Linear-time Differential Inverse Kinematics: an Augmented Lagrangian Perspective

Bruce Wingo*,†, Ajay Suresha Sathya†, Stéphane Caron†, Seth Hutchinson* and Justin Carpentier†
†Inria, École normale supérieure, CNRS, PSL Research University, 75005 Paris, France
*Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA
Email: bwingo@gatech.edu ajay.sathya@inria.fr, stephane.caron@inria.fr, seth@gatech.edu, justin.carpentier@inria.fr

***Abstract*—Differential inverse kinematics is a core robotics problem whose state-of-the-art solutions are currently based on quadratic programming. In this paper, we revisit it from the perspective of augmented Lagrangian methods (AL) and the related alternating direction method of multipliers (ADMM). By embracing AL techniques in the spirit of the rigid-body dynamics algorithms proposed by Featherstone, we introduce a method that solves equality-constrained differential IK problems with linear-time complexity. Combined with the ADMM strategy popularized by OSQP, we handle the same class of problems as QP-based differential IK, but scaling linearly with problem dimensions rather than cubically. We implement our approach as C++ open-source software and evaluate it on a benchmark of robotic-arm and humanoid-locomotion tasks. We measure computation times 2–3× shorter than the QP-based state of the art.**

I. Introduction

Inverse kinematics, the calculation of a configuration that achieves prescribed workspace tasks, has been a long-standing problem in robotics. While analytical solutions can be found mathematically for systems with a few degrees of freedom, the approach scales to at most 6–7 degrees of freedom by automated symbolic calculus [16]. Differential inverse kinematics, the computation of *velocities* that bring workspace goals closer to achievement over time, has emerged as a more general alternative capable of handling systems as large as humanoid robots with multiple simultaneous workspace tasks [32, 41].

Initially handled with Moore-Penrose inversion and nullspace projection, it was later established that differential inverse kinematics with priorities is an instance of lexicographic optimization [18], where the hierarchy between tasks can be exploited by specialized solvers such as the open-source LexLS [17]. Embracing optimization further enabled the integration of configuration and joint-velocity limits as inequality constraints.

Conflicts between tasks in lexicographic optimization lead to singular sub-problems [49], but the strict hierarchy of tasks can be relaxed into a quadratic program (QP) where tasks are weighted in a single objective function. This approach relies on a class of *convex* optimization problems that has received more analysis and software development. It is regularized by the design of its weighted objective and further regularized with the common practice of Levenberg-Marquardt damping [44] to handle unfeasible workspace targets. Owing to these refinements, QP formulations have become the go-to approach to differential inverse kinematics, with applications ranging as far as multitask humanoid locomotion pipelines [29, 10].

Regularization around task conflicts is however not the only tradeoff made in QP formulations. For one, solving a differential IK quadratic program is still a resource-constrained operation, with computation times on the same scale of order as the frequency of the control loops they run in. The use of analytical IK by reinforcement learning agents such as [30] suggests a Pareto front of best-suited solutions, with QP-based differential IK on the safer-slower side and analytical solutions on the riskier-faster one.

In this work, we identify a way to push the boundaries of this Pareto front: we leverage the structure of the kinematic tree, which is abstracted away in QP matrices, to propose a new algorithm that solves the same problem as QP-based differential IK, yet empirically faster and with linear-time complexity. Additionally, differential IK is used as an inner solver in various software like inverse kinematics solvers [42, 6] and sampling-based constrained motion planners [7], where it might be called nearly a hundred times if not thousands of times in case of motion planners. Therefore, our contributions can significantly accelerate these downstream computationally expensive downstream tasks.

Contributions. We exploit the structure of the kinematic tree with a formulation inspired by the constrained articulated body algorithm (constrainedABA) [39], which similarly attained linear-time complexity for constrained dynamics problems. However, constrainedABA only considered the equality-constrained forward dynamics problems and therefore does not support additional terms handled in QP-based differential IK such as joint-space and task-space inequality constraints. We thus propose a new derivation to account for these additional terms. Our primary contributions are:

1) **Linear complexity differential IK solver:** We derive a differential IK inner solver with linear complexity in both joint-space and motion-constraint dimensions.
2) **Inequality constraints:** We propose an ADMM-based strategy dealing with inequality constraints, where each ADMM iteration is made efficient by using the aforementioned inner solver.
3) **Software:** We have implemented our algorithm in a

C++ library, LOIK [1], which we will release and support as open-source software after peer review of this manuscript.

We refer to the resulting algorithm as LOIK, standing *e.g.*, for low-overhead inverse kinematics.

II. BACKGROUND

A. QP-based formulations of inverse kinematics

When casting the differential inverse kinematics (IK) problem into a quadratic program (QP), motion constraints of the former become equality constraints of the latter. In the simplest setting with a single task, the QP can be solved in closed-form by solving a linear system. However, in general, differential IK may include conflicting tasks that result in numerical singularities. The weighted approach addresses these conflicts by defining slack variables on equality constraints, and relaxing the problem into a weighted quadratic penalization over these slack variables. The resulting problem is then solved leveraging mature off-the-shelf QP solvers [43, 23, 3]. While the effectiveness of this methodology has been extensively demonstrated [1, 29], the complexity of tuning weights between interacting tasks increases significantly with the number of tasks. An alternate strategy that does not suffer from this to strictly prioritize motion constraints [41], which requires solving a lexicographic QP. Lexicographic QPs can be straightforwardly solved by solving a cascade of QPs [28, 38], one at each priority level, but its complexity does not scale well with the number of priority levels. The hierarchical quadratic programming (HQP) solver [18] proposed a hierarchical complete orthogonal decomposition (H-COD) solver to efficiently solve lower priority motion constraints in the nullspace of higher-priority ones. This strategy was found to result in a computational cost comparable to the weighted QP strategy despite supporting strict prioritization of constraints.

B. Structure-exploiting QP solvers

All the above inverse kinematics approaches are formulated in the joint space with the generalized velocities being the QP's decision variables. This results in a worst-case cubic computational complexity for these methods in terms of the problem size (the robot's degrees of freedom (DoF) and the dimensionality of motion constraints). However, the structure of kinematic trees can be exploited to significantly speed up the QP solvers by leveraging the Riccati recursion that is commonly used in optimal control problem (OCP) solvers [36]. Compared to OCPs, the Riccati recursion algorithm requires straightforward adaptation to deal with the branching in kinematic trees, which can be derived using dynamic programming (DP). This idea was first used to develop linear complexity forward dynamics algorithms by Vereshchagin [46], resulting in an algorithm practically identical to Featherstone's articulated body algorithm [19, 20]. Vereshchagin's approach was later generalized in the PV solver [34, 47] to compute equality-constrained robot dynamics problems (see [40] for an expository derivation). However, the PV solver has a cubic complexity in the dimensionality of constraints. Recent work [39] on constrained articulated body algorithms (constrainedABA) has leveraged the augmented Lagrangian method to design a linear complexity solver in both robot DoFs and motion constraint dimensionality. Similar ideas are yet to be exploited for solving inverse kinematics problems. Furthermore, the constrainedABA does not support inequality constraints, which are important for inverse kinematics problems.

