dot{J}_1 + \dot{K}_1J_1 \\ \vdots \\ K_i\dot{J}_i + \dot{K}_iJ_i \\ \vdots \\ K_n\dot{J}_n + \dot{K}_nJ_n \end{bmatrix} \in \mathbb{R}^{m\times n}, \text{ and } \mathbf{k}(\mathbf{q_p}, \dot{\mathbf{q}}) := \begin{bmatrix} \mathbf{k}_1 \\ \vdots \\ \mathbf{k}_i \\ \vdots \\ \mathbf{k}_n \end{bmatrix} \in \mathbb{R}^m.$$

The constrained dynamics problem involves simultaneously solving (3) and the linear system

$$M(\mathbf{q_p})\ddot{\mathbf{q}} + \mathbf{c}(\mathbf{q_p}, \dot{\mathbf{q}}) + J(\mathbf{q_p})^T\boldsymbol{\lambda} = \boldsymbol{\tau} \tag{4}$$

for unknowns $\ddot{\mathbf{q}}$ and $\boldsymbol{\lambda}$, where, $M(\mathbf{q_p}) \in \mathbb{R}^{n\times n}$, $\mathbf{c}(\mathbf{q_p}, \dot{\mathbf{q}}) \in \mathbb{R}^n$, and $\boldsymbol{\lambda} \in \mathbb{R}^m$ are the JSIM, generalized force due to Coriolis, centrifugal and gravity effects, and the Lagrange multipliers associated with the acceleration constraint, respectively. Solving for $\ddot{\mathbf{q}}$ in (4) (which is always possible because $M(\mathbf{q_p})$ is positive definite) and substituting in (3) gives the operational-space form of constrained dynamics [27] (with term dependencies dropped for brevity from now on, when it is clear from the context)

$$\Lambda^{-1}\boldsymbol{\lambda} = \dot{J}\dot{\mathbf{q}} - \mathbf{k} + JM^{-1}(\boldsymbol{\tau} - \mathbf{c}) \tag{5}$$

with $\Lambda(\mathbf{q_p})^{-1} := (J(\mathbf{q_p})(M(\mathbf{q_p}))^{-1}J(\mathbf{q_p})^T) \in \mathbb{R}^{m\times m}$ and $\Lambda(\mathbf{q_p})$ is the OSIM. The inverse OSIM $\Lambda(\mathbf{q_p})^{-1}$ captures the inertial coupling between constraints, where the ith column of $\Lambda(\mathbf{q_p})^{-1}$ is the acceleration along all the constraint directions caused by $\lambda_i = 1$ (ith constraint force with unit magnitude).

Remark 1: Since $M(\mathbf{q_p})$ is a positive definite matrix, if J has full row-rank, Λ^{-1} has full rank, is invertible, and Λ exists. Then, (5) permits a unique solution for $\boldsymbol{\lambda}$.

Remark 2: J may not have full row-rank in over-constrained systems, when constraints conflict with each other or due to loss of J_i's rank at kinematic singular configurations and depending on the numerical values of $\mathbf{k}$, there exists either no solution or an infinite number of solutions for $\boldsymbol{\lambda}$.

Typical strategies to address singular Λ^{-1} include Tikhonov regularization, proximal-point iterations [6], Moore–Penrose pseudo-inverse using the singular value decomposition (SVD), relaxing the constraints with weighted quadratic penalties [7] or employing prioritized conflict resolution [44]. Since a discussion of these different strategies is not the focus here, we assume that J has full row-rank in the rest of this article.

C. Featherstone's LTL Algorithms

We now review Featherstone's sparsity-exploiting algorithms and introduce terms that will be benchmarked in Section IX. The LTL algorithm [43] is a Cholesky decomposition for the JSIM

$$L^T L = M \tag{6}$$

where $L \in \mathbb{R}^{n\times m}$ is a lower triangular matrix. In contrast to the traditional LLT Cholesky algorithm [45], the LTL method ensures no fill-in (preserves the sparsity pattern of M in L) even without resorting to pivoting methods that choose an elimination ordering. The idea was extended in the LTL-OSIM algorithm [32] to compute the OSIM for kinematic trees, where the sparsity pattern of J is also exploited

$$Y = JL^{-1} \tag{7}$$

where $Y \in \mathbb{R}^{m\times n}$ also has the same sparsity pattern as J and

$$\Lambda^{-1} = YY^T. \tag{8}$$

D. Forward Kinematics

Let a kinematic tree have n links indexed from 1 to n. The world link (assumed to be a fixed inertial frame) is assigned the 0 index. The ith joint connects the ith link to its parent link $\pi(i)$. The world link is tree's root and does not have a parent link. For floating-base robots, such as quadrupeds, a chosen link b (usually the torso) is connected to the world link through a free joint. $\gamma(i)$ is the set of ith link's children. A link j is a leaf link if $\gamma(j) = \varnothing$.

The spatial poses, velocities, and accelerations of all links in the tree can be computed recursively in a forward sweep starting from the root (world link) using

$$X_j = \left(X_{\pi(j)}\right)\left({}^{\{\pi(j)\}}X_{j'}\right)\left({}^{\{j'\}}X_j\right) \tag{9}$$

$$\mathbf{v}_j = \mathbf{v}_{\pi(j)} + S_j\dot{\mathbf{q}}_j \tag{10}$$

$$\mathbf{a}_j = \mathbf{a}_{\pi(j)} + S_j\ddot{\mathbf{q}}_j + \mathbf{v}_j \times S_j\dot{\mathbf{q}}_j \tag{11}$$

where ${}^{\{\pi(j)\}}X_{j'}$ is the jth link's pose in its parent link's frame when the jth joint is at its home pose (usually computed from the robot unified robot description format (URDF) file or the Denavit–Hartenberg (DH) parameters) and ${}^{\{j'\}}X_j$ is the spatial transformation due to the jth joint's displacement. $S_j \in \mathbb{M}^{6\times n_j}$ is the jth joint's motion subspace, where n_j is the joint's DOF (usually 1). $S_j\dot{\mathbf{q}}_j$ is the jth joint's contribution to $\mathbf{v}_i$. Let $T_j \in \mathbb{F}^{6\times n_j}$ be the jth joint's force subspace, such that $T_j\boldsymbol{\tau}_j$ is the joint's contribution to $\mathbf{f}_j$.

Remark 3: The force subspace T_j is the dual of the motion subspace S_j, hence, $S_j^T T_j = I_{n_j\times n_j}$ [1, Eq. 3.39].

E. Gauss' Principle

GPLC [18] is an optimization-based formulation of classical mechanics, which is not as well known or widely used as the Lagrangian formulation. Refer [46] for a detailed discussion on GPLC, according to which, a constrained system under the influence of forces undergoes accelerations that are as close as possible (in a weighted least-squares sense) to the unconstrained motion of the system under the same nonconstraint forces. For a system of rigid bodies with spatial inertia tensor $H_i \in \mathbb{R}^{6\times 6}$ of the ith link, under the external forces $\mathbf{f}_i$, which includes the bias forces $\mathbf{v}_i \times^* H_i\mathbf{v}_i$, the resulting accelerations $\mathbf{a}_i$ are the minimizers of the following optimization problem [47]:

$$\underset{\mathbf{a}_1,\ldots,\mathbf{a}_n}{\text{minimize}} \quad \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right)^T H_i\left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right) \tag{12a}$$

$$\text{subject to} \quad \text{motion constraints.} \tag{12b}$$

F. Dynamic Programming Principle

DP [19] is a general theoretical framework for optimizing a function through a series of nested optimizations over the decision variables in some order. DP's efficiency can crucially depend on the variable elimination order. Each DP step optimizes over a function to return a function, so its computation is intractable, unless the intermediate functions can be efficiently parameterized. The discrete-time LQR problem is one

such exception, where all the intermediate functions have the quadratic form. Fortunately, for kinematic tree mechanisms, the GPLC problem is algebraically identical to the discrete-time LQR problem with scenario trees and can be solved efficiently using DP. This robot dynamics-LQR connection forms the basis of the derivations in this article.

III. Derivation of the Constrained Dynamics Solver

In this section we derive the PV solver for fixed-base kinematic chains with end-effector motion constraints. We first formulate the optimization problem in Section III-A, then, derive its solution using DP in Section III-B.

A. Problem Formulation

Consider a kinematic chain with the links indexed, such that $\pi(i) = i-1$, with 0th link being the world link. The GPLC optimization problem (12) for this chain is

$$\underset{\mathbf{a}_1,\ldots,\mathbf{a}_n,\ddot{q}}{\textbf{minimize}} \quad \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right)^T H_i \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right) \tag{13a}$$

$$\textbf{subject to} \quad \mathbf{a}_i = \mathbf{a}_{i-1} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i},\ i = 1,2,\ldots,n \tag{13b}$$

$$K_n\mathbf{a}_n = \mathbf{k}_n, \quad \mathbf{a}_0 = -\mathbf{a}_{\text{grav}} \tag{13c}$$

where (13b) implicitly encodes joint motion constraints using (11), $\mathbf{a}_{b,i} := \mathbf{v}_i \times S_i\dot{\mathbf{q}}_i$ is the bias acceleration, (13c) encodes the end-effector constraint (a common pattern, e.g., when the end-effector is wiping a table) and the fixed-base constraint, and $\mathbf{a}_{\text{grav}}$ is the acceleration-due-to-gravity vector. The reason for setting $\mathbf{a}_0$ to $-\mathbf{a}_{\text{grav}}$ will be explained in Section III-B2. The parameters in the problem, such as H_i, $\mathbf{f}_i$, $\mathbf{a}_{b,i}$, and S_i are computed using the inputs to the problem, namely, $\mathbf{q_p}$, $\dot{\mathbf{q}}$, $\boldsymbol{\tau}$, and the robot model.

The problem in (13) is algebraically identical to the discrete-time LQR problem: the forward propagation of link acceleration along the kinematic chain [see (13b)] is analogous to the LQR's forward state propagation in time, with $\mathbf{a}_i$ and $\ddot{\mathbf{q}}_i$ corresponding to the LQR's states and controls, respectively.

Remark 4: Either $\mathbf{a_i}$s or $\ddot{\mathbf{q}}$ can be considered the *free* variables in (13) as one can be computed from the other using (13b) because S_i always has full rank [34].

Remark 5: The inertia tensor H_i is symmetric positive definite for all links, therefore, (13) is a strongly convex quadratic program (QP) with a unique solution, when feasible.

Conflicting constraints or unachievable desired accelerations at configuration $\mathbf{q}_p$ can make the QP infeasible.

B. DP Solution

We now solve the optimization problem in (13) using DP by following the textbook LQR derivation [2, Ch. 1]. The recurrence relation constraints in (13b) and the $\mathbf{a}_0 = -\mathbf{a}_{\text{grav}}$ constraint will be eliminated via substitution. However, unlike the textbook version, (13) has a hard "terminal" constraint (due to the end-effector constraint), which cannot be similarly eliminated via substitution. Therefore, we adapt the textbook derivation to instead solve for the primal-dual saddle point of QP's Lagrangian, which includes only the end-effector motion constraint as the joint and fixed-base constraints are eliminated through substitution

$$\mathcal{L}(\ddot{\mathbf{q}}, \boldsymbol{\lambda}) := \sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right)^T H_i \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right) + \boldsymbol{\lambda}^T\left(K_n\mathbf{a}_n - \mathbf{k}_n\right). \tag{14}$$

We define "cost-to-go Lagrangian" as the tail problem consisting of the Lagrangian terms corresponding to the ith link and its descendants

$$V_i(\mathbf{a}_{i-1}, \ddot{\mathbf{q}}_i, \ldots, \ddot{\mathbf{q}}_n, \boldsymbol{\lambda}) : = \sum_{j=i}^{n} \frac{1}{2}\left(\mathbf{a}_j - H_j^{-1}\mathbf{f}_j\right)^T H_j \left(\mathbf{a}_j - H_j^{-1}\mathbf{f}_j\right) + \boldsymbol{\lambda}^T\left(K_n\mathbf{a}_n - \mathbf{k}_n\right).$$

Due to its additive structure, the cost-to-go Lagrangian follows the recurrence relation (after simplifying the quadratic objective and grouping the constant terms)

$$V_i(\mathbf{a}_{i-1}, \ddot{\mathbf{q}}_i, \ldots, \ddot{\mathbf{q}}_n, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T\mathbf{a}_i + V_{i+1}(\mathbf{a}_i, \ddot{\mathbf{q}}_{i+1}, \ldots, \ddot{\mathbf{q}}_n, \boldsymbol{\lambda}) + \text{constant}.$$

When convenient, we will drop constant terms from now on for brevity. The Bellman's recurrence relation [19] for the optimal cost-to-go Lagrangian is

$$V_i^*(\mathbf{a}_{i-1}, \boldsymbol{\lambda}) = \min_{\ddot{\mathbf{q}}_i}\left\{\frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T\mathbf{a}_i + V_{i+1}^*(\mathbf{a}_i, \boldsymbol{\lambda})\right\}. \tag{15}$$

Optimizing the cost-to-go Lagrangian at the end-effector

$$V_n(\mathbf{a}_{n-1}, \ddot{\mathbf{q}}_n, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{a}_n^T H_n \mathbf{a}_n - \mathbf{f}_n^T\mathbf{a}_n + \boldsymbol{\lambda}^T\left(K_n\mathbf{a}_n - \mathbf{k}_n\right) \tag{16}$$

over $\ddot{\mathbf{q}}_n$ gives $V_n^*(\mathbf{a}_{n-1}, \boldsymbol{\lambda})$. To do this, we first substitute $\mathbf{a}_n$ with the acceleration recursion equation in (13b)

$$\begin{aligned} V_n(\mathbf{a}_{n-1}, \ddot{\mathbf{q}}_n, \boldsymbol{\lambda}) &= \frac{1}{2}\left(\mathbf{a}_{n-1} + S_n\ddot{\boldsymbol{q}}_n + \mathbf{a}_{b,n}\right)^T H_n \left(\mathbf{a}_{n-1} + S_n\ddot{\boldsymbol{q}}_n + \mathbf{a}_{b,n}\right) \\ &\quad - \mathbf{f}_n^T\left(\mathbf{a}_{n-1} + S_n\ddot{\boldsymbol{q}}_n + \mathbf{a}_{b,n}\right) \\ &\quad + \boldsymbol{\lambda}^T\left(K_n\left(\mathbf{a}_{n-1} + S_n\ddot{\boldsymbol{q}}_n + \mathbf{a}_{b,n}\right) - \mathbf{k}_n\right). \end{aligned} \tag{17}$$