C. Solving Constrained QP problems via ADMM

A powerful strategy for solving constrained optimization problems is the class of augmented Lagrangian methods (ALM) (also known as the method of multipliers), popularized in the late 1960s [24] and [35]. The ALM transforms the original constrained optimization problem into an equivalent primal-dual problem over a so-called augmented Lagrangian function, which is constructed by adding an appropriate penalty term to the ordinary Lagrangian function. ALMs have recently witnessed a renewed increase in popularity in robotics and control [23, 3, 25, 26] especially due to their suitability for warm-starting.

In the past decade, one flavor of the method of multipliers, in particular, the Alternating Direction Method of Multipliers (ADMM), has gained significant interest and rapid adoption within the optimization community. First introduced in the 1970s by [21], ADMM is tailored to convex-constrained optimization problems with separable decision variables and objectives. In the context of constrained QPs, this translates to treating equality and inequality constraints separately (using slack variables) and solving separate sub-problems pertaining to only a subset of the constraints. These sub-problems are often easier to solve than the original ones.

Several state-of-the-art QP solvers exploit the ALM approach [3], [43], [15]. [43] specifically employs ADMM and is the closest in terms of formulation to our proposed solver. However, one major distinction between the current state-of-the-art and our proposed solution is that our solver can efficiently exploit the specific sparsity patterns arising in IK problems.

III. LOW-COMPLEXITY DIFFERENTIAL INVERSE KINEMATICS: DERIVATION

A. Constrained inverse kinematics ADMM formulation

In the most general sense, first order constrained differential inverse kinematics can be formulated as a constrained QP problem:

$$\min_{\bar{\mathbf{x}}:=[\mathbf{v},\boldsymbol{\nu}],\ \bar{\mathbf{z}}} \quad \frac{1}{2}\bar{\mathbf{x}}^\top \bar{\mathbf{P}}\bar{\mathbf{x}} + \bar{\mathbf{q}}\bar{\mathbf{x}}$$

$$\text{s.t.} \quad \bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{z}} \tag{1a}$$

$$\bar{\mathbf{z}} \in \mathcal{K} \tag{1b}$$

where the primal decision variable $\bar{\mathbf{x}} := [\mathbf{v}, \boldsymbol{\nu}]$ consists of the collection of spatial velocities of every link in the robot,

[1]Our code will be made available at https://github.com/Simple-Robotics/Loik

$\mathbf{v} := [\mathbf{v}_0, \cdots, \mathbf{v}_{n_b}]$, with each $\mathbf{v}_i \in \mathbb{M}^6$ [20]. $\boldsymbol{\nu}$ represent the generalized velocities of the robot. $\bar{\mathbf{z}}$ are slack variables, through which both equality and inequality constraints, on $\bar{\mathbf{x}}$, can be enforced. $\mathcal{K}$ denote the appropriate composite convex constraint set. To illustrate, equality constraints $\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}}$ can be enforced by a degenerate constraint set: $\mathcal{K}_{\text{eq}} := \{\bar{\mathbf{z}} \mid \bar{\mathbf{z}} \in [\bar{\mathbf{b}}, \bar{\mathbf{b}}]\}$. Box constraints can be similarly achieved by defining a box constraint set with a lower and upper bound. Other common constraint types such as second-order cone constraints can be defined naturally within problem (1) formulation. The advantage of defining the inverse kinematics problem in the mixed spatial and joint coordinates is that: constraints naturally expressed in either coordinate can be trivially combined without the need for expensive Jacobian calculations. But more importantly, this mixed-coordinate formulation allows one to fully exploit the sparsity pattern induced by the robot's kinematic tree. By doing so, we are able to produce a constrained inverse kinematics algorithm that scales linearly with the number of links in the robot and with the number of constraints specified in the problem.

To make the subsequent derivation of the proposed algorithm more intuitive, we restrict our presentation to only a subset of problems in the form of problem (1). Specifically, the class of IK problems with task space equality constraints (defined through links' spatial velocities) and joint velocity box constraints. This subclass of problems encapsulates the vast majority of IK problems one may encounter in practice. Extending the provided derivation to problems of form (1) that are not captured by the following exposition should be straightforward.

Consider the following IK problem: minimizing a quadratic tracking objective in link spatial velocities, subject to forward kinematics constraints, task space equality constraints and joint velocity box constraints:

$$\begin{aligned}
\min_{\mathbf{v},\boldsymbol{\nu}} \quad & \sum_{i=1}^{n_b} \frac{1}{2}\|\mathbf{v}_i - \mathbf{v}_i^{\text{ref}}\|^2_{\mathbf{H}_i^{\text{ref}}} \\
\text{s.t.} \quad & \mathbf{v}_i = \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i \quad , \forall i \in \mathcal{N}_{\text{b}} && \text{(2a)} \\
& \mathbf{v}_0 = \mathbf{0} && \text{(2b)} \\
& \mathbf{A}_i\mathbf{v}_i = \mathbf{b}_i \quad , \forall i \in \mathcal{N}_{\text{b}} && \text{(2c)} \\
& \boldsymbol{\nu} - \mathbf{z} = \mathbf{0} \quad , \quad \mathbf{z} \in \mathcal{K}_{\text{box}} && \text{(2d)} \\
& \mathbf{A}_i^{\text{in}}\mathbf{v}_i - \mathbf{z}_i^{\text{in}} = \mathbf{0}, \quad \mathbf{z}_i^{\text{in}} \in \mathcal{K}_{\text{box}}^i, \quad \forall i \in \mathcal{N}_{\text{b}} && \text{(2e)}
\end{aligned}$$