Then, we collect the linear-quadratic terms in $\ddot{\mathbf{q}}_n$ and solve for the optimal $\ddot{\mathbf{q}}_n^*$, where the quadratic function's gradient is zero

$$\ddot{\mathbf{q}}_n^* = \left(S_n^T H_n S_n\right)^{-1} S_n^T\left\{\mathbf{f}_n - H_n(\mathbf{a}_{n-1} + \mathbf{a}_{b,n}) - K_n^T\boldsymbol{\lambda}\right\}$$

substituting which back in (17) provides $V_n^*(\mathbf{a}_{n-1}, \boldsymbol{\lambda})$, which remains a quadratic form in $\mathbf{a}_{n-1}$ and $\boldsymbol{\lambda}$. Therefore, let us hypothesize that $V_i^*(\mathbf{a}_{i-1}, \boldsymbol{\lambda})$ minimizes the following quadratic form

$$V_i^*(\mathbf{a}_{i-1}, \boldsymbol{\lambda}) = \min_{\ddot{\mathbf{q}}_i}\left\{\frac{1}{2}\mathbf{a}_i^T H_i^A \mathbf{a}_i - \frac{1}{2}\boldsymbol{\lambda}^T L_i^A \boldsymbol{\lambda}\right. \tag{18a}$$

$$+\boldsymbol{\lambda}^T K_i^A \mathbf{a}_i - \mathbf{f}_i^{AT}\mathbf{a}_i + \mathbf{l}_i^T\boldsymbol{\lambda}\Big\} + \text{constant}$$

$$= \min_{\ddot{\mathbf{q}}_i}\Big\{\frac{1}{2}(\mathbf{a}_{i-1} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i})^T H_i^A(\mathbf{a}_{i-1} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i}) - \frac{1}{2}\boldsymbol{\lambda}^T L_i^A\boldsymbol{\lambda} + \boldsymbol{\lambda}^T K_i^A\left(\mathbf{a}_{i-1} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i}\right) - \mathbf{f}_i^{AT}\left(\mathbf{a}_{i-1} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i}\right) + \mathbf{l}_i^T\boldsymbol{\lambda}\Big\} + \text{constant}. \quad (18b)$$

where (18b) is obtained by substituting (13b) in (18a). Optimizing (18b) over $\ddot{\mathbf{q}}_i$ by setting the objective function's gradient to zero gives

$$\ddot{\mathbf{q}}_i^* = D_i^{-1}S_i^T\left\{\mathbf{f}_i^A - H_i^A(\mathbf{a}_{i-1} + \mathbf{a}_{b,i}) - K_i^{AT}\boldsymbol{\lambda}\right\} \quad (19)$$

where $D_i^{-1} := (S_i^T H_i^A S_i)^{-1} \in \mathbb{R}^{n_i\times n_i}$ exists because S_i always has full column rank [34] and H_i^A (which we will show to be the articulated body inertia matrix) is positive definite. Back-substituting $\ddot{\mathbf{q}}_i^*$ from (19) in (18b) gives $V_i^*(\mathbf{a}_{i-1}, \boldsymbol{\lambda})$, substituting which in the Bellman recurrence relation (15) for $V_{i-1}^*(\mathbf{a}_{i-2}, \boldsymbol{\lambda})$ gives the following recursive formulae for the hypothesized quadratic form in (18a)

$$H_{i-1}^A = H_{i-1} + P_i H_i^A \quad (20a)$$

$$\mathbf{f}_{i-1}^A = \mathbf{f}_{i-1} + P_i\left(\mathbf{f}_i^A - H_i^A\mathbf{a}_{b,i}\right) \quad (20b)$$

$$K_{i-1}^{AT} = P_i K_i^{AT} \quad (20c)$$

$$\mathbf{l}_{i-1} = \mathbf{l}_i + K_i^A\left\{\mathbf{a}_{b,i} + S_i D_i^{-1}S_i^T(\mathbf{f}_i^A - H_i^A\mathbf{a}_{b,i})\right\} \quad (20d)$$

$$L_{i-1}^A = L_i^A + K_i^A S_i (D_i)^{-1} S_i^T K_i^{AT} \quad (20e)$$

where $P_i := (I_{6\times 6} - H_i^A S_i (D_i)^{-1} S_i^T) \in \mathbb{R}^{6\times 6}$ is the projection matrix that propagates forces and inertia backward through the ith joint.

The end-effector cost-to-go Lagrangian in (16) conforms to the hypothesized quadratic form in (18a), with $H_n^A = H_n$, $\mathbf{f}_n^A = \mathbf{f}_n$, $K_n^A = K_n$, $\mathbf{l}_n = -\mathbf{k}_n$, and $L_n^A = \mathbf{0}_{n\times n}$ being the starting point of the backward recursion using (20). With this, we can show inductively that the assumed quadratic form validly parameterizes the optimal cost-to-go-Lagrangian.

Performing backward recursion until the root link yields $V_1^*(\mathbf{a}_0, \boldsymbol{\lambda})$'s expression, where the known value of $\mathbf{a}_0 = -\mathbf{a}_{\text{grav}}$ is directly substituted, thereby eliminating all the primal variables of the Lagrangian to obtain the dual function

$$V_0^*(\boldsymbol{\lambda}) = -\frac{1}{2}\boldsymbol{\lambda}^T L_0^A\boldsymbol{\lambda} + \boldsymbol{\lambda}^T\left(\mathbf{l}_0 + K_0^A\mathbf{a}_0\right). \quad (21)$$

Assuming that L_0^A has full rank, the dual function has the unique maximizer as follows:

$$\boldsymbol{\lambda}^* = \left(L_0^A\right)^{-1}\left(\mathbf{l}_0 + K_0^A\mathbf{a}_0\right). \quad (22)$$

The numerical value of $\boldsymbol{\lambda}^*$ computed above enables rolling out the "control policy" in a forward sweep to compute the optimal joint accelerations $\ddot{\mathbf{q}}_i^*$s using (19) and (13b).

1) Details on $\mathbf{f}_i$: $\mathbf{f}_i$ is the resultant of all the nonconstraint forces acting on the ith link, namely, the force due to ith joint torque $\boldsymbol{\tau}_i$, the bias forces, the reaction force from $\boldsymbol{\tau}_{i+1}$, and all the other external forces

$$\mathbf{f}_i = T_i\boldsymbol{\tau}_i - \mathbf{v}_i \times^* H_i\mathbf{v}_i - T_{i+1}\boldsymbol{\tau}_{i+1} + \mathbf{f}_i^{\text{ext}}. \quad (23)$$

Note: The total reaction force on the ith link due to $\boldsymbol{\tau}_{i+1}$, must also include the backward propagation of the force acting on the $i+1$th link due to $\boldsymbol{\tau}_{i+1}$, $T_{i+1}\boldsymbol{\tau}_{i+1}$, using (20b) in addition to the immediate reaction force $-T_{i+1}\boldsymbol{\tau}_{i+1}$

$$\begin{aligned} &-T_{i+1}\boldsymbol{\tau}_{i+1} + P_{i+1}(T_{i+1}\boldsymbol{\tau}_{i+1}) \\ &= -H_{i+1}^A S_{i+1}(D_{i+1})^{-1}S_{i+1}^T T_{i+1}\boldsymbol{\tau}_{i+1} \\ &= -H_{i+1}^A S_{i+1}(D_{i+1})^{-1}\boldsymbol{\tau}_{i+1} \end{aligned} \quad (24)$$

which agrees with the known result on the backward reaction forces applied by joint actuators [1, Eq. 7.20].

2) Including the Effect of Gravity: The straightforward approach to account for gravity is to include the each link's weight in (23), but a more efficient and commonly used trick [48] is to add a gravity field by setting $\mathbf{a}_0 \leftarrow -\mathbf{a}_{\text{grav}}$, where $\mathbf{a}_{\text{grav}}$. Then, $\mathbf{a}_i = -\mathbf{a}_{\text{grav}}$ if the ith link is in equilibrium and $\mathbf{a}_i = 0$ if it is in free fall. This addition of gravitational acceleration to each link's acceleration must also be reflected the acceleration constraints through the update

$$\mathbf{k}_n \leftarrow \mathbf{k}_n - K_n\mathbf{a}_{\text{grav}}.$$

IV. Physical Interpretation

We will now provide the physical interpretation for the backward recursion in (20). This section is involved for readers not familiar with existing propagation-based constrained dynamics literature and may be skipped/skimmed during the first read. P_i is the projection matrix, that propagates $\mathbf{f}_i$ through the ith joint to the $i-1$th link after removing the component that causes the ith joint's motion. It is used in (20b) to propagate the forces backward in the chain. P_i also propagates the inertia of the descendant links through the ith joint in (20a), to compute the well known articulated body inertia H_i^A. Suppose that the ith link was disconnected from its parent link but remained connected to its descendant links, H_i^A would be this link's apparent inertia including the influence of all the descendant links. D_i is the apparent inertia of the ith link along the ith joint, obtained by projecting H_i^A onto the ith joint's motion subspace S_i.

In the absence of end-effector constraints, only (20a) and (20b) need to be computed during the backward recursion and these two formulae are identical to the inertia and force propagation equations in Featherstone's well known ABA [16], which remains the fastest algorithm to compute unconstrained forward dynamics [1]. The PV solver reduces to ABA in the unconstrained setting and an unconstrained LQR-based derivation would essentially be an alternate derivation for the ABA algorithm.

Each row of K_n is the unit spatial force exerted by the end-effector due to the associated constraint, whose magnitude (the unknown Lagrange multipliers) must be solved for. These unit constraint forces are propagated backward in the chain similarly to the nonconstraint forces using the force propagator matrix P_i in (20c). Therefore, $-K_i^{AT}\boldsymbol{\lambda}$ is the force felt at the ith link due to end-effector constraint forces.

Substituting the solution for joint accelerations from (19) into the acceleration recurrence relation in (13b) gives

$$\mathbf{a}_i = P_i^T \left(\mathbf{a}_{i-1} + \mathbf{a}_{b,i}\right) + S_i D_i^{-1} S_i^T \left(\mathbf{f}_i^A - K_i^{AT} \boldsymbol{\lambda}\right) \tag{25}$$

where P_i^T is the projection operation that propagates $\mathbf{a}_{i-1}$ to child link i, after removing $\mathbf{a}_{i-1}$'s acceleration component along S_i. This reveals an interesting symmetric relationship between the forward acceleration propagator P_i^T and the backward force propagator P_i about the ith joint, previously noted in [49]. Let us compose the force propagators to define the extended force propagator [31]

$$P_i^n := P_i P_{i+1} \ldots P_n, \quad \text{and} \quad P_{n+1}^n := I_{6\times 6} \tag{26}$$

that directly propagates end-effector forces to the $i-1$th link. Due to the symmetric relationship, P_i^{nT} propagates accelerations from the $i-1$th link to the end-effector directly. Repeated substitution of (19) for all joints in the acceleration recurrence relation (13b) gives

$$\begin{aligned}\mathbf{a}_n = {} & P_1^{nT} \mathbf{a}_0 + \sum_{i=1}^{n} P_i^{nT} \mathbf{a}_{b,i} \\ & + \sum_{i=1}^{n} \left\{ P_{i+1}^{nT} S_i D_i^{-1} S_i^T \left(\mathbf{f}_i^A - K_i^{AT} \boldsymbol{\lambda}\right) \right\}. \end{aligned} \tag{27}$$

From the constraint propagation equations in (20c), one can easily verify that

$$K_i^A = K_n P_{i+1}^{nT}. \tag{28}$$

We remind readers that the end-effector acceleration constraint is $K_n \mathbf{a}_n + \mathbf{l}_n = 0$. Let us call $K_n \mathbf{a}_n$, *constraint acceleration* (because it is the end-effector acceleration along the constrained direction) and $-\mathbf{l}_n$ the desired constraint acceleration. Substituting $\mathbf{a}_n$ from (27) in the acceleration constraint equation and simplifying using (28) gives

$$\begin{aligned} K_n \mathbf{a}_n + \mathbf{l}_n = {} & K_0^A \mathbf{a}_0 + \sum_{i=1}^{n} K_i^A P_i^T \mathbf{a}_{b,i} \\ & + \sum_{i=i}^{n} \{ K_i^A S_i D_i^{-1} S_i^T (\mathbf{f}_i^A - K_i^{AT} \boldsymbol{\lambda}) \} + \mathbf{l}_n = 0. \end{aligned} \tag{29a}$$

$K_0^A \mathbf{a}_0$ is the constraint acceleration due to the known fixed-base acceleration. Collecting the terms not containing the unknown $\boldsymbol{\lambda}$ in the previous equation and comparing with backward recursion in (20d), one can verify that

$$\mathbf{l}_{i-1}^A = \sum_{k=i}^{n} \left\{ K_k^A P_k^T \mathbf{a}_{b,k} + \left(K_k^A S_k D_k^{-1} S_k^T \mathbf{f}_k^A\right) \right\} + \mathbf{l}_n \tag{30}$$

recursively computes constraint acceleration caused by the bias accelerations, bias forces, joint torques, and external forces up to the ith joint and updates the desired constraint acceleration that must be supplied by the unknown constraint forces. Comparing (20e) and (29), we see (20e) recursively computes the $\boldsymbol{\lambda}$-dependent terms in (29) with

$$L_{i-1}^A = \sum_{k=i}^{n} K_k^A S_k D_k^{-1} S_k^T K_k^A \tag{31}$$

where the jth column of L_{i-1}^A is the constraint accelerations caused by a unit magnitude jth constraint force due to motions along the joints from the nth joint back up to the ith joint in the chain. L_0^A represents the inertial coupling between constraints considering the whole tree's motion, providing intuition for why L_0^A must be the inverse OSIM Λ^{-1}, which was previously defined in the joint-space in (5).

$$\Lambda^{-1} = J M^{-1} J^T = K_n \left(J_n M^{-1} J_n^T\right) K_n^T \tag{32}$$

where $J_n M^{-1} J_n^T$ maps any force acting on the end-effector $\mathbf{f}_n$ to end-effector acceleration caused due to this force

$$\mathbf{a}_n^f := \left(J_n M^{-1} J_n^T\right) \mathbf{f}_n. \tag{33}$$

From (27), we collect all the terms depending on $\mathbf{f_n}$ that cause end-effector acceleration (remember that $\mathbf{f}_i^A$ also depends on $\mathbf{f_n}$ because of inward force recursion) to get

$$\mathbf{a}_n^f = \left\{ \sum_{i=1}^{n} P_{i+1}^{nT} S_i D_i^{-1} S_i^T P_{i+1}^n \right\} \mathbf{f}_n. \tag{34}$$

In (33) and (34) have linear mappings from $\mathbf{f}_n$ to $\mathbf{a}_n^f$, where $\mathbf{f}_n$ is free to take on any value in $\mathbb{R}^6$ and the linear mappings depend only on $\mathbf{q_p}$. Thus, it must be that $J_n M^{-1} J_n^T = \sum_{i=1}^{n} P_{i+1}^{nT} S_i D_i^{-1} S_i^T P_{i+1}^n$. Pre and postmultiplying this equality with K_n and K_n^T, we get

$$K_n \left(J_n M^{-1} J_n^T\right) K_n^T = \sum_{i=1}^{n} K_n P_{i+1}^{nT} S_i D_i^{-1} S_i^T P_{i+1}^n K_n^T \tag{35}$$

where using (32), (28), and (31), we get $\Lambda^{-1} = L_0^A$. The physical interpretation presented here is essentially the argument used in [33] to derive their constrained dynamics solver for kinematic loops, which we refer readers to for more insight especially related to the effect of internal kinematic loops. Compared with [33], our derivation is mathematical using the DP algorithm and does not require readers to possess physical insight. The physical interpretation provided here is only a post hoc explanation. However, the derivation in [33] does not assume prior optimization knowledge and may be more accessible to some readers, especially for those familiar with Featherstone's ABA algorithm derivation [16], because [33] is a natural extension of [16] that follows a similar variable elimination approach.

V. Extension to Trees With Floating-Base

We now extend the original PV solver, that only dealt with end-effector constrained fixed-base kinematic chains, to kinematic trees with possibly a floating-base, and possibly motion constraints on any link. We first modify the problem formulation to allow kinematic trees in Section V-A, solve it using DP in Section V-B and finally present the algorithm and analyze the computational complexity in Section V-C.

A. Problem Formulation

The GPLC optimization problem for a given tree is

$$\underset{a,\ddot{q}}{\textbf{minimize}} \quad \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right)^T H_i \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right) \tag{36a}$$

$$\text{subjectto} \quad \mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i},\ i = 1, 2, \ldots, n \tag{36b}$$

$$K_i\mathbf{a}_i = \mathbf{k}_i, \quad i = 1, \ldots, n \tag{36c}$$

where, $\pi(j)$ and $\gamma(j)$ are the parent link and the set of children for any given link j, respectively, as explained in Section II-D. Compared with the problem in (13), the recurrence relation in (36b) is indexed differently due to the tree structure, and any link's motion can be constrained in (36c). It is easily verifiable that the problem remains a strongly convex QP, but it is no longer analogous to a simple discrete-time LQR problem. Instead, this problem shares its structure with scenario-trees from control of systems with dynamics uncertainty [50]. However, the DP approach remains applicable and will provide a tree-structured Riccati recursion [51].

B. DP Solution

Similarly to kinematic chains, we apply DP on the Lagrangian of the optimization problem in (36)

$$\mathcal{L}(\ddot{\mathbf{q}}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_n) = \sum_{i=1}^{n} \left\{ \frac{1}{2} \left(\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T \mathbf{a}_i\right) + \boldsymbol{\lambda}_i^T \left(K_i\mathbf{a}_i - \mathbf{k}_i\right) \right\}. \tag{37}$$

For notational simplicity in the upcoming derivation, let us define $\boldsymbol{\lambda}_i^A := [\boldsymbol{\lambda}_i^T, \boldsymbol{\lambda}_{\gamma(i)_1}^{AT}, \boldsymbol{\lambda}_{\gamma(i)_2}^{AT} \ldots \boldsymbol{\lambda}_{\gamma(i)_{\mathcal{C}(i)}}^{AT}]^T$ as the concatenation of the multipliers associated with constraints on the ith link and its descendants, where $\mathcal{C}(i)$ is the cardinality of the set $\gamma(i)$. Analogously to the (15), the Bellman recurrence for the optimal cost-to-go Lagrangian for the kinematic tree is as follows:

$$V_i^*(\mathbf{a}_{\pi(i)}, \boldsymbol{\lambda}^A) = \min_{\ddot{q}_i} \left\{ \frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{f}_i^T\mathbf{a}_i + \boldsymbol{\lambda}_i^T(K_i\mathbf{a}_i - \mathbf{k}_i) + \sum_{j\in\gamma(i)} V_j^*(\mathbf{a}_i, \boldsymbol{\lambda}_j^A) \right\} + \text{constant}. \tag{38}$$

Similarly to (18a), let us hypothesize that the optimal cost-to-go Lagrangian has the quadratic form

$$V_i^*(\mathbf{a}_{\pi(i)}, \boldsymbol{\lambda}_i^A) = \min_{\ddot{\mathbf{q}}_i} \left\{ \frac{1}{2}\mathbf{a}_i^T H_i^A \mathbf{a}_i - \frac{1}{2}\boldsymbol{\lambda}_i^{AT} L_i^A \boldsymbol{\lambda}_i^A + \boldsymbol{\lambda}_i^{AT} K_i^A \mathbf{a}_i - \mathbf{f}_i^{AT}\mathbf{a}_i + \mathbf{l}_i^T\boldsymbol{\lambda}_i^A \right\} + \text{constant}. \tag{39}$$

Substituting $\mathbf{a}_i$ above using (36b) gives

$$V_i^*(\mathbf{a}_{\pi(i)}, \boldsymbol{\lambda}_i^A) = \min_{\ddot{\mathbf{q}}_i} \left\{ \frac{1}{2}(\mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i})^T H_i^A (\mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i}) - \frac{1}{2}\boldsymbol{\lambda}_i^{AT} L_i^A \boldsymbol{\lambda}_i^A + \boldsymbol{\lambda}_i^{AT} K_i^A (\mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i}) - \mathbf{f}_i^{AT}(\mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i}) + \mathbf{l}_i^T\boldsymbol{\lambda}_i^A \right\} + \text{constant}. \tag{40}$$

Optimizing this function for optimal $\ddot{\mathbf{q}}_i$ gives

$$\ddot{\mathbf{q}}_i^* = (D_i)^{-1} S_i^T \left\{ \mathbf{f}_i^A - H_i^A(\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i}) - K_i^{AT}\boldsymbol{\lambda}_i^A \right\} \tag{41}$$

substituting which back into (40) gives $V_i^*(\mathbf{a}_{\pi(i)}, \boldsymbol{\lambda}_i^A)$.

Substituting the expression $V_k^*(\mathbf{a}_i, \boldsymbol{\lambda}_k^A)$, thus, computed for all $k \in \gamma(i)$ in Bellman recurrence relation (38) confirms that the optimal cost-to-go function has the quadratic form hypothesized in (39) for link i if the hypothesis holds for all the children links $k \in \gamma(i)$. The quadratic form for the ith link is given by the recursive equations

$$H_i^A = H_i + \sum_{k\in\gamma(i)} P_k H_k^A \tag{42a}$$

$$\mathbf{f}_i^A = \mathbf{f}_i + \sum_{k\in\gamma(i)} P_k \left(\mathbf{f}_k^A - H_k^A\mathbf{a}_{b,k}\right) \tag{42b}$$

$$K_i^A = \begin{bmatrix} K_i \\ \vdots \\ K_k^A P_k^T \\ \vdots \end{bmatrix} \tag{42c}$$

$$\mathbf{l}_i = \begin{bmatrix} -\mathbf{k}_i \\ \vdots \\ \mathbf{l}_k + K_k^A \left\{\mathbf{a}_{b,k} + S_k D_k^{-1} S_k^T \left(\mathbf{f}_k^A - H_k^A\mathbf{a}_{b,k}\right)\right\} \\ \vdots \end{bmatrix} \tag{42d}$$

$$L_i^A = \begin{bmatrix} \mathbf{0}_{m_i\times m_i} & & \\ & \ddots & \\ & & L_k^A + K_k^A S_k (D_k)^{-1} S_k^T K_k^{AT} \\ & & & \ddots \end{bmatrix}. \tag{42e}$$

The cost-to-go Lagrangian at any leaf node conforms to the assumed quadratic form with, $H_j^A = H_j$, $L_j^A = \mathbf{0}_{m_j\times m_j}$, $K_j^A = K_j$, $\mathbf{f}_j^A = \mathbf{f}_j$, and $\mathbf{l}_j = -\mathbf{k}_j$ for all j that are leaf links. Therefore, it can be shown again inductively that the equations assumed in (39) correctly model the cost-to-go function.

For a fixed-base robot, the backward recursion is performed until the base link 0, and the known fixed-base acceleration is substituted to obtain the dual function, which is maximized to compute the optimal dual variables $\boldsymbol{\lambda}_0^{A*}$ (assuming that L_0^A has full rank) analogously to (22)

$$\boldsymbol{\lambda}_0^{A*} = \left(L_0^A\right)^{-1} \left(\mathbf{l}_0 + K_0^A\mathbf{a}_0\right). \tag{43}$$

For a floating-base robot, the backward sweep is conducted until the floating-base link b, from where the optimal base

acceleration and the dual variables are the saddle point of the optimal cost-to-go Lagrangian at the floating-base

$$\boldsymbol{\lambda_b}^{A*}, \mathbf{a}_b^* = \underset{\boldsymbol{\lambda}_b^A}{\mathbf{argmax}} \left\{ \min_{\mathbf{a_b}} \left(\frac{1}{2}\mathbf{a}_b^T H_b^A \mathbf{a}_b - \frac{1}{2}\boldsymbol{\lambda}_b^{AT} L_b^A \boldsymbol{\lambda}_b^A + \boldsymbol{\lambda}_b^{AT} K_b^A \mathbf{a}_b - \mathbf{f}_b^{AT} \mathbf{a}_b + \mathbf{l}_b^T \boldsymbol{\lambda}_b^A \right) \right\}. \tag{44}$$

The stationary gradient condition of the first-order necessary KKT conditions provides the simultaneous linear equations as follows:

$$\mathbf{a}_b^* = \left(H_b^A\right)^{-1} \left(\mathbf{f}_b^A - K_b^{AT} \boldsymbol{\lambda}_b^{A*}\right) \tag{45}$$

$$\boldsymbol{\lambda}_b^{A*} = \left(L_b^A\right)^{-1} \left(K_b^A \mathbf{a}_b^* + \mathbf{l}_b\right). \tag{46}$$

We can substitute $\mathbf{a}_b^*$ from (45) in (46) to get

$$\boldsymbol{\lambda}_b^{A*} = \left(L_b^A + K_b^A \left(H_b^A\right)^{-1} K_b^{AT}\right)^{-1} \left(K_b^A \left(H_b^A\right)^{-1} \mathbf{f}_b^A + \mathbf{l}_b\right) \tag{47}$$

and the optimal base acceleration is, then, recovered using (45) and the inverse OSIM matrix is

$$L_0^A = \left(L_b^A + K_b^A \left(H_b^A\right)^{-1} K_b^{AT}\right) \tag{48}$$

which is no different from performing the usual backward recursion at the free-joint b with, $S_b = I_{6\times 6}$ as the free joint is allowed to move in all directions.

Alternately, if L_b^A is invertible one can also substitute the expression for $\boldsymbol{\lambda}_b^{A*}$ from (46) in to (45) to get

$$\mathbf{a}_b^* = \left(H_b^A + K_b^{AT} \left(L_b^A\right)^{-1} K_b^A\right)^{-1} \left(\mathbf{f}_b^A - K_b^{AT} \left(L_b^A\right)^{-1} \mathbf{l}_b\right) \tag{49}$$

and optimal Lagrange multipliers can then be recovered using (46). The accelerations of the rest of the segments are then computed in the second forward sweep (rollout). The choice computing (47) or (49) can significantly impact the computational efficiency of the algorithm depending on the branching structure and the number of constraints.

Suppose that kinematic tree branches at the floating-base, then, L_b^A has a block-diagonal structure, because the L_i^A terms from different branches occupy their respective diagonal block in (42e). Factorizing or inverting $(L_b^A)^{-1}$ is easier due to this block-diagonal structure. Then computing (49) requires solving a small linear system of fixed size 6×6, which makes using (49) a superior choice in this case. On the other hand, computing (48) performs a dense $m \times m$ update to L_b^A, which destroys the block-diagonal sparsity pattern and, then, requires solving a dense linear system of size $m \times m$.

C. Algorithm

Algorithm 1 presents the PV solver for kinematic trees with floating-base. Let $\mathcal{S}$ be an ordered list of all the links in the kinematic tree, such that i precedes j in the list if ith link is the jth link's ancestor. Let $\mathcal{S}_r$ be the reversed list of $\mathcal{S}$. In Algorithm 1, we use (49) instead of (47).

Algorithm 1: PV Solver for Kinematic Trees With Floating-Base.