where tracking weights $\mathbf{H}_i^{\text{ref}}$ and references $\mathbf{v}_i^{\text{ref}}$ in the objective define the *soft* constraints of the IK problem, while $\mathbf{A}_i$ and $\mathbf{b}_i$ defines the *hard* task equality constraints on ith link's spatial velocity. We denote by $\pi(i)$ the parent of a body in the kinematic tree, with $\mathcal{N}_{\text{b}} := \{1, \cdots, n_b\}$ the set of all links in the kinematic tree, except the 0-th link, which represents the universe/world reference frame. $\mathbf{v}_i$ represents each link's spatial velocity. Please note that for floating-base robots, the floating-base link is assumed to be connected to the 0-th link with a 'free'-joint that permits motions in all 6 dimensions. $\pi(i)$ represents the index of link i's parent link. $\mathbf{S_i}$s are the motion subspace matrices whose columns span the motion subspace of link i. $\mathbf{S_i}$s always have full column rank [2]. The vector $\boldsymbol{\nu}$ denotes generalized velocities for the entire robot. Slack variables $\mathbf{z}$ are introduced for the box constraint set $\mathcal{K}_{\text{box}} := \{\mathbf{z} \mid \mathbf{z}_{\text{lb}} \leq \mathbf{z} \leq \mathbf{z}_{\text{ub}}\}$. We shall refer to Eqn. (2a) and (2b) as **kinematics** constraints, and Eqn. (2c) as **motion** constraints. $\mathbf{A}_i^{\text{in}}$ together with $\mathcal{K}_{\text{box}}^i := \{\mathbf{z}_i^{\text{in}} \mid \mathbf{z}_{\text{lb}}^i \leq \mathbf{z}_i^{\text{in}} \leq \mathbf{z}_{\text{ub}}^i\}$ defines hard task inequality constraints on $\mathbf{v}_i$.

Due to the algebraic similarities between Eqn. (2e) and Eqn. (2c), the role of $\mathbf{A}_i^{\text{in}}$ and $\mathbf{z}_i^{\text{in}}$ would be identical to that of $\mathbf{A}_i$ and $\mathbf{b}_i$ respectively in the recursive algorithm derived subsequently. The projection operations on $\mathbf{z}_i^{\text{in}}$ and the treatment of the task inequality constraints' Lagrange multipliers is also analogous to the projection operations on $\mathbf{z}_i$ and treatment of Lagrange multipliers associated with the joint inequality constraints in Eqn. (2d). Therefore, to avoid repetition and to enhance notational clarity, the task inequality constraints depicted in Eqn. (2e) will be omitted in the subsequent derivation.

It is not difficult to verify that problem (2) takes on the standard QP of the form (1) by unrolling the recursive definition of kinematics constraints (2b). The recursive relationship, between body i's spatial velocity and its parent body's spatial velocity $\mathbf{v}_{\pi(i)}$ and its parent joint's generalized velocity $\boldsymbol{\nu}_i$, implies that $\mathbf{v}_i$ is fully determined by the joint velocities $\boldsymbol{\nu}_i$s preceding i and the root link spatial velocity $\mathbf{v}_0$. The proposed algorithm exploits this recursive relationship and enforces satisfaction of kinematics constraints (2a) and (2b) through back substitutions, resulting in an efficient and linear complexity algorithm. Based on this observation, we define an augmented Lagrangian function, $\mathcal{L}^{\mathcal{A}}$ consisting of only the remaining constraints (motion constraints) that need to be solved for in the problem (2):

$$\begin{aligned}
\mathcal{L}^{\mathcal{A}}_{\text{IK}} := & \sum_{i=1}^{n_b} \Bigg\{ \frac{1}{2}\|\mathbf{v}_i - \mathbf{v}_i^{\text{ref}}\|^2_{\mathbf{H}_i^{\text{ref}}} + \mathbb{I}_{\mathcal{K}_i}(\mathbf{z}_i) \\
& + \mathbf{y}_i^\top(\mathbf{A}_i\mathbf{v}_i - \mathbf{b}_i) + \mathbf{w}_i^\top(\boldsymbol{\nu}_i - \mathbf{z}_i) \\
& + \frac{\mu_{\text{task}}}{2}\|\mathbf{A}_i\mathbf{v}_i - \mathbf{b}_i\|_2^2 + \frac{\mu_{\text{box}}}{2}\|\boldsymbol{\nu}_i - \mathbf{z}_i\|_2^2 \Bigg\} && \text{(3a)} \\
= & \sum_{i=1}^{n_b} \left\{ \frac{1}{2}\mathbf{v}_i^\top\mathbf{H}_i\mathbf{v}_i + \mathbf{p}_i^\top\mathbf{v}_i + \frac{1}{2}\boldsymbol{\nu}_i^\top\mathbf{R}_i\boldsymbol{\nu}_i + \mathbf{r}_i^\top\boldsymbol{\nu}_i + \cdots \right\} && \text{(3b)}
\end{aligned}$$

where $\mathbb{I}_{\mathcal{K}_i}$ is the indicator function for the constraint set $\mathcal{K}_i$ (with value 0 inside and $+\infty$ outside the set). Terms not involving $\mathbf{v}_i$ and $\boldsymbol{\nu}_i$ are omitted in (3b). Collecting terms in $\mathbf{v}_i$ and $\boldsymbol{\nu}_i$ defines the quadratic and affine terms as:

$$\begin{aligned}
\mathbf{H}_i &:= \mu_{\text{task}}\mathbf{A}_i^\top\mathbf{A}_i + \mathbf{H}_i^{\text{ref}} && \text{(4a)} \\
\mathbf{p}_i &:= \mathbf{A}_i^\top\mathbf{y}_i - \mathbf{H}_i^{\text{ref}^\top}\mathbf{v}_i^{\text{ref}} - \mu_{\text{task}}\mathbf{A}_i^\top\mathbf{b}_i && \text{(4b)} \\
\mathbf{R}_i &:= \mu_{\text{box}}\mathbf{I} && \text{(4c)} \\
\mathbf{r}_i &:= \mathbf{w}_i - \mu_{\text{box}}\mathbf{z}_i && \text{(4d)}
\end{aligned}$$

Under appropriate constraint qualification assumptions [33], the solution to the original problem (2) can be obtained by solving the equivalent primal-dual augmented Lagrangian saddle point problem:

$$(\mathbf{v}^*, \boldsymbol{\nu}^*, \mathbf{z}^*, \mathbf{y}^*, \mathbf{w}^*) = \arg\max_{\mathbf{y},\mathbf{w}} \min_{\mathbf{v},\boldsymbol{\nu},\mathbf{z}} \ \mathcal{L}_{\text{IK}}^{\mathcal{A}}(\mathbf{v}, \boldsymbol{\nu}, \mathbf{z}, \mathbf{y}, \mathbf{w})$$
$$\text{s.t.} \quad \mathbf{v}_i = \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i \quad , \forall i \in \mathcal{N}_\text{b}$$
$$\mathbf{v}_0 = \mathbf{0} \tag{5a}$$