Require: $\mathbf{q}^{\mathrm{p}}$, $\dot{\mathbf{q}}$, $\boldsymbol{\tau}$, K_is, $\mathbf{k}_i$s, $X_{\{b\}}$, $\mathbf{v}_{\{b\}}$, robot model

First forward sweep

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

Backward sweep

7: **for** i in $\mathcal{S}_r$
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: $\mathbf{f}_{\pi(i)}^A \leftarrow \mathbf{f}_{\pi(i)}^A + P_i(\mathbf{f}_i^A - H_i^A \mathbf{a}_{b,i})$
10: $H_{\pi(i)}^A \leftarrow H_{\pi(i)}^A + P_i H_i^A$
11: $K_{\pi(i)}^A \leftarrow \begin{bmatrix} K_{\pi(i)}^A \\ K_i^A P_i^T \end{bmatrix}$
12: $L_{\pi(i)}^A \leftarrow \begin{bmatrix} L_{\pi(i)}^A & \\ & L_i^A + K_i^A S_i (D_i)^{-1} S_i^T K_i^{AT} \end{bmatrix}$
13: $\mathbf{l}_{\pi(i)} \leftarrow \begin{bmatrix} \mathbf{l}_{\pi(i)} \\ \mathbf{l}_i + K_i^A \{\mathbf{a}_{b,i} + S_i D_i^{-1} S_i^T (\mathbf{f}_i^A - H_i^A \mathbf{a}_{b,i})\} \end{bmatrix}$
14: $\mathbf{a}_b^* = (H_b^A + K_b^{AT}(L_b^A)^{-1} K_b^A)^{-1} (\mathbf{f}_b^A - K_b^{AT}(L_b^A)^{-1} \mathbf{l}_b)$
15: $\boldsymbol{\lambda}_b^{A*} = (L_b^A)^{-1}(K_b^A \mathbf{a}_b^* + \mathbf{l}_b)$

Second forward sweep (roll-out)

16: **For** i in $\mathcal{S}$
17: $\ddot{\mathbf{q}}_i^* = (S_i^T H_i^A S_i)^{-1} S_i^T \{\mathbf{f}_i^A - H_i^A (\mathbf{a}_{\pi(i)} + \mathbf{a}_{b,i}) - K_i^{AT} \boldsymbol{\lambda}_i^{A*}\}$
18: $\mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i \ddot{\mathbf{q}}_i^* + \mathbf{a}_{b,i}$

1) Computational Complexity: We now analyze the worst-case computational complexity of Algorithm 1. The computations in Lines 2, 3, 4, 5, 7, 8, 9, and 17 each require fixed number of operations at every joint and requires $O(n)$ operations in total. The Lines 10, 12, and 16 require $O(m)$ operations per at most d executions, where d is the depth of the tree requiring $O(md)$ operations. Line 11 needs $O(m^2)$ operations per joint and $O(m^2\, d)$ operations in total. Factorizing L_b^A in Line 13 has the worst case complexity of $O(m^3)$. Aggregating these terms, the algorithm requires $O(n + m^2\, d + m^3)$ operations in the worst case.

Best case complexity: The computational complexity is significantly better than the worst case complexity for favorable tree structures and constraints. Suppose that the branching occurs at the (floating) base link and there is one end-effector (a constrained link with at most 6-D constraint) per branch. Quadrupeds and humanoid robots often have this structure. Let r be the number of branches and d be the length of the longest branch. Line 11 is executed d times for r branches leading to $O(dr)$ operations. Similarly factorizing the block-diagonal matrix L_b^A needs $O(r)$ operations for each block of size at most

6×6. As $m = O(r)$, the total complexity of the constrained dynamics for this tree is $O(n + md + m)$. The equality of Λ^{-1} and L_0 established in Section IV can be repeated for kinematic trees as well using identical arguments and, hence, will be skipped for the sake of brevity.

VI. Soft Gauss' Principle

We have considered only hard motion constraints so far, but it is also conceivable to relax these motion constraint through a penalty method and solve this easier problem, which is further always feasible even if the constraints are linearly dependent. This is precisely the approach taken in the MuJoCo toolbox [7], [42], a popular rigid body dynamics simulator using the so-called "soft Gauss' principle", where the hard motion constraints are relaxed through a quadratic penalty

$$\underset{a,\ddot{q}}{\textbf{minimize}} \quad \sum_{i=1}^{n} \frac{1}{2} \Big\{ \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right)^T H_i \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right) + \left(K_i\mathbf{a}_i - \mathbf{k}_i\right)^T R_i^{-1} \left(K_i\mathbf{a}_i - \mathbf{k}_i\right) \Big\} \tag{50a}$$

$$\text{subject to} \quad \mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i},\ i = 1, 2, \ldots, n \tag{50b}$$

where $R_i \in \mathbb{R}^{m_i \times m_i}$ is a diagonal positive definite matrix. After expanding the objective function in (50a), collecting the quadratic and linear terms and ignoring the constant terms, we get an equivalent optimization problem

$$\underset{a,\ddot{q}}{\textbf{minimize}} \quad \sum_{i=1}^{n} \Big\{ \frac{1}{2}\mathbf{a}_i^T \left(H_i + K_i^T R_i^{-1} K_i\right) \mathbf{a}_i - \left(\mathbf{f}_i + K_i R_i^{-1}\mathbf{k}_i\right)^T \mathbf{a}_i \Big\} + \text{const} \tag{51a}$$

$$\text{subject to} \quad \mathbf{a}_i = \mathbf{a}_{\pi(i)} + S_i\ddot{\mathbf{q}}_i + \mathbf{a}_{b,i},\ i = 1, 2, \ldots, n \tag{51b}$$

which is a special case of the kinematic tree optimization problem in (36), but without motion constraints (apart from the joint constraints in (50b), which will be eliminated through substitution) and with the modified H_i and f_i terms

$$H_i \leftarrow H_i + K_i^T R_i^{-1} K_i; \quad \mathbf{f}_i \leftarrow \mathbf{f}_i + K_i R_i^{-1}\mathbf{k}_i. \tag{52}$$

As there are no motion constraints, the L_i^A, $\mathbf{l}_i$, and K_i^A terms are not computed for the soft Gauss' problem, for which the Algorithm 1 reduces simply to ABA with the update in (52).

A. Computational Complexity

The ABA has $O(n)$ complexity whereas the inertia and forces updates in (52) require $O(m)$ operations. Therefore, the total computational complexity for solving the soft Gauss' principle is $O(m + n)$. The state-of-the-art simulator MuJoCo solves the problem in the joint-space resulting in a significantly higher computational complexity of $O(nd^2 + m^2\, d + d^2\, m)$. It uses the composite rigid body algorithm algorithm [52, Method 3] to compute the JSIM and factorizes it, which has worst-case complexity of $O(nd^2)$. It considers constraints by modifying the JSIM [7, Eq. 7] analogously to our inertia update in (52) and solves this updated inertia matrix using the matrix inversion lemma (MIL) accounting for the additional terms in the complexity.

Algorithm 2: PV-OSIM Algorithm.

Require $\mathbf{q}^{\mathrm{P}}$, K_is, robot model

First forward sweep

1: **for** i in $\mathcal{S}$ **do**
2: $\quad X_{\{i\}} = X_{\{\pi(i)\}}{}^{\{\pi(i)\}} X_{\{i'\}}{}^{\{i'\}} X_{\{i\}}$
3: $\quad K_i^A \leftarrow K_i$; $L_i^A \leftarrow \mathbf{0}_{m_i \times m_i}$ $H_i^A \leftarrow H_i$;

Backward sweep

4: **for** i in $\mathcal{S}_r$ **do**
5: $\quad D_i = S_i^T H_i^A S_i$; $P_i = (I_{6\times6} - H_i^A S_i (D_i)^{-1} S_i^T)$
6: $\quad H_{\pi(i)}^A \leftarrow H_{\pi(i)}^A + P_i H_i^A$
7: $\quad K_{\pi(i)}^A \leftarrow \begin{bmatrix} K_{\pi(i)}^A \\ K_i^A P_i^T \end{bmatrix}$
8: $\quad L_{\pi(i)}^A \leftarrow \begin{bmatrix} L_{\pi(i)}^A & \\ & L_i^A + K_i^A S_i (D_i)^{-1} S_i^T K_i^{AT} \end{bmatrix}$
9: **if** floating-base? **then**
10: $\quad L_0^A = (L_b^A + K_b^A (H_b^A)^{-1} K_b^{AT})$
11: $\Lambda = (L_0^A)^{-1}$

VII. $O(n)$ Algorithm for OSIM

The OSIM itself is an important expression in many rigid-body simulators in both the robotics and the computer graphics (where its inverse is known as the Delassus operator) communities. It also has applications in constrained inverse dynamics [53] and dynamically-consistent nullspace projection in prioritized torque control [54]. OSIM is particularly useful for resolving inequality constraints (also called unilateral constraints), because an inequality constraint becoming inactive can be easily handled by removing the corresponding row and column of the inverse OSIM and efficiently updating the factorization [55]. Therefore, we isolate the OSIM computations in the PV solver and present a standalone algorithm. Further, we propose an at-best structure exploitation for floating-base robots that avoids factorizing the dense inverse OSIM, which all the existing approaches perform, to the best of the authors' knowledge. Finally, we end the section with a qualitative comparison of the proposed algorithm with the existing $O(n)$ complexity OSIM solvers KJR [28] and EFPA [31].

A. PV-OSIM Algorithm

Algorithm 2 lists the PV solver computations necessary for the OSIM.

B. PV-OSIM-Fast for Floating-Base Robots

For floating-base trees with branching at the base link, L_b^A has block diagonal structure. This sparsity structure is lost in the update in Line 10 in Algorithm 2 (48) by adding a dense matrix to L_b^A. The inverse OSIM (and the OSIM) is a dense matrix for floating-base robots because the constraints on different branches are coupled through the floating-base. All existing

approaches, that we know of, compute this dense inverse OSIM and factorize it, which scales poorly in the presence of many constraints. We propose to avoid this by exploiting the structure of the update in (48).

The update to L_b^A in (48) is structurally a symmetric rank-6 update. If we assume that L_b^A is invertible, which is a reasonable assumption for floating-base robots like humanoids and quadrupeds during operation, the MIL [56] can be used to factorize L_0^A without having to explicitly construct this dense matrix. The MIL states

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U\left(C^{-1}+VA^{-1}U\right)^{-1}VA^{-1} \tag{53}$$

applying which to solve (48) yields

$$\begin{aligned}\left(L_0^A\right)^{-1} &= \left(L_b^A\right)^{-1} - \left(L_b^A\right)^{-1} K_b^A \left\{H_b^A \right. \\ &\quad \left. + K_b^{AT}\left(L_b^A\right)^{-1}K_b^A\right\}^{-1} K_b^{AT}\left(L_b^A\right)^{-1} \qquad (54)\\ &= \Lambda_b - Ł_K\left\{H_b^A + K_b^{AT}Ł_K\right\}^{-1}Ł_K^T \qquad (55)\end{aligned}$$

where $\Lambda_b := (L_b^A)^{-1}$ (easy to compute because of its block diagonal structure, which is retained even after inversion) and $Ł_K := \Lambda_b K_b^A$. Please note that the RHS of the above equation is not evaluated to get the $(L_0^A)^{-1}$ matrix as that would destroy sparsity. Instead, the RHS is meant to be directly multiplied with vectors, similarly to how solving a linear system involves factorization and not matrix inversion.

1) Computational Complexity of PV-OSIM-Fast: The original PV-OSIM algorithm, computes and factorizes the dense L_0^A, which requires $O(m^3)$ operations. In contrast, the structure exploiting method computes Λ_b, which requires $O(\frac{m^3}{r^2})$ operations, and L_K, which requires $O(\frac{m^2}{r})$ operations, bringing the total complexity to $O(\frac{m^3}{r^2})$, where we have assumed for simplicity of analysis that the m constraints are equally distributed among the r branches. Thus, the proposed algorithm in this section can provide a significant speed-up for factorizing the inverse OSIM of floating-base robots with a favorable branching structure compared with the existing approaches that all solve dense linear systems.

1) Limitation of PV-OSIM-fast: Strictly speaking, PV-OSIM-fast is applicable in a subset of the cases where the regular PV-OSIM is applicable because of its assumption that L_b^A is invertible. It is possible that L_b^A is not invertible, but L_0^A is invertible due to the addition of symmetric rank-6 matrix in (48). This situation may occur if there is a high-dimensional constraint applied on a link close to the base link or if the robot reaches a kinematically singular configuration.

C. Comparison With Existing $O(n)$ OSIM Algorithms

We now compare the PV-OSIM algorithm with the existing recursive $O(n)$ algorithms: the KJR algorithm [22], [28], whose optimized version was presented in [32], and the EFPA [31]. The three algorithms share the main idea of propagating the inverse inertia matrices, but differ significantly in the details. The primary structural difference of the PV-OSIM is that it computes the inverse OSIM in two sweeps, whereas both KJR and EFPA require three sweeps.

This difference arises because PV-OSIM computes inverse inertia due to the motion of the ith joint and its descendants directly in the *constraint space* L_i^A during the backward sweep, using the EFP to propagate constraint forces to a joint and the constraint accelerations back to the constrained link. However, both KJR and EFPA first compute the articulated body inertia in a backward sweep and, then, compute the spatial inverse inertia matrices of size 6×6 for all the necessary links in a forward sweep, which is avoided in the PV-OSIM. Propagating spatial inverse inertia matrices is a particularly expensive operation since they need to be transformed from one link's frame to another's (because dynamics algorithms are efficiently implemented in the link frame) in KJR and EFPA. This transformation is not required in PV-OSIM because the inverse inertia is directly computed in the constraint space. Then, KJR and EFPA compute the relative inverse inertia (essentially the matrix that maps forces on one link to the accelerations caused on another link) between every pair of links that are constrained. KJR performs computation inefficiently by propagating the relative spatial inverse inertia matrices through the path connecting two constrained links for every possible pair of constrained links. EFPA computes the relative inverse inertia matrices more efficiently by directly transmitting the constraint forces and accelerations between constrained links through a common ancestor link using EFP. Finally, after all these inverse inertia matrices are computed, EFPA and KJR project them to the constraint space to get the inverse OSIM.

Thus, the PV-OSIM appears to exploit the structure of the problem better by using one less sweep to compute the inverse OSIM and its computational performance relative to existing OSIM algorithms will be benchmarked in Section IX-B. It must be noted that despite performing some extra computations, the EFPA algorithm has a lower order computational complexity of $O(n+md+m^2)$ compared with the $O(n+m^2\,d+m^2)$ complexity of the PV-OSIM for computing the inverse OSIM. Therefore, for kinematic trees of high depth and many constraints, we can expect the EFPA algorithm to be faster than the PV-OSIM, which we test in Section IX-B.

Also, note that the derivation of KJR or EFPA is complex and requires significant knowledge of and insight into efficient dynamics algorithms literature, whereas the PV-OSIM derivation is relatively simpler and self-contained as we are able to derive it from first principles (Gauss' principle) within this article. Moreover, all the existing approaches compute and factorize the dense inverse OSIM matrix for floating-base robots, which the PV-OSIM-fast algorithm in Section VII-B avoids.

VIII. Early Multiplier Resolution

The original PV solver first eliminates the primal variables, recursively computes the inverse OSIM, and factorizes it, which results in a worst case $O(n+m^2\,d+m^3)$ complexity. This can get particularly expensive when $m \sim O(n)$. However, if computing the OSIM is not required (for some other purpose during control or simulation), we can generalize the elimination ordering by aggressively eliminating dual variables earlier during the backward sweep to obtain an algorithm with an

improved complexity of only $O(n+m)$. We now derive this algorithm by adapting our original PV solver derivation. This early elimination idea was already partly introduced in (49), when we eliminated the dual variables just before eliminating $\mathbf{a}_b$, and will be further developed now. A form of early elimination is also proposed in [33], where they eliminate the constraint forces of an internal kinematic loop as soon as all the link accelerations within that loop are eliminated.

According to Bellman's principle of optimality [19], the solution to an optimization problem also optimizes its tail subproblem. Hence, for the tail subproblem at the ith link

$$\boldsymbol{\lambda}_i^{A*}(\mathbf{a}_{\pi(i)}) = \underset{\boldsymbol{\lambda}^A}{\mathbf{argmax}} \;\; V_i^*\left(\mathbf{a}_{\pi(i)}, \boldsymbol{\lambda}_i^A\right). \tag{56}$$

The objective function above is of the form in (39) and is guaranteed to be bounded above and have a solution only when L_i^A has full rank. There is a rank-n_i update to L_i^A at every ith joint during the backward recursion [see (42e)]

$$L_i^A \leftarrow L_i^A + K_i^A S_i (D_i)^{-1} S_i^T K_i^{AT}. \tag{57}$$

Substituting the SVD [45] of L_i^A in (39) gives

$$\begin{aligned}\boldsymbol{\lambda}_i^{A*} = \underset{\boldsymbol{\lambda}^A}{\mathbf{argmax}} \Bigg\{ & -\frac{1}{2}\boldsymbol{\lambda}_i^{AT}\begin{bmatrix}U_i^1 & U_i^2\end{bmatrix}\begin{bmatrix}\Sigma_i & \\ & 0\end{bmatrix}\begin{bmatrix}U_i^{1\,T} \\ U_i^{2\,T}\end{bmatrix}\boldsymbol{\lambda}_i^A \\ & + \mathbf{a}_i^T K_i^{AT}\begin{bmatrix}U_i^1 & U_i^2\end{bmatrix}\begin{bmatrix}U_i^{1\,T} \\ U_i^{2\,T}\end{bmatrix}\boldsymbol{\lambda}_i^A \\ & + \mathbf{l}_i^T\begin{bmatrix}U_i^1 & U_i^2\end{bmatrix}\begin{bmatrix}U_i^{1\,T} \\ U_i^{2\,T}\end{bmatrix}\boldsymbol{\lambda}_i^A \Bigg\} + \text{constant}\end{aligned} \tag{58}$$

where $\Sigma_i \in \mathbb{R}^{m_{ir}\times m_{ir}}$ is the diagonal matrix of the positive singular values, $U_i^1 \in \mathbb{R}^{m_{if}\times m_{ir}}$, and $U_i^2 \in \mathbb{R}^{(m_{if})\times(m_{if}-m_{ir})}$ are the singular vectors corresponding to the positive and zero singular values of L_i^A, respectively, m_{ir} and m_{if} are the rank and the size of L_i^A, respectively. The left and right singular vectors are equal because L_i^A is symmetric. Moreover, the singular vectors are orthonormal

$$\begin{bmatrix}U_i^1 & U_i^2\end{bmatrix}\begin{bmatrix}U_i^{1\,T} \\ U_i^{2\,T}\end{bmatrix} = I_{m_{if}\times m_{if}} \tag{59}$$

which we use to project $\boldsymbol{\lambda}_i^A$, K_i^A, and $\mathbf{l}_i$ to two mutually orthogonal linear bases

$$\begin{aligned}\tilde{\boldsymbol{\lambda}}_i^A &= U_i^{1\,T}\boldsymbol{\lambda}_i^A, \quad \hat{\boldsymbol{\lambda}}_i^A = U_i^{2\,T}\boldsymbol{\lambda}_i^A, \\ \tilde{K}_i^A &= U_i^{1\,T}K_i^A, \quad \hat{K}_i^A = U_i^{2\,T}K_i^A, \\ \tilde{\mathbf{l}}_i &= U_i^{1\,T}\mathbf{l}_i, \quad \hat{\mathbf{l}}_i = U_i^{2\,T}\mathbf{l}_i,\end{aligned} \tag{60}$$

where $\tilde{\boldsymbol{\lambda}}_i^A \in \mathbb{R}^{m_{ir}}$, $\tilde{K}_i^A \in \mathbb{R}^{m_{ir}\times 6}$, $\tilde{\mathbf{l}}_i \in \mathbb{R}^{m_{ir}}$, and $\hat{\boldsymbol{\lambda}}_i^A \in \mathbb{R}^{(m_{if}-m_{ir})}$, $\hat{K}_i^A \in \mathbb{R}^{(m_{if}-m_{ir})\times 6}$, $\hat{\mathbf{l}}_i \in \mathbb{R}^{(m_{if}-m_{ir})}$ are the components of $\boldsymbol{\lambda}_i^A$, K_i^A, and $\mathbf{l}_i$ in the basis spanned by the singular vectors U_i^1 and U_i^2, respectively. Using these quantities, the optimization problem in (58) can be decoupled into a separate optimization problem and a dual feasibility condition along the columnspace and nullspace of L_i^A, respectively,

$$\tilde{\boldsymbol{\lambda}}_i^A = \underset{\tilde{\boldsymbol{\lambda}}_i^A}{\mathbf{argmax}} \left\{ -\frac{1}{2}\tilde{\boldsymbol{\lambda}}_i^{AT}\Sigma_i\tilde{\boldsymbol{\lambda}}_i^A + \mathbf{a}_i^T\tilde{K}_i^{AT}\tilde{\boldsymbol{\lambda}}_i^A + \tilde{\mathbf{l}}_i^T\tilde{\boldsymbol{\lambda}}_i^A \right\} \tag{61a}$$

$$\hat{K}_i^A\mathbf{a}_i + \hat{\mathbf{l}}_i = 0. \tag{61b}$$

The solution to (61a) is easily computed due to the diagonality of Σ_i

$$\tilde{\boldsymbol{\lambda}}_i^{A*} = \Sigma_i^{-1}\left(\tilde{K}_i^A\mathbf{a}_i + \tilde{\mathbf{l}}_i\right). \tag{62}$$

Substituting (62) back into the cost-to-go Lagrangian in (39) gives the following updates to its terms:

$$\begin{aligned}&H_i^A \leftarrow H_i^A + \tilde{K}_i^{AT}\Sigma_i^{-1}\tilde{K}_i^A, \; \mathbf{f}_i^A \leftarrow \mathbf{f}_i^A + \tilde{K}_i^{AT}\Sigma_i^{-1}\tilde{\mathbf{l}}_i \\ &K_i^A \leftarrow \hat{K}_i^A, \; \mathbf{l}_i \leftarrow \hat{\mathbf{l}}_i, \; \boldsymbol{\lambda}_i^A \leftarrow \hat{\boldsymbol{\lambda}}_i^A \\ &L_i^A \leftarrow 0_{(m_{if}-m_{ir})\times(m_{if}-m_{ir})}.\end{aligned} \tag{63}$$

The backward recursion is performed using these modified terms in (42). The early elimination is performed at each joint after $L_{(i)}^A$ is updated, which resets $L_{(i)}^A$ to zero matrix. Early elimination reduces the number of propagated constraints at each ith joint by m_{ir}, which is the rank of $K_i^A S_i$ and usually equal to n_i, except in the case of redundant constraints or kinematic singularities. If all the constraints are eliminated before reaching the root node, the backward sweep reduces to the ABA algorithm.

During the forward sweep, the optimal $\boldsymbol{\lambda}_i^{A*}$ is reconstructed using $\tilde{\boldsymbol{\lambda}}_i^{A*}$ from (62) and $\hat{\boldsymbol{\lambda}}_i^{A*}$ (available from the previous link) by transforming back to the original basis

$$\boldsymbol{\lambda}_i^{A*} = U_i^1\tilde{\boldsymbol{\lambda}}_i^{A*} + U_i^2\hat{\boldsymbol{\lambda}}_i^{A*}. \tag{64}$$

For the common case of a single DOF joint, L_i^A undergoes a rank-1 update in (57) and computing its SVD is computationally simple, with the singular vectors given by the following symmetric reflection matrix: [3]

$$\begin{bmatrix}U_i^1 & U_i^2\end{bmatrix} = I_{m_{if}\times m_{if}} - 2\frac{\mathbf{w}_i\mathbf{w}_i^T}{\mathbf{w}_i^T\mathbf{w}_i} \tag{65}$$

and the positive singular value is

$$\Sigma_i = \left[\|K_i^A S_i\|^2/D_i\right] \tag{66}$$

where

$$\mathbf{w}_i = K_i^A S_i + \frac{(K_i^A S_i)_1}{|(K_i^A S_i)_1|}\|K_i^A S_i\|\mathbf{e}_1 \tag{67}$$

where $(K_i^A S_i)_1$ is the first element of the vector $(K_i^A S_i)$ and $\mathbf{e}_1$ is the first canonical basis vector.

Remark 6: Since the rank-1 update SVD can be computed using just $K_i^A S_i$ and D_i, the L_i^A matrix need not be explicitly updated. Furthermore, U_i^1 and U_i^2 matrices are not explicitly computed either because they are only needed for multiplying

[3][Online]. Available: https://math.stackexchange.com/questions/704238/singular-value-decomposition-of-rank-1-matrix

other matrices in (60) and (64), which is efficiently achieved by simply multiplying the RHS of (65). For example

$$\begin{bmatrix}\tilde{K}_i^A & \hat{K}_i^A\end{bmatrix} = \left\{ I_{m_{if}\times m_{if}} - 2\frac{\mathbf{w}_i\mathbf{w}_i^T}{\mathbf{w}_i^T\mathbf{w}_i}\right\} K_i^A. \quad (68)$$

Remark 7: Equation (67) assumes $(K_i^A S_i)_1 \neq 0$. If $(K_i^A S_i)_1 = 0$, the rows of $K_i^A S_i$ are permuted, such that $(K_i^A S_i)_1 \neq 0$, similarly to the pivoting methods in matrix factorization algorithms [45].

Remark 8: If $K_i^A S_i = 0_{m_{if}\times 1}$, the ith joint's acceleration is unaffected by the constraint forces $K_i^{AT}\boldsymbol{\lambda}_i^A$. In this case, the rank-1 update of L_i^A in (57) would only add a zero matrix and is not performed. The terms K_i^A and $\mathbf{l}_i$ are propagated to the parent link as in the original solver using (42c) and (42d) without size reduction.

Complexity analysis

The PV-early solver's salient feature compared with the PV solver is that L_i^A is not computed (hence, L_0^A is not factorized) and the matrices K_i^A and $\mathbf{l}_i^A$ reduce in size during the backward sweep instead of growing with the accumulation of constraints. If the number of rows of K_i^A and $\mathbf{l}_i^A$ is bounded by 6, the complexity of the PV-early solver is $O(n+m)$, as the number of operations at every joint is bounded by a constant.

Remark 9: If the K_i^A and $\mathbf{l}_i^A$ have more than six rows in the PV-early solver, it implies an over constrained system with more than six constraints on a link's acceleration. Then, either the constraints are feasible with redundant constraints or infeasible, when one can remove the redundant constraints to obtain a constraint matrix K_i^A with at most six rows or declare infeasibility early, respectively.

IX. Experiments and Discussion

We now benchmark and discuss the proposed algorithms as follows.

1) We explain our implementation.
2) We benchmark the OSIM computation.
3) We benchmark the CDAs themselves.
4) We empirically test the computational scaling of the different algorithms.
5) We discuss results and limitations of the proposed algorithms.

A. Implementation

We implemented the algorithms by extending Featherstone's highly readable MATLAB software toolbox SpatialV2 [57]. For computing the OSIM, we implemented PV-OSIM and PV-OSIM-fast algorithms and to benchmark them we also implemented the KJR, EFPA, and LTL [32], [43] algorithms. For computing the constrained dynamics, we implemented PV, PV-early, and the PV-soft algorithms and to benchmark them we also implemented the CDAs using Featherstone's sparsity-exploiting LTL approach considering both the hard and the soft motion constraints. Robot specific C-code was generated for these algorithms using CasADi's scalar expressions (SX) [58] and its runtimes are used for the comparison. All the numerical experiments are performed on a single CPU core on a laptop with Intel i7-8850H CPU @ 2.60 GHz processor running an Ubuntu 18.04 operating system. We disabled Intel `Turbo Boost` during the benchmarking to reduce CPU frequency variability.