ADMM is particularly suited to solve problems like (5), where primal decision variables $\mathbf{v}, \boldsymbol{\nu}, \mathbf{z}$ appear as separable terms in the augmented Lagrangian [9]. The ADMM iterates for solving problem (5) are given as:

$$(\mathbf{v}^{k+1}, \boldsymbol{\nu}^{k+1}) = \arg\min_{\mathbf{v},\boldsymbol{\nu}} \ \mathcal{L}_{\text{IK}}^{\mathcal{A}}(\mathbf{v}, \boldsymbol{\nu}, \mathbf{z}^k, \mathbf{y}^k, \mathbf{w}^k) \tag{6a}$$
$$\text{s.t.} \quad \mathbf{v}_i = \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i \quad , \forall i \in \mathcal{N}_\text{b}$$
$$\mathbf{v}_0 = \mathbf{0} \tag{6b}$$
$$\mathbf{z}^{k+1} = \arg\min_{\mathbf{z}} \ \mathcal{L}_{\text{IK}}^{\mathcal{A}}(\mathbf{v}^{k+1}, \boldsymbol{\nu}^{k+1}, \mathbf{z}, \mathbf{y}^k, \mathbf{w}^k) \tag{6c}$$
$$\mathbf{y}^{k+1} = \mathbf{y}^k + \mu_{\text{task}}^k(\mathbf{A}\mathbf{v}^{k+1} - \mathbf{b}) \tag{6d}$$
$$\mathbf{w}^{k+1} = \mathbf{w}^k + \mu_{\text{box}}^k(\boldsymbol{\nu}^{k+1} - \mathbf{z}^{k+1}) \tag{6e}$$

Dual updates (6d) and (6e) are cheap to compute. The primal $\mathbf{z}$ update (6c) step boils down to performing a projection of the primal variables onto the constraint set. To make this explicit:

$$\begin{aligned}
\mathbf{z}^{k+1} &= \arg\min_{\mathbf{z}} \ \mathcal{L}^{\mathcal{A}}(\mathbf{v}^{k+1}, \boldsymbol{\nu}^{k+1}, \mathbf{z}, \mathbf{y}^k, \mathbf{w}^k) \\
&= \arg\min_{\mathbf{z}} \ \left\{ \mathbb{I}_{\mathcal{K}_\text{box}}(\mathbf{z}) + \mathbf{w}^{k\top}(\boldsymbol{\nu}^{k+1} - \mathbf{z}) + \frac{\mu_{\text{box}}^k}{2}\|\boldsymbol{\nu}^{k+1} - \mathbf{z}\|_2^2 + \cdots \right\} \\
&= \arg\min_{\mathbf{z}\in\mathcal{K}_\text{box}} \ \left\{ \frac{\mu_{\text{box}}^k}{2}\|\boldsymbol{\nu} + \frac{1}{\mu_{\text{box}}^k}\mathbf{w} - \mathbf{z}\|_2^2 - \frac{\mu_{\text{box}}^k}{2}\|\frac{1}{\mu_{\text{box}}^k}\mathbf{w}\|_2^2 \right\} \\
&= \Pi_{\mathcal{K}_\text{box}}\left(\boldsymbol{\nu} + \frac{1}{\mu_{\text{box}}^k}\mathbf{w}\right)
\end{aligned} \tag{7}$$

where the third equality in (7) comes from completing the square. $\Pi_{\mathcal{K}_\text{box}}$ denotes the constraint projection operator. For the case of box projection, $\Pi_{\mathcal{K}_\text{box}}(\mathbf{x}) := \min(\max(\mathbf{x}, \mathbf{x}_\text{lb}), \mathbf{x}_\text{ub})$.

B. Inverse kinematics as an LQR problem

The main computational bottleneck within the ADMM loop (6) is the primal $\mathbf{v}$ and $\boldsymbol{\nu}$ update step (6a)-(6b), which involves solving an equality-constrained QP problem, which we state explicitly:

$$\min_{\mathbf{v},\boldsymbol{\nu}} \quad \sum_{i=1}^{n_b} \left\{ \frac{1}{2}\mathbf{v}_i^\top\mathbf{H}_i\mathbf{v}_i + \mathbf{p}_i^\top\mathbf{v}_i + \frac{1}{2}\boldsymbol{\nu}_i^\top\mathbf{R}_i\boldsymbol{\nu}_i + \mathbf{r}_i^\top\boldsymbol{\nu}_i \right\} \tag{8a}$$
$$\text{s.t.} \quad \mathbf{v}_i = \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i \quad , \forall i \in \mathcal{N}_\text{b} \tag{8b}$$
$$\mathbf{v}_0 = \mathbf{0} \tag{8c}$$

State-of-the-art QP solvers [43, 3, 23] can already exploit the sparsity structure of the problem quite effectively, and this is usually achieved through an efficient general-purpose sparse linear algebra back-end (such as QDLDL [43], and that of ProxQP [3]). However, leveraging problem-specific sparsity, such as those present in robot's kinematics constraints, offers greater efficiency gains.

Readers familiar with the optimal control literature will recognize problem (8) takes on the form of a Linear-Quadratic-Regulator (LQR) problem. Notice, however, that the LQR "dynamics" equations (8b) are index-shifted in the "control" $\boldsymbol{\nu}_i$. The causal relationship defined by the index-shifted "dynamics" equation implies that in the state-feedback control sense, the feedback "control" $\boldsymbol{\nu}_i$ should be defined using the current "state" $\mathbf{v}_{\pi(i)}$, in the form of

$$\boldsymbol{\nu}_i = \mathbf{k}_i + \mathbf{K}_i\mathbf{v}_{\pi(i)} \tag{9}$$

instead of in the form of $\boldsymbol{\nu}_i = \mathbf{k}_i + \mathbf{K}_i\mathbf{v}_i$ as in the case if the "dynamics" is non-index-shifted. This state-feedback "control" hypothesis (9) will be verified in Sec. III-C as part of the recursion derivation.

Drawing inspiration from the similarity between problem (8) and the standard discrete-time LQR problem, we derive a Riccati-like recursion algorithm in the next subsection for the equality-constrained QP inner sub-problem, within the ADMM loop, by exploiting the kinematic tree-induced sparsity. However, we inform the readers that the next subsection is the densest part of the paper, which may be skipped by readers safely during a quick reading or if they are not interested in the derivation.

C. Recursion Derivation

For better numerical stability, proximal terms on the primal decision variables, $\frac{\rho_\mathbf{v}}{2}\|\mathbf{v}_i - \mathbf{v}_i^\text{prev}\|_2^2$ and $\frac{\rho_\boldsymbol{\nu}}{2}\|\boldsymbol{\nu}_i - \boldsymbol{\nu}_i^\text{prev}\|_2^2$, should be added to the objective function of (8). The added proximal regularization term guarantees that (8) will always have a unique solution, even in the presence of redundant motion constraints (causing $\mathbf{H}_i$s to lose rank when $\mathbf{H}_i^\text{ref}$s are zero) and at kinematic singularities. With the added proximal regularization terms, the quadratic and affine terms in (4) are updated to:

$$\widetilde{\mathbf{H}}_i := \mathbf{H}_i + \rho_\mathbf{v}\mathbf{I} \quad , \quad \widetilde{\mathbf{p}}_i := \mathbf{p}_i - \rho_\mathbf{v}\mathbf{v}_i^\text{prev} \tag{10a}$$
$$\widetilde{\mathbf{R}}_i := \mathbf{R}_i + \rho_\boldsymbol{\nu}\mathbf{I} \quad , \quad \widetilde{\mathbf{r}}_i := \mathbf{r}_i - \rho_\boldsymbol{\nu}\boldsymbol{\nu}_i^\text{prev} \tag{10b}$$

The proximal augmented Lagrangian of problem (8) is given by:

$$\mathcal{L}_{\text{IK}'} := \sum_{i=1}^{n_b} \left\{ \frac{1}{2}\mathbf{v}_i^\top\widetilde{\mathbf{H}}_i\mathbf{v}_i + \widetilde{\mathbf{p}}_i^\top\mathbf{v}_i + \frac{1}{2}\boldsymbol{\nu}_i^\top\widetilde{\mathbf{R}}_i\boldsymbol{\nu}_i + \widetilde{\mathbf{r}}_i^\top\boldsymbol{\nu}_i + \mathbf{f}_i^\top(\mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i - \mathbf{v}_i) \right\} + \mathbf{f}_0\mathbf{v}_0 \tag{11}$$

solving (8) as a primal-dual saddle point problem requires one to solve the following KKT system:

$$\partial_{\mathbf{v}_0}\mathcal{L}_{\text{IK}'} = \mathbf{f}_0 + \sum_{j\in\gamma(0)} \mathbf{f}_j = \mathbf{0} \tag{12a}$$
$$\partial_{\mathbf{f}_0}\mathcal{L}_{\text{IK}'} = \mathbf{v}_0 = \mathbf{0} \tag{12b}$$
$$\partial_{\mathbf{v}_i}\mathcal{L}_{\text{IK}'} = \widetilde{\mathbf{H}}_i\mathbf{v}_i + \widetilde{\mathbf{p}}_i - \mathbf{f}_i + \sum_{j\in\gamma(i)} \mathbf{f}_j = \mathbf{0} \tag{12c}$$
$$\partial_{\boldsymbol{\nu}_i}\mathcal{L}_{\text{IK}'} = \widetilde{\mathbf{R}}_i\boldsymbol{\nu}_i + \widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top\mathbf{f}_i = \mathbf{0} \tag{12d}$$
$$\partial_{\mathbf{f}_i}\mathcal{L}_{\text{IK}'} = \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i - \mathbf{v}_i = \mathbf{0} \tag{12e}$$

where $\gamma(i)$ is the set of descendants of a link i in the kinematic tree. Observe that at the leaf nodes of the kinematic tree, i.e., when $l \in \mathbb{L}$, (with $\mathbb{L}$ denoting the leaf node set of a given kinematic tree) $\gamma(l) = \varnothing$ (because leaf node links have no children links), the dual feasibility equation (12c) reduces to: $\mathbf{f}_l = \widetilde{\mathbf{H}}_l \mathbf{v}_l + \widetilde{\mathbf{p}}_l, \quad \forall l \in \mathbb{L}$. From this observation, we hypothesize that the recursion equation for the dual variable $\mathbf{f}_i$ is:

$$\mathbf{f}_i := \widehat{\mathbf{H}}_i \mathbf{v}_i + \widehat{\mathbf{p}}_i, \quad \forall i \in \mathcal{N}_\text{b} \tag{13}$$

with terminal recursion defined at the leaf nodes of the kinematic tree by $\widehat{\mathbf{H}}_l := \widetilde{\mathbf{H}}_l, \quad \widehat{\mathbf{p}}_l := \widetilde{\mathbf{p}}_l, \quad \forall l \in \mathbb{L}$.

Substituting both the recursion hypothesis (13) and the primal feasibility equation (12e) into the dual feasibility equation (12d) for $\mathbf{f}_i$ and $\mathbf{v}_i$:

$$\begin{aligned}
\widetilde{\mathbf{R}}_i \boldsymbol{\nu}_i + \widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{H}}_i \mathbf{v}_i + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i &= \mathbf{0} \\
\widetilde{\mathbf{R}}_i \boldsymbol{\nu}_i + \widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{H}}_i (\mathbf{v}_{\pi(i)} + \mathbf{S}_i \boldsymbol{\nu}_i) + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i &= \mathbf{0} \\
\boldsymbol{\nu}_i = -\mathbf{D}_i^{-1} \mathbf{S}_i^\top \widehat{\mathbf{H}}_i \ \mathbf{v}_{\pi(i)} - \mathbf{D}_i^{-1} (\widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i)
\end{aligned} \tag{14}$$

where $\mathbf{D}_i := \widetilde{\mathbf{R}}_i + \mathbf{S}_i^\top \widehat{\mathbf{H}}_i \mathbf{S}_i$. Eqn. (14) indicates that $\boldsymbol{\nu}_i$ is in the form of the state-feedback "control" hypothesis proposed in (9), when viewing problem (8) from the LQR perspective.