Implementing rigid body dynamics algorithms efficiently involves various nuances discovered by the robotics community over the years. For example, computing quantities in the local body frame instead of the inertial world frame can significantly reduce the number of operations needed [48]. Thus, our implementation also uses body frame, though, the derivation of the algorithms in this article uses inertial frame for notational simplicity. Also using the DH structure for modeling the robot kinematics, whenever possible, makes the dynamics algorithms more efficient [59]. However, this is not always possible, e.g., for kinematic trees, where a parent link can, in general, have DH structure with only one of the children joints. [32], [43] carefully accounted for these nuances in their comparison of the LTL and ABA algorithms. In addition, robot design can also significantly influence the operation count, e.g., some links in the Kuka Iiwa have a 90° rotation between the parent joint's axis and the child joint's axis, resulting in a rotation matrix with only three nonzeros (either 1 or –1) requiring even fewer computations than DH nodes. Therefore, an algorithm's operation count is robot-specific, and manually counting them for a given robot and constraint combination taking into account all the computational nuances would be tedious. Conveniently, CasADi's SX expression graph of an algorithm automatically provides the operation count allowing us to compare the best possible robot-specific operation count of the different algorithms, which we report later in this section.

Our implementation further uses simple optimizations such as avoiding matrix–matrix operations whenever possible, and performing Cholesky factorization and solve instead of computing matrix inverses. The source code of the implementation[4] and the simulation videos of the proposed algorithms[5] are made available. Baumgarte's stabilization was used in the simulations to stabilize the constraints over a long period of time [60], choosing a stabilization period of 0.1 s to avoid overly stiff dynamics as suggested in [1, Sec.-8.3], which interested readers are referred to for further details.

In our numerical experiments below, H and H_3 denotes a general 6-D and 3-D constraint on the "hand" link of a robot, with the corresponding K_i being a random matrix of size 6×6 and 3×6, respectively. F and F_3 are defined similarly for the "foot" link. For the Iiwa, the end-effector was considered the hand link.

B. Benchmarking the OSIM Algorithms

Fig. 2 gives the operation count along with internal breakdown for the proposed PV-OSIM and PV-OSIM-fast algorithms along with the existing state-of-the-art (SOTA) $O(n+md+$

[4][Online]. Available: https://github.com/AjSat/spatial_V2

[5][Online]. Available: https://kuleuven-my.sharepoint.com/:f:/g/personal/ajay_sathya_kuleuven_be/EkuNpQF8BF5NhMiFajeiskIB0steWelFr_sxQkGa1P_Nrg?e=GS9SPghttps://tinyurl.com/z78hkaah

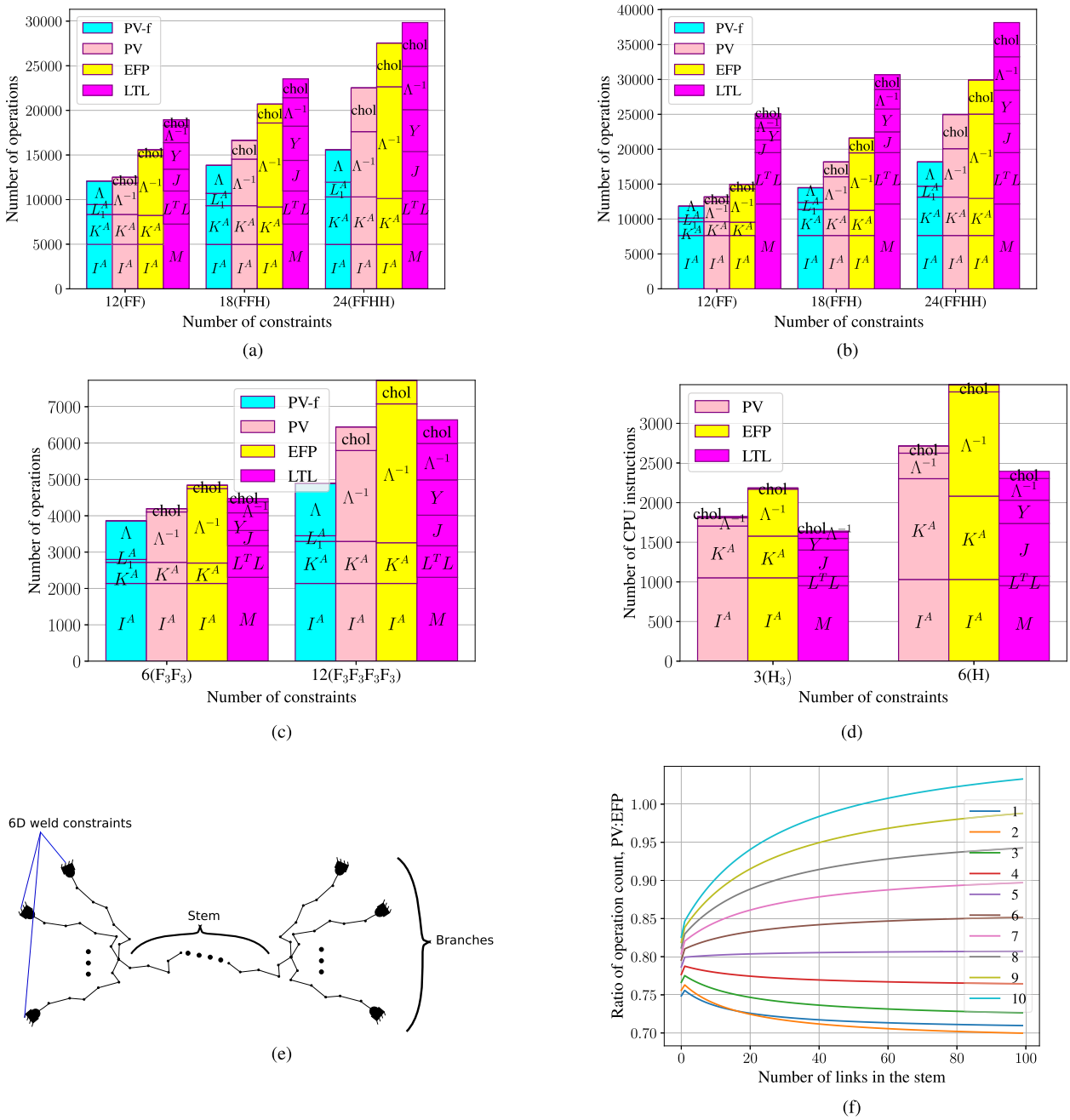


Fig. 2. Benchmarking the number of computation operations of the OSIM algorithms for various robots. (a) Comparing the OSIM algorithms on the Atlas robot. (b) Comparing the OSIM algorithms on the Talos robot. (c) Comparing the OSIM algorithms on the Unitree Go1 quadruped. (d) Comparing the OSIM algorithms on the KUKA Iiwa manipulator. (e) Long stalk on which the computational scaling is studied. (f) Computational scaling of PV-OSIM versus EFPA. The key indicates the number of branches on either side of the long stalk for the mechanism in Fig. 2(e).

m^3) EFPA algorithm [31] and the SOTA sparsity-exploiting $O(nd^2 + m^2\, d + dm^2)$ LTL-OSIM algorithm [32]. Similar to [31], we found KJR to be significantly slower than EFPA for all the considered robots and KJR would also scale worse due to its higher complexity, hence, we omit the KJR results.

We found PV-OSIM to be more efficient than the EFPA for all the considered robots. With the computation of articulated body inertia I^A, the task-space EFP K^A, and the Cholesky decomposition of inverse OSIM requiring the same number of computations for both algorithm, the difference arises in the inverse OSIM Λ^{-1} computation. This is because EFPA requires an additional forward sweep, that propagates inverse inertia matrices forward with expensive similarity transformations, unlike the PV-OSIM as discussed in Section VII-C.

LTL-OSIM was the fastest algorithm for the KUKA Iiwa, which has only 7 DOF. However, for the 18 DOF Go1 robot the PV-OSIM was already slightly faster than the LTL-OSIM due to its lower computational complexity. For bigger robots like

the Atlas (37 DOF) and Talos (50 DOF), LTL was the slowest of all the considered algorithms due to its higher computational complexity. A major difference between the LTL versus EFPA comparison in [31], (which found EFPA to be slower than LTL for the Honda Asimo robot) and ours is that we also include the cost of computing the constraint Jacobian computation J in the LTL algorithm. We believe this to be a fairer comparison because the PV-OSIM and EFPA algorithms do not require J. K^A propagates forces and accelerations from end-effectors to other links fulfilling a role similar to J in LTL. For fewer number of constraints, both PV-OSIM and EFPA are faster than LTL for the Atlas robot. However, if we assume that J is computed elsewhere and is available for re-use, its computation cost can be excluded from LTL operation count. Then, our findings would concur with [31], where LTL would be faster than EFPA for Atlas with 18 or 24 constraints, but still slower than the PV-OSIM. For Talos, LTL was not competitive with the lower order methods especially due to the expense of computing and factorizing a bigger JSIM.

The PV-OSIM-fast avoids computing and factorizing the dense inverse OSIM matrix explicitly using the MIL, and scales better than the PV-OSIM as the size of the OSIM matrix increases. It is the fastest algorithm for the considered floating-base robots and even nearly $2\times$ faster than the LTL for the humanoid robots.

Though the PV-OSIM was computationally faster than the EFPA for all the considered robots, the EFPA has a lower order computational complexity of $O(n + md + m^2)$ compared with the $O(n + m^2\ d + m^2)$ of the PV-OSIM for computing the inverse OSIM Λ^{-1}. This would make EFPA scale better than PV-OSIM for longer mechanisms with many constraints. To test this, we consider a long-stemmed mechanism (n_{stem} is the number of links in the stem). From both stem ends, m_{branches} chains of seven links each branch out as shown in Fig. 2(e). Each branch's tip link is fixed with a 6-D weld constraint.

Fig. 2(f) shows the computational scaling of the ratio of PV-OSIM and EFPA operation counts w.r.t. n_{stem} for different values of m_{branches}. EFPA was found to be always slower than PV-OSIM for up to eight branches (8×6 constraints propagated) irrespective of n_{stem}. For nine or more branches, the EFPA eventually becomes more efficient than PV-OSIM at a crossover point stem length n_{stem}. The value of the crossover point depends on m_{branches} as well as the branches' link length for the considered mechanism. More branches would reduce the crossover point as EFPA can more efficiently propagate large number of constraints through the stem links. Shorter branch length can also reduce the crossover point because the constraint propagation through the stem links (where EFPA is more efficient than PV-OSIM) will form a fraction of the computations. For a $m_{\text{branches}} = 10$, the crossover $n_{\text{stem}} = 54$ for branch length of 7, which is a very large mechanism with $54 + 10 \times 7 \times 2 = 194$ links. For an extreme branch length of only one link, the crossover n_{stem} can be as small as 7. Based on these findings, we conclude that the PV-OSIM requires fewer operations for most realistic robot mechanisms unless one is considering a heavily constrained mechanism with most constraints propagated through a large fraction of the joints.

TABLE I
BENCHMARKING COMPUTATIONAL PERFORMANCE OF PV SOLVER WITH OTHER CONSTRAINED DYNAMIC SOLVERS IN MUJOCO AND PINOCCHIO

Robot	PV	PV-e	LTL	Pin*	PV-s	LTL-s	Mu*
Iiwa (0D)	**0.55**	**0.55**	0.63	2.15	**0.55**	0.63	3.11
Iiwa (H_3)	0.75	**0.61**	0.83	2.73	**0.61**	0.80	4.45
Iiwa (H)	**1.01**	1.09	1.08	3.53	**0.63**	0.89	4.88
Go1 (0D)	**1.65**	**1.67**	1.74	4.68	**1.64**	1.74	7.10
Go1 (F_3)	1.88	**1.81**	1.96	5.61	**1.70**	1.84	11.2
Go1 ($2F_3$)	2.10	**1.98**	2.20	6.40	**1.76**	1.98	12.0
Go1 ($3F_3$)	2.32	**2.14**	2.48	7.33	**1.82**	2.16	12.8
Go1 ($4F_3$)	2.53	**2.33**	2.85	8.20	**1.90**	2.33	13.5
Atlas (0D)	**3.44**	**3.47**	4.64	12.3	**3.47**	4.64	15.9
Atlas (F)	4.59	**3.94**	5.88	15.4	**3.61**	5.58	31.5
Atlas (2F)	6.09	**4.40**	7.52	18.5	**3.73**	6.61	34.2
Atlas (2F+H)	7.37	**5.03**	8.69	22.3	**3.76**	6.93	36.5
Atlas(2F+2H)	8.27	**5.52**	11.8	26.5	**3.82**	7.77	38.8
Talos (0D)	**4.92**	**4.97**	8.14	17.1	**4.96**	8.28	23.6
Talos (F)	5.63	**5.48**	9.25	21.1	**4.96**	8.65	51.3
Talos (2F)	6.72	**6.45**	10.9	25.2	**4.99**	9.21	54.3
Talos (2F+H)	8.40	**7.06**	13.4	30.0	**5.08**	10.6	57.0
Talos(2F+2H)	10.13	**7.40**	15.4	34.7	**5.11**	11.9	59.4

* Pin and Mu are nominal execution of PINOCCHIO and MUJOCO without code-generation and hence cannot be considered fair comparison.
All Times are in microseconds.

C. Benchmarking CDAs

We compared the PV solver, PV-e solver, and the PV-s solver with the state-of-the-art sparsity exploiting LTL solver of Featherstone [32], [43]. The LTL-OSIM [32] solver is a popular algorithm implemented in the high-performance simulator software PINOCCHIO [61]. The LTL solver is also used in MUJOCO [42], which uses a joint-space version of the soft-Gauss principle. To make a fair comparison with the LTL solvers, we implemented them ourselves and Table I reports the computation time taken by the different algorithms. The type and the number of constraints imposed are reported next to the robot name in parentheses.