Substituting the relationship between $\boldsymbol{\nu}_i$ and $\mathbf{v}_{\pi(i)}$, i.e., (14), back into the primal feasibility equation (12e):

$$\begin{aligned}
\mathbf{v}_i &= \mathbf{v}_{\pi(i)} + \mathbf{S}_i (-\mathbf{D}_i^{-1} \mathbf{S}_i^\top \widehat{\mathbf{H}}_i \ \mathbf{v}_{\pi(i)} - \mathbf{D}_i^{-1} (\widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i)) \\
&= (\mathbf{I} - \mathbf{S}_i \mathbf{D}_i^{-1} \mathbf{S}_i^\top \widehat{\mathbf{H}}_i) \ \mathbf{v}_{\pi(i)} - \mathbf{S}_i \mathbf{D}_i^{-1} (\widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i)
\end{aligned} \tag{15}$$

Using this relationship between $\mathbf{v}_i$ and $\mathbf{v}_{\pi(i)}$, the recursion hypothesis (13) becomes:

$$\begin{aligned}
\mathbf{f}_i &= \widehat{\mathbf{H}}_i (\mathbf{I} - \mathbf{S}_i \mathbf{D}_i^{-1} \mathbf{S}_i^\top \widehat{\mathbf{H}}_i) \ \mathbf{v}_{\pi(i)} \\
&\quad - \widehat{\mathbf{H}}_i \mathbf{S}_i \mathbf{D}_i^{-1} (\widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i) + \widehat{\mathbf{p}}_i \\
&= \mathcal{P}_i \ \widehat{\mathbf{H}}_i \ \mathbf{v}_{\pi(i)} + \mathcal{P}_i \widehat{\mathbf{p}}_i - \widehat{\mathbf{H}}_i \mathbf{S}_i \mathbf{D}_i^{-1} \widetilde{\mathbf{r}}_i
\end{aligned} \tag{16}$$

where the projection matrix $\mathcal{P}_i := \mathbf{I} - \widehat{\mathbf{H}}_i \mathbf{S}_i \mathbf{D}_i^{-1} \mathbf{S}_i^\top$ strongly resembles the projection matrix in the articulated-body-algorithm (ABA) [20], except for IK problems, $\widehat{\mathbf{H}}_i$s do not represent a meaningful physical quantity, as opposed to the articulated link inertia in the case of ABA.

To obtain the final recursion equations, (16) is substituted back into the dual feasibility equation (12c) in place of $\mathbf{f}_j$s, and along with the identity $j \in \gamma(i) \Rightarrow \pi(j) = i$:

$$\begin{aligned}
\mathbf{f}_i &= \widetilde{\mathbf{H}}_i \mathbf{v}_i + \widetilde{\mathbf{p}}_i \\
&\quad + \sum_{j \in \gamma(i)} \left\{ \mathcal{P}_j \ \widehat{\mathbf{H}}_j \ \mathbf{v}_{\pi(j)} + \mathcal{P}_j \widehat{\mathbf{p}}_j - \widehat{\mathbf{H}}_j \mathbf{S}_j \mathbf{D}_j^{-1} \widetilde{\mathbf{r}}_j \right\} \\
&= \Big(\widetilde{\mathbf{H}}_i + \sum_{j \in \gamma(i)} \left\{ \mathcal{P}_j \ \widehat{\mathbf{H}}_j \right\} \Big) \mathbf{v}_i \\
&\quad + \widetilde{\mathbf{p}}_i + \sum_{j \in \gamma(i)} \left\{ \mathcal{P}_j \widehat{\mathbf{p}}_j - \widehat{\mathbf{H}}_j \mathbf{S}_j \mathbf{D}_j^{-1} \widetilde{\mathbf{r}}_j \right\}
\end{aligned} \tag{17}$$

thus, the recursion hypothesis for the dual variables $\mathbf{f}_i$ is proven to be correct by induction, with recursion matrices given by:

$$\widehat{\mathbf{H}}_i := \widetilde{\mathbf{H}}_i + \sum_{j \in \gamma(i)} \left\{ \mathcal{P}_j \ \widehat{\mathbf{H}}_j \right\} \tag{18a}$$

$$\widehat{\mathbf{p}}_i := \widetilde{\mathbf{p}}_i + \sum_{j \in \gamma(i)} \left\{ \mathcal{P}_j \widehat{\mathbf{p}}_j - \widehat{\mathbf{H}}_j \mathbf{S}_j \mathbf{D}_j^{-1} \widetilde{\mathbf{r}}_j \right\} \tag{18b}$$

To summarize the recursion strategy for computing $\mathbf{v}$ and $\boldsymbol{\nu}$: Starting at the leaf nodes of the kinematic tree, we initialize $\widehat{\mathbf{H}}_l$ to $\widetilde{\mathbf{H}}_l$ and $\widehat{\mathbf{p}}_l$ to $\widetilde{\mathbf{p}}_l$, for all l in the leaf nodes set $\mathcal{L}$. Then perform a backward pass from leaf nodes down to the root. At each link j, we compute the quantities $\mathcal{P}_j \ \widehat{\mathbf{H}}_j$ and $\mathcal{P}_j \widehat{\mathbf{p}}_j - \widehat{\mathbf{H}}_j \mathbf{S}_j \mathbf{D}_j^{-1} \widetilde{\mathbf{r}}_j$, and add them to the parent joint $\pi(j)$'s $\widehat{\mathbf{H}}_{\pi(j)}$ and, $\widehat{\mathbf{p}}_{\pi(j)}$ respectively. If the link indices are appropriately ordered, i.e. $\forall j \in \mathcal{N}_b, \ \pi(j) < j$, then one single backward pass from leaf nodes to the root will update all $\widehat{\mathbf{H}}_i$s and $\widehat{\mathbf{p}}_i$s. Using them, we can compute the primal variable updates by performing a forward pass on the kinematic tree starting from the root node $i = 0$:

$$\mathbf{v}_0 := \mathbf{0} \tag{19a}$$

for all $i \in \mathcal{N}_b$:

$$\boldsymbol{\nu}_i := -\mathbf{D}_i^{-1} \mathbf{S}_i^\top \widehat{\mathbf{H}}_i \ \mathbf{v}_{\pi(i)} - \mathbf{D}_i^{-1} (\widetilde{\mathbf{r}}_i + \mathbf{S}_i^\top \widehat{\mathbf{p}}_i) \tag{19b}$$

$$\mathbf{v}_i := \mathbf{v}_{\pi(i)} + \mathbf{S}_i \boldsymbol{\nu}_i \tag{19c}$$

The recursion equations derived in (18) strongly resemble that of the ABA [20]. As it has been shown in [40], ABA can be recovered from applying the dynamic programming (DP) principle to the articulated body problem. For a given LQR problem, the recursion equations obtained from applying the DP principle will agree with those obtained from manipulating the KKT matrix associated with the primal-dual saddle point problem. However, one important distinction between these recursion quantities is that, for ABA, quantities equivalent to $\widehat{\mathbf{H}}_i$s and $\widehat{\mathbf{p}}_i$s have physical meanings pertaining to the link inertia and bias terms, while in the case of constrained IK primal sub-problem (8), they capture the effects of both *hard* motion constraints and *soft* tracking constraints on an individual body link.

IV. Low-Complexity Differential Inverse Kinematics: Algorithm

In this section, we will gather the derivations from Section III into a concise algorithmic representation. Additionally, we will provide further details on essential algorithmic components including residual computation, convergence criteria, penalty parameters updates, and feasibility detection. Despite being algorithmic details, these elements are fundamental to ensuring rapid and stable convergence of the algorithm.

We first present the final constrained IK algorithm in Algorithm 1, where the equality-constrained QP IK sub-problem is solved on line 5 using the proposed three-pass recursion Algorithm 2, where the set of topologically ordered link indices is denoted as $\mathfrak{L}$ while $\mathfrak{L}_\text{rev}$ denotes the set with the reversed order.

Algorithm 1 Constrained IK algorithm LOIK

Require: robot model, $\mathbf{q}$, $\mathbf{v}_i^{\text{init}}$s, $\boldsymbol{\nu}^{\text{init}}$, $\mathbf{H}_i^{\text{ref}}$s, $\mathbf{v}_i^{\text{ref}}$s, $\mathbf{A}_i$s, $\mathbf{b}_i$s, $\boldsymbol{\nu}_{\text{lb}}$, $\boldsymbol{\nu}_{\text{ub}}$, $\rho_{\mathbf{v}}$, $\rho_{\boldsymbol{\nu}}$, μ, α^{μ}, σ_{abs}, σ_{rel}, max_iter

$\mathbf{v} = \mathbf{v}^{\text{init}}$; $\boldsymbol{\nu} = \boldsymbol{\nu}^{\text{init}}$; $\mathbf{z} = \mathbf{0}$; $\mathbf{y} = \mathbf{0}$; $\mathbf{w} = \mathbf{0}$
$\mu_{\text{task}} = \alpha^{\mu}\ \mu$; $\mu_{\text{box}} = \mu$
for idx **in range**(max_iter) **do**
  $\mathbf{v}_i^{\text{prev}} \leftarrow \mathbf{v}_i$ and $\boldsymbol{\nu}_i^{\text{prev}} \leftarrow \boldsymbol{\nu}_i$
  $\mathbf{v}, \boldsymbol{\nu} \leftarrow$ **Equality Constrained IK Recursion**
  Constraint Projection
  $\mathbf{z} \leftarrow \Pi_{\mathcal{K}_{\text{box}}}(\boldsymbol{\nu} + \frac{1}{\mu_{\text{box}}^k}\mathbf{w})$
  Dual Update
  $\mathbf{y} \leftarrow \mathbf{y} + \mu_{\text{task}}(\mathbf{A}\mathbf{v} - \mathbf{b})$
  $\mathbf{w} \leftarrow \mathbf{w} + \mu_{\text{box}}(\boldsymbol{\nu} - \mathbf{z})$
  Compute ADMM Residuals
  Check Convergence
  Check Infeasibility
  if Primal Infeasible then
    Infeasibility Tail-Solve
  Update $\boldsymbol{\mu}$

Algorithm 2 Equality-constrained IK recursion

Require: robot model, $\mathbf{q}$, $\mathbf{v}_i$s, $\mathbf{y}$, $\mathbf{w}$, $\mathbf{v}^{\text{prev}}$, $\boldsymbol{\nu}^{\text{prev}}$, $\mathbf{H}_i^{\text{ref}}$s, $\mathbf{v}_i^{\text{ref}}$s, $\mathbf{A}_i$s, $\mathbf{b}_i$s, $\rho_{\mathbf{v}}$, $\rho_{\boldsymbol{\nu}}$, μ_{task}, μ_{box}

Forward Pass 1
for i **in** $\mathfrak{L}$ **do**
  $\widehat{\mathbf{H}}_i \leftarrow \mu_{\text{task}}\mathbf{A}_i^{\top}\mathbf{A}_i + \mathbf{H}_i^{\text{ref}} + \rho_{\mathbf{v}}\mathbf{I}$
  $\widehat{\mathbf{p}}_i \leftarrow \mathbf{A}_i^{\top}\mathbf{y}_i - (\mathbf{H}_i^{\text{ref}})^{\top}\mathbf{v}_i^{\text{ref}} - \mu_{\text{task}}\mathbf{A}_i^{\top}\mathbf{b}_i - \rho_{\mathbf{v}}\mathbf{v}_i^{\text{prev}}$
  $\widetilde{\mathbf{R}}_i \leftarrow (\mu_{\text{box}} + \rho_{\boldsymbol{\nu}})\mathbf{I}$
  $\widetilde{\mathbf{r}}_i \leftarrow \mathbf{w}_i - \mu_{\text{box}}\mathbf{z}_i - \rho_{\boldsymbol{\nu}}\boldsymbol{\nu}_i^{\text{prev}}$
Backward Pass
for i **in** $\mathfrak{L}_{\text{rev}}$ **do**
  $\mathbf{D}_i = \widetilde{\mathbf{R}}_i + \mathbf{S}_i^{\top}\widehat{\mathbf{H}}_i\mathbf{S}_i$; $\mathcal{P}_i = \mathbf{I} - \widehat{\mathbf{H}}_i\mathbf{S}_i\mathbf{D}_i^{-1}\mathbf{S}_i^{\top}$
  $\widehat{\mathbf{H}}_{\pi(i)} \leftarrow \widehat{\mathbf{H}}_{\pi(i)} + \mathcal{P}_i\ \widehat{\mathbf{H}}_i$
  $\widehat{\mathbf{p}}_{\pi(i)} \leftarrow \widehat{\mathbf{p}}_{\pi(i)} + \mathcal{P}_i\widehat{\mathbf{p}}_i - \widehat{\mathbf{H}}_i\mathbf{S}_i\mathbf{D}_i^{-1}\widetilde{\mathbf{r}}_i$
Forward Pass 2
for i **in** $\mathfrak{L}$ **do**
  $\boldsymbol{\nu}_i \leftarrow -\mathbf{D}_i^{-1}\big(\mathbf{S}_i^{\top}\widehat{\mathbf{H}}_i\ \mathbf{v}_{\pi(i)} + \widetilde{\mathbf{r}}_i + \mathbf{S}_i^{\top}\widehat{\mathbf{p}}_i\big)$
  $\mathbf{v}_i \leftarrow \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i$