The computation times for the nominal C++ and C execution of PINOCCHIO (Pin) and MUJOCO (Mu), respectively, cannot be considered a fair comparison because they do not use code-generation, (which prunes unnecessary computations) and may compute additional quantities that are not required for constrained dynamics. We still report their computation timings for reference and indicative purpose of the speed-ups these software may achieve by exploiting code-generation.

1) Hard Motion Constraints: The PV-solver was as fast or faster than the sparsity-exploiting LTL methods for all the considered robots. The difference, while negligible for the 7 DOF Iiwa robot, widens for larger robots and more constraints due to its lower order complexity. Our PV-e solver scales even better than the PV solver, due to its lower order complexity of $O(n + m)$. For larger robots like Atlas or Talos with a high number of constraints, PV-e offers nearly a 50% and 30% reduction in computation compared with LTL and the PV-solver, respectively.

2) Soft Constraints: The last three columns of the Table I present the computation times of our PV-s solver (see Section VI), our implementation of the MUJOCO's soft Gauss principle using LTL and the nominal C execution in MUJOCO itself. In MUJOCO, we imposed 6-D `weld`-type equality constraints

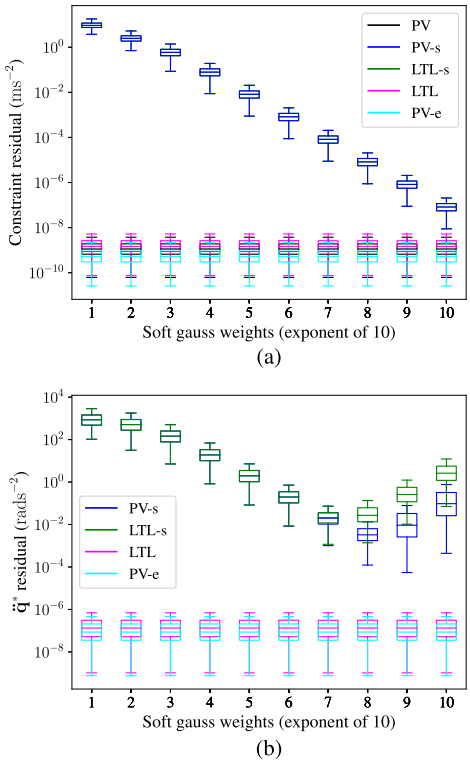


Fig. 3. Benchmarking the numerical accuracy of soft Gauss solver for different weights. (a) Constraint residuals for different soft Gauss weights. (b) Residuals of $\ddot{\mathbf{q}}^*$ w.r.t PV.

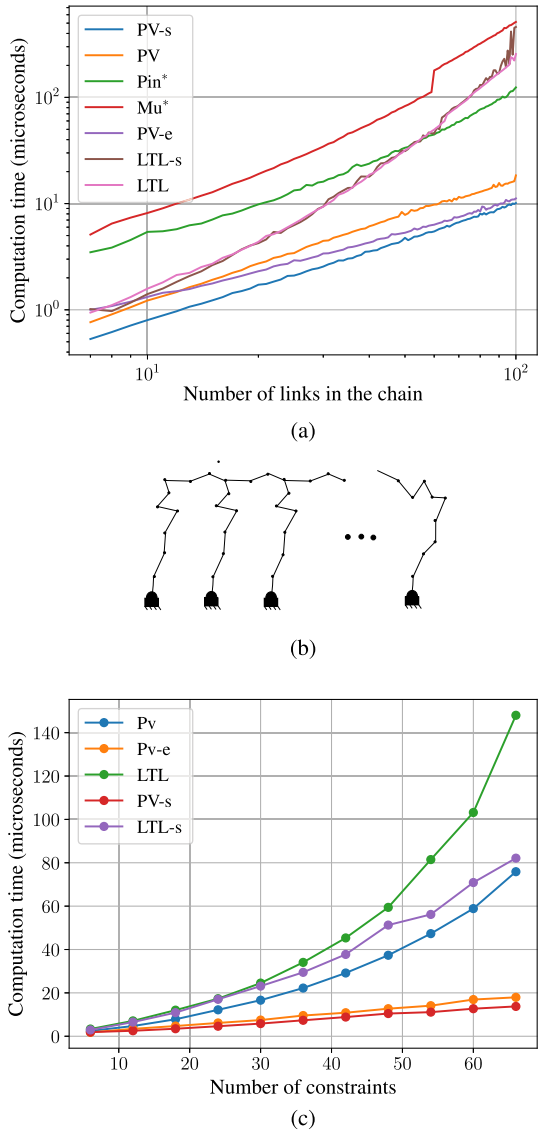


Fig. 4. Computational scaling of the different algorithms. (a) Computational scaling for chains with fixed-base and 6-D constrained end effector. (b) Ladder mechanism where $m \sim O(n)$. (c) Benchmarking computational scaling for the ladder mechanism.

for F or H and 3-D `connect`-type equality constraints for F_3 and H_3, respectively. We deactivated all other constraints and frictional contacts (turned ON by default in MuJoCo) to ensure that it solves the same equality constrained dynamics problems. The PV-s implementation is significantly faster than all the other algorithms. It is nearly twice as fast as LTL-s and nearly thrice as fast as LTL (which arguably solves harder problem with hard motion constraints). It is unlikely that any constrained dynamics algorithm, that we know of, can compete with PV-s since its computation cost is nearly the same as that of the ABA algorithm (unconstrained forward dynamics algorithm with $O(n)$ complexity).

3) Accuracy of the Proposed Solvers: We benchmarked the accuracy of the soft Gauss principle for different value of weights in Fig. 3. We present the whisker plots of ℓ_2 norm of the constraint residuals in Fig. 3(a), and the ℓ_2 norm of the difference in $\ddot{\mathbf{q}}^*$ computed by the PV solver (reference algorithm because it considers hard motion constraints) in Fig. 3(b) for the Talos robot with 2H+2F constraint (both its feet and hands are fixed with a full 6-D constraint) at 1000 different randomly sampled joint configurations. PV, PV-e, and LTL that solve for hard equality constraints satisfy the constraint to high level of accuracy, with PV-e appearing to be numerically slightly stabler than the other two. Both the soft Gauss solvers, PV-s, and LTL-s, have a significantly higher value of constraint residual, though the residual keeps reducing as the penalty weights are increased. Both PV-s and LTL-s satisfy the constraints equally well. However, for weights higher than a certain point ($\sim 10^8$), the optimal joint accelerations computed by the soft Gauss solvers and the hard Gauss solvers begin to diverge due to numerical issues, where the high penalty weights begin to affect the joint acceleration solution in the nullspace of the constraints. Between the two soft Gauss solvers, PV-s appears to be more numerically stable than LTL-s.

D. Computational Scaling

We empirically tested the computational scaling of the different CDAs and present the results in Fig. 4. In Fig. 4(a), we show computational times of the different algorithms for kinematic chains ranging from 6 to 100 revolute joints. The end-effectors are fixed with full 6-D constraints. As expected, the $O(n)$ complexity PV, PV-e, and PV-s solvers scale linearly

and more gracefully than the higher order LTL and LTL-s algorithms used in PINOCCHIO and MUJOCO, respectively. Beyond a certain number of links, the generated C-code for LTL and LTL-s became too large for effective compiler optimization and they became slower than even the nominal C++ execution in PINOCCHIO. Then, we compared the different algorithms on a highly constrained ladder-shaped mechanism [see Fig. 4(b)] with $m \sim O(n)$, with each rung consisting of seven links. The segment connecting two ends of a rung on one side has three links and the other ends of the rung are constrained to be fixed with full 6-D constraints. The computational timings of different algorithms as more rungs (and constraints) are added to the mechanism are presented in Fig. 4(c). The PV solver with its cubic complexity in the number of constraints also begins to scale badly like the LTL and LTL-s solvers, whereas the $O(m+n)$ solvers PV-e and PV-s scale linearly.

E. Discussion and Limitations

Parallel algorithms: Our comparison was limited to implementations on a single core. However, the divide-and-conquer algorithms [37], [38], [39], [40] may be computationally faster, especially for bigger mechanisms, when multiple cores are utilized. On a single core, however, they are unlikely to be faster for typical robots since they are known to be several times more expensive than ABA [38]. However, due to a lack of open source implementation and due to the complexity of their implementation, we leave this comparison for future work.

Among these divide-and-conquer methods the PV solver appears to be most closely related to the pivoted divide-and-conquer algorithm (DCAp) algorithm [38], which has outward acceleration propagation and inward force propagation similar to the PV solver and the ABA is shown to be a special case of DCAp. It appears to be possible to provide an alternative derivation of the PV solver from the DCAp algorithm by placing a handle on the floating-base and the constrained links. The handles on the constrained links would be in the constraint space instead of the spatial handle explicitly considered in [38]. Then, using the two-handle equation in [38, Sec. 4.1], for a specific order of assembly from the leaf nodes to the root, it is possible to show that [38, Eqs. (29a), (29g), (29b), (29 h), and (29d)] correspond to (42a), (42b), (42c), (42d), and (42e) respectively. However, such an assembly ordering is not the recommended ordering in divide-and-conquer algorithms as it does not assemble two trees of similar sizes, which is necessary for obtaining a reduced order complexity in the divide-and-conquer methods.

Though there is no direct analogue for the PV-early algorithm in DCAp, a simpler form of early elimination can also be performed in DCAp when the L^A matrix reaches full rank by eliminating the constraint forces by taking Schur complement. Due the divide-and-conquer methods being among the most complex rigid-body dynamics algorithms in literature, deriving the PV solver this way may not be of interest to readers. However, this connection opens up interesting possibilities for parallelizing the algorithm, which we leave for future work.

Closed-loop solvers: The PV solver is closely related to the algorithms in [33] and [34]. In the PV solver's backward recursion, the (20c), (20d), and (20e) correspond to [33, Eqs. (16c), (18a), and (18b)], respectively, and [34, Eqs. (41c), (51b), and (51a)], respectively. Applicationwise, the main difference between PV-solver and [33] and [34] is that we consider known acceleration constraints, (which includes all the loop closure constraints with the ground as a special case), while both [33] and [34] tackle the harder problem of internal kinematic loop constraints. We also explicitly consider floating-base systems, which was not considered in [33], whereas [34] does consider floating-base systems in one of their examples though not in the main derivation. Both [33] and [34] can be straight-forwardly adapted to solve the constrained dynamics problems considered by the PV solver. This connection between the PV solver, [33] and [34] appears to not have been made in existing literature. Despite not being a fundamentally new algorithm, the expository PV solver derivation in Section III and Section V is of value to the readers because it utilizes a different LQR perspective that permitted a mechanistic derivation of the algorithms, that would make the material accessible to researchers with control and optimization background. In contrast, [33] required significant physical insight to come up with an efficient propagation of Newton–Euler solutions similarly to the ABA algorithm [16]. However, [33] approach may be more accessible to researchers with a background in mechanics and without prior experience in optimal control or optimization.

$O(n+m)$ *solvers:* Our expository derivation also allowed us to easily derive two different and original (to the best of the authors' knowledge) $O(n+m)$ solvers, using the soft Gauss principle adopted by MUJOCO and early elimination of dual variables. A form of early elimination is also proposed in [33] and [34], where they eliminate the dual variables of a loop after passing over all the links in that loop. For certain robot architectures where the loops are not heavily interconnected (the same link being part of multiple loops), their early elimination procedure leads to $O(m+n)$ performance. Another $O(n+m)$ complexity solver for closed loops [35], also uses zero-mass phantom links for loop-cutting and early elimination at the loop level similarly to [33], but employs a Lagrange multiplier-free formulation based on Kane's equations [36] and propagates additional inertia terms unlike [33] and [34]. Despite its interesting approach, [35] is fairly complex, lacks an open-source implementation and is unclear if it is an improvement over [33] and [34] and comparing them for closed loops may be an interesting future step. However, our early elimination is fundamentally different from these methods as it reduces the dimensionality of constraint equations at every joint.

The SVD currently proposed for PV-early is admittedly an expensive algorithm for multiDOF joints, when we cannot exploit the efficient rank-1 update formulae presented in Section VIII, unless the multiDOF joints are modeled as several equivalent fictitious single DOF joints in a chain. However, this workaround is nonideal as it introduces issues like representation singularity and nonphysical meaning of velocities of these fictitious joints. It may be worthwhile to explore replacing the SVD with the more efficient rank-revealing QR decomposition [45] in the future, which provides the desired orthogonal bases similarly to the SVD.

OSIM and computational benchmarking: That the backward recursion in PV solver, [34] and [33] provides an efficient algorithm to compute the OSIM is a new connection made in this article that we could not find in literature. We are also not aware of existing work that computationally benchmarked the PV-solver or the [33] and [34] algorithms with the currently popular sparsity-exploiting methods of Featherstone for the constrained dynamics problems considered in this article. Our findings indicate that for larger robots like the humanoid robots the sparsity-exploiting methods are not competitive with the PV solver, which has implications for the existing simulators and as well as for biomechanical applications where the DOF are typically over 100.

Our benchmarking methodology included code-generating and compiling robot-specific C code, which while contributing to the speeds we observe, is also a limitation as we need to know all the possible contact situations that may arise. Nominal C++ implementations, such as in PINOCCHIO can deal with these scenarios more effectively as they do not require recompilation at runtime. However, in many applications, e.g., humanoid walking, all the possible contact scenarios can be compiled in advance and loaded depending on the contact scenario using look-up tables. In any case, the speed-up we observed due to code-generation is high enough that it is interesting for simulators to explore a hybrid method combining the strengths of both code-generation and nominal C++ execution for different parts of the algorithm.

Finally, we refer interested readers to several extensions of the unconstrained LQR algorithm to equality-constrained problems [62], [63], [64], [65] in a control setting. Out of these methods [62] is analogous to the original PV solver and [64]'s method is most similar to our PV-early solver, where they also used SVD.

X. Conclusion

A. Conclusion

We provided a self-contained derivation of several advanced CDAs from the first principles by connecting it to the LQR problem. Our derivation, building upon Vereshchagin's approach, is much simpler than the better known SOA framework of Rodriguez [20] that uses this LQR connection. Our expository derivation extended the original PV solver to floating-base kinematic trees, which resulted in an algorithm closely related to [34] and [33], but is derived using a different LQR perspective. This article makes constrained dynamics accessible to researchers in optimization and control as well as roboticists, with knowledge of control, that currently treat robot dynamics as a black-box and are, therefore, unable to debug or adapt existing dynamics software to their applications. The LQR connection can foster transfer of software and ideas between fields in the future. For example, recent research from data-driven LQR control may transfer to robust control of robots with uncertain dynamics. The optimization perspective in our derivation is valuable as accounting for uncertainty in parameters is performed naturally in an optimization framework [66], [67].

The equality we showed between LQR's dual Hessian and the inverse OSIM provided an efficient state-of-the-art OSIM algorithm, which we further significantly accelerated for specific, but common, robot structures that have branching at the base. The LQR-based approach allowed straightforward derivation for the PV-s and PV-early algorithms, resulting in two original algorithms with $O(n+m)$ complexity. Our numerical experiments suggest that the PV solver is computationally superior to currently popular higher order sparse factorization algorithms by Featherstone for larger robots like the humanoid robot Atlas, for which the LTL needs up to $2\times$ more computations than the PV-solver. This PV-solver speed-up can be arbitrarily higher for longer mechanisms, typical in biomechanical applications, due to the inherent complexity difference. Finally, our work recognizes the historical contribution of Popov and Vereshchagin who proposed the first $O(n)$ CDA, which remarkably remains the state-of-the-art nearly fifty years after its invention and yet remains largely unknown in the robotics community.

B. Future Work

There are multiple exciting directions for future work, apart from the applications in robot control and trajectory optimization. The algorithms presented here are limited to equality constraints, and it is a natural research direction to extend the algorithms to include internal kinematic loops, frictional contacts, and unilateral contact constraints. We will also explore proximal point iterations [6] for applying the solver to problems with illconditioned and nearly redundant constraints. Analytical gradients, which are found to be faster than automatic differentiation, can also be developed for the PV solver for optimal control and reinforcement learning applications. In particular, transfer of new research results from data-driven LQR to robot control is an exciting future research direction.

Acknowledgment

The authors would like to thank Prof. J. Swevers, B. Vandewal, and A. Vigoya for valuable feedback on previous manuscript versions and the anonymous reviewers for their valuable comments and suggestions, especially the Reviewer 1 for an extensive review and suggesting important missing references (e.g., [33]).

References

[1] R. Featherstone, *Rigid Body Dynamics Algorithms*. Berlin, Germany: Springer, 2014.
[2] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, vol. 2. Madison, WS, USA: Nob Hill Publishing, 2017.
[3] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 69–81, 2014.
[4] M. Neunert, F. Farshidian, A. W. Winkler, and J. Buchli, "Trajectory optimization through contacts and automatic gait discovery for quadrupeds," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1502–1509, Jul. 2017.
[5] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2021. [Online]. Available: http://pybullet.org
[6] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and sparse resolution of constrained dynamic equations," in *Proc. Robot., Sci. Syst.*, 2021.

[7] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 6054–6061.

[8] J. Lee et al., "DART: Dynamic animation and robotics toolkit," *J. Open Source Softw.*, vol. 3, no. 22, Feb. 2018, Art. no. 500.

[9] B. Plancher, S. M. Neuman, R. Ghosal, S. Kuindersma, and V. J. Reddi, "Grid: GPU-accelerated rigid body dynamics with analytical gradients," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2022, pp. 6253–6260.

[10] D. Baraff, "Linear-time dynamics using lagrange multipliers," in *Proc. ACM SIGGRAPH*, 1996, pp. 137–146.

[11] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL, USA: CRC, 2017.

[12] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge, U.K.: Cambridge Univ. Press, 2017.

[13] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 1168–1175.

[14] A. Mesbah et al., "Fusion of machine learning and MPC under uncertainty: What advances are on the horizon?," in *Proc. Amer. Control Conf.*, 2022, pp. 342–357.

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

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

[17] R. Featherstone and D. Orin, "Robot dynamics: Equations and algorithms," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, pp. 826–834.

[18] C. F. Gauß, "Über ein neues allgemeines grundgesetz der mechanik," *J. fur die Reine und Angewandte Math.*, vol. 4, pp. 232–235, 1829.

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

[20] G. Rodriguez, "Kalman filtering, smoothing, and recursive robot arm forward and inverse dynamics," *IEEE J. Robot. Automat.*, vol. 3, no. 6, pp. 624–639, Dec. 1987.

[21] G. Rodriguez, A. Jain, and K. K.-Delgado, "A spatial operator algebra for manipulator modeling and control," *Int. J. Robot. Res.*, vol. 10, no. 4, pp. 371–381, 1991.

[22] G. Rodriguez and K. K.-Delgado, "Spatial operator factorization and inversion of the manipulator mass matrix," *IEEE Trans. Robot. Autom*, vol. 8, no. 1, pp. 65–76, Feb. 1992.

[23] J. P. Popov, A. F. Vereshchagin, and S. L. Zenkevič, *Manipuljacionnyje Roboty: Dinamika i Algoritmy*. Nauka, 1978.

[24] A. F. Vereshchagin, "Modeling and control of motion of manipulational robots," *Sov. J. Comput. Syst. Sci.*, vol. 27, no. 5, pp. 29–38, 1989.

[25] A. Shakhimardanov, "Composable robot motion stack: Implementing constrained hybrid dynamics using semantic models of kinematic chains," Leuven, 2015. [Online]. Available: https://lirias.kuleuven.be/1747300?limo=0

[26] S. Schneider and H. Bruyninckx, "Exploiting linearity in dynamics solvers for the design of composable robotic manipulation architectures," in *Proc. IEEE/RSJ Int. Conf. Int. Robots. Syst.*, 2019, pp. 7439–7446.

[27] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational-space formulation," *IEEE J. Robot. Autom.*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

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

[29] G. Rodriguez, A. Jain, and K. Kreutz, "Spatial operator algebra framework for multibody system dynamics," in *Proc. 3rd Annu. Conf. Aerosp. Comput. Control,* 1989.

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

[31] 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.*, 2012, pp. 4911–4917.

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

[33] M. Otter, H. Brandl, and R. Johanni, "An algorithm for the simulation of multibody systems with kinematic loops," in *Proc. 7th World Congr. Theory Mach. Mechanisms Sevilla*, 1987.

[34] D.-S. Bae and E. J. Haug, "A recursive formulation for constrained mechanical system dynamics: Part II. closed loop systems," *J. Struct. Mechanics*, vol. 15, no. 4, pp. 481–506, 1987.

[35] K. S. Anderson and J. Critchley, "Improved order-n performance algorithm for the simulation of constrained multi-rigid-body dynamic systems," *Multibody Syst. Dyn.*, vol. 9, pp. 185–212, 2003.

[36] T. R. Kane and D. A. Levinson, *Dynamics, Theory and Applications*. New York, NY, USA.: McGraw Hill, 1985.

[37] R. Featherstone, "A divide-and-conquer articulated-body algorithm for parallel O (log (n)) calculation of rigid-body dynamics. part 1: Basic algorithm," *Int. J. Robot. Res.*, vol. 18, no. 9, pp. 867–875, 1999.

[38] R. Featherstone, "A divide-and-conquer articulated-body algorithm for parallel O (log (n)) calculation of rigid-body dynamics. part 2: Trees, loops, and accuracy," *Int. J. Robot. Res.*, vol. 18, no. 9, pp. 876–892, 1999.

[39] K. Yamane and Y. Nakamura, "Comparative study on serial and parallel forward dynamics algorithms for kinematic chains," *Int. J. Robot. Res.*, vol. 28, no. 5, pp. 622–629, 2009.

[40] K. D. Bhalerao, J. Critchley, and K. Anderson, "An efficient parallel dynamics algorithm for simulation of large articulated robotic systems," *Mechanism Mach. Theory*, vol. 53, pp. 86–98, 2012.

[41] K. D. Bhalerao, J. Critchley, D. Oetomo, R. Featherstone, and O. Khatib, "Distributed operational space formulation of serial manipulators," *J. Comput. Nonlinear Dyn.*, vol. 9, no. 2, 2014, Art. no. 021012.

[42] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Int. Robots. Syst.*, 2012, pp. 5026–5033.

[43] 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.

[44] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *Int. J. Robot. Res.*, vol. 33, no. 7, pp. 1006–1028, 2014.

[45] G. H. Golub and C. F. V. Loan, *Matrix Computations*. Baltimore, MD, USA: JHU Press, 2013.

[46] F. E. Udwadia and R. E. Kalaba, *Analytical Dynamics : A New Approach*, Cambridge, U.K.: Cambridge Univ. Press, 1996.

[47] H. Bruyninckx and O. Khatib, "Gauss' principle and the dynamics of redundant and constrained manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2000, pp. 2563–2568.

[48] 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 Proc. Volumes*, vol. 19, no. 14, pp. 95–100, 1986.

[49] K. W. Lilly, *Efficient Dyn. Simul. of Mult. Chain Robot. Syst.*. Columbus, OH, USA: Ohio State Univ., 1989.

[50] S. Lucia, T. Finkler, and S. Engell, "Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty," *J. Process Control*, vol. 23, no. 9, pp. 1306–1319, 2013.

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

[52] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *J. Dyn. Syst. Meas. Control*, vol. 104, no. 3, pp. 205–211, Sep. 1982, doi: 10.1115/1.3139699.

[53] L. Righetti, J. Buchli, M. Mistry, M. Kalakrishnan, and S. Schaal, "Optimal distribution of contact forces with inverse-dynamics control," *Int. J. Robot. Res.*, vol. 32, no. 3, pp. 280–298, 2013.

[54] A. Dietrich, C. Ott, and A. A.-Schäffer, "An overview of null space projections for redundant, torque-controlled robots," *Int. J. Robot. Res.*, vol. 34, no. 11, pp. 1385–1400, 2015.

[55] 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, Apr. 2018. [Online]. Available: www.raisim.com

[56] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *Ann. Math. Statist.*, vol. 21, no. 1, pp. 124–127, 1950.

[57] R. Featherstone, "Spatial v2," 2015. [Online]. Available: http://royfeatherstone.org/spatial/v2/index.html

[58] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[59] S. McMillan and D. E. Orin, "Efficient computation of articulated-body inertias using successive axial screws," *IEEE Trans. Robot. Autom*, vol. 11, no. 4, pp. 606–611, Aug. 1995.

[60] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Comput. Meth. Appl. Mech. Eng.*, vol. 1, no. 1, pp. 1–16, 1972.

[61] J. Carpentier et al., "The Pinocchio c library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *Proc. IEEE Int. Symp. Syst. Integr.*, 2019, pp. 614–619.
[62] J. H. Park, S. Han, and W. H. Kwon, "LQ tracking controls with fixed terminal states and their application to receding horizon controls," *Syst. Control Lett.*, vol. 57, no. 9, pp. 772–777, 2008.
[63] M. Giftthaler and J. Buchli, "A projection approach to equality constrained iterative linear quadratic optimal control," in *Proc. IEEE Int. Conf. Hum. Robot.*, 2017, pp. 61–66.
[64] F. Laine and C. Tomlin, "Efficient computation of feedback control for equality-constrained LQR," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6748–6754.
[65] L. Vanroye, J. De Schutter, and W. Decré, "A generalization of the Riccati recursion for equality-constrained linear quadratic optimal control," *Optim. Control Appl. Methods*, 2023, doi: 10.1002/oca.3064.
[66] A. B.-Tal, L. E. Ghaoui, and A. Nemirovski, *Robust Optimization*, vol. 28. Princeton, NJ, USA: Princeton Univ. Press, 2009.
[67] A. Shapiro, D. Dentcheva, and A. Ruszczynski, *Lectures on Stochastic Programming: Modeling and Theory*. Philadelphia, PA, USA: SIAM, 2021.

Ajay Suresha Sathya received the bachelors degree in mechanical engineering from NITK Surathkal, Surathkal, India, in 2016, and the masters and Ph.D. degrees in mechanical engineering from KU Leuven, Leuven, Belgium, in 2018 and 2023, respectively.

His research interests include developing efficient dynamics simulators and high performance robot controllers.

Herman Bruyninckx received the master's degrees in mathematics (Licentiate) in 1984, computer science (Burgerlijk Ingenieur) in 1987, and mechatronics in 1988, all from the KU Leuven, Leuven, Belgium, where he received the doctoral degree in engineering, in 1995.

He is currently a full-time Full Professor with the KU Leuven, and part-time with the Eindhoven University of Technology, Eindhoven, The Netherlands. The complementary objectives are to realize the following systems with the least amount of resource costs, with "good enough" quality, and with full "explainability". His research interests include the composability of the most advanced, knowledge driven algorithms for the dynamics of motion control of complex robotics applications, with distributed sensor processing, and resource monitoring.

Wilm Decré (Member, IEEE) received the B.S., M.S., and the Ph.D. degrees in mechanical engineering from KU Leuven, Leuven, Belgium, in 2004, 2006, and 2011, respectively.

He is a Research Manager with the Department of Mechanical Engineering of KU Leuven. His research interests include sensor- and optimization-based control of robot systems, numerical optimization algorithms and applications, learning and optimal control and estimation, and real-time and embedded software design.

Goele Pipeleers received the M.Sc. degree in mechanical engineering and the Ph.D. degree in mechanical engineering from the KU Leuven, Leuven, Belgium, in 2004 and 2009, respectively.

She is currently with Materialise N.V., Leuven, Belgium. Her current research interest focuses on innovations in additive manufacturing.