A. Primal Dual Residual

In the ideal case when the IK problem (2) is both primal and dual feasible, i.e., strong duality holds, and the solution to the Lagrangian saddle point problem via first order optimality conditions will coincide with the solution to the original problem [8]. Therefore, for feasible problems, it makes sense to define convergence criteria through the primal and dual residuals of the saddle point conditions for the Lagrangian of the original constrained inverse kinematics problem, (2). This Lagrangian is given by:

$$\mathcal{L}_{\text{IK}} := \sum_{i=1}^{n_b} \Bigg\{ \frac{1}{2}\|\mathbf{v}_i - \mathbf{v}_i^{\text{ref}}\|_{\mathbf{H}_i^{\text{ref}}}^2 + \mathbf{f}_i^{\top}(\mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i - \mathbf{v}_i) + \mathbf{y}_i^{\top}(\mathbf{A}_i\mathbf{v}_i - \mathbf{b}_i) + \mathbf{w}_i^{\top}(\boldsymbol{\nu}_i - \mathbf{z}_i) + \mathbb{I}_{\mathcal{K}_i}(\mathbf{z}_i) \Bigg\} \tag{20}$$

The saddle point conditions for Lagrangian (20) are presented in Appendix A-A. To check for the convergence of Algorithm 1, one needs to ensure that dual feasibility conditions (31a–31b) and primal feasibility conditions (31e–31g) are satisfied. We define the primal residual as:

$$\varepsilon_{\text{primal}} := \max\Big(\underbrace{\|\mathbf{A}\mathbf{v} - \mathbf{b}\|_{\infty}}_{\varepsilon_{\text{primal},\mathbf{y}}},\ \underbrace{\|\boldsymbol{\nu} - \mathbf{z}\|_{\infty}}_{\varepsilon_{\text{primal},\mathbf{w}}}\Big) \tag{21}$$

note that primal feasibility condition (31e) does not contribute to the primal residual definition because they are automatically satisfied when applying the proposed recursion to the equality-constrained sub-problem. Likewise, we define the dual residual as:

$$\begin{aligned} \varepsilon_{\text{dual},\mathbf{v}} &:= \|\mathbf{H}^{\text{ref}}\mathbf{v} - \mathbf{H}^{\text{ref}^{\top}}\mathbf{v}^{\text{ref}} + \mathbf{A}^{\top}\mathbf{y} + \delta\mathbf{f}\|_{\infty} \\ \varepsilon_{\text{dual},\nu} &:= \|\mathbf{S}^{\top}\mathbf{f} + \mathbf{w}\|_{\infty} \\ \varepsilon_{\text{dual}} &:= \max(\varepsilon_{\text{dual},\mathbf{v}}\ ,\ \varepsilon_{\text{dual},\nu}) \end{aligned} \tag{22}$$

It is worth noting that in the dual residual computation, the Lagrange multipliers $\mathbf{f}_i$s for kinematics constraints are needed. However, since in the formulation of the augmented Lagrangian (3) for the original constrained IK problem, kinematics constraints are omitted, we do not have direct access to these multipliers during the outer ADMM iterations. Instead, we propose to use the multipliers from the equality-constrained sub-problem, for the kinematics constraints, in place of the outer ADMM multiplier for the same constraint. Indeed, after the second forward pass, $\mathbf{f}_i$s can be computed using (13) at minimal computational cost.

Both primal and dual residual go to zero (if the problem is feasible) when Algorithm 1 converges. In practice, however, $\varepsilon_{\text{primal}}$ and $\varepsilon_{\text{dual}}$ are checked against tolerances computed using heuristics such as those proposed by [9] and [43]. Convergence checking in our solver is presented below:

Algorithm 3 Convergence criteria

Require: $\mathbf{v}_i$s, $\boldsymbol{\nu}$, $\mathbf{z}$, $\mathbf{y}$, $\mathbf{w}$, $\mathbf{f}$, $\mathbf{H}_i^{\text{ref}}$s, $\mathbf{v}_i^{\text{ref}}$s, $\mathbf{A}_i$s, $\mathbf{b}_i$s, σ_{abs}, σ_{rel}

$\sigma_{\text{primal}}, \sigma_{\text{dual}} =$ **Tolerance Heuristics**$(\sigma_{\text{abs}}, \sigma_{\text{rel}})$
$\varepsilon_{\text{primal},\mathbf{y}} = \|\mathbf{A}\mathbf{v} - \mathbf{b}\|_{\infty}$; $\varepsilon_{\text{primal},\mathbf{w}} = \|\boldsymbol{\nu} - \mathbf{z}\|_{\infty}$
$\varepsilon_{\text{dual},\mathbf{v}} = \|\mathbf{H}^{\text{ref}}\mathbf{v} - \mathbf{H}^{\text{ref}^{\top}}\mathbf{v}^{\text{ref}} + \mathbf{A}^{\top}\mathbf{y} + \Delta\mathbf{f}\|_{\infty}$
$\varepsilon_{\text{dual},\nu} = \|\mathbf{S}^{\top}\mathbf{f} + \mathbf{w}\|_{\infty}$
$\varepsilon_{\text{primal}} = \max(\varepsilon_{\text{primal},\mathbf{y}}, \varepsilon_{\text{primal},\mathbf{w}})$
$\varepsilon_{\text{dual}} = \max(\varepsilon_{\text{dual},\mathbf{v}}, \varepsilon_{\text{dual},\nu})$
if $\varepsilon_{\text{primal}} < \sigma_{\text{primal}}$ **and** $\varepsilon_{\text{dual}} < \sigma_{\text{dual}}$ **then**
  Converged = True

B. Feasibility Detection

For constrained IK applications, it is often possible that the primal problem (2) is infeasible (e.g., box constraints are too tight to allow the motion constraints to be strictly satisfied). For general convex QP problems, even when the primal problem is feasible, the solution could be unbounded below, causing the dual problem to be infeasible. When such cases are encountered, it is paramount to not only detect infeasibilities in the problem, but also produce solutions with good theoretical guarantees.

To this end, we leverage the theorem of strong alternatives [8]. Following the derivation provided by [5] for infeasibility certificates of general convex QP problems, we specialize them for differential IK problems of the form (2), and summarize the primal infeasibility detection in the following algorithm:

Algorithm 4 Primal infeasibility detection

Require: $\mathbf{f}$, $\mathbf{f}^{\text{prev}}$, $\mathbf{y}$, $\mathbf{y}^{\text{prev}}$, $\mathbf{w}$, $\mathbf{w}^{\text{prev}}$, $\boldsymbol{\nu}_{\text{lb}}$, $\boldsymbol{\nu}_{\text{ub}}$, $\mathbf{A}_i$s, $\mathbf{b}_i$s, σ_{pinf}

$\delta\mathbf{f} = \mathbf{f} - \mathbf{f}^{\text{prev}}$, $\delta\mathbf{y} = \mathbf{y} - \mathbf{y}^{\text{prev}}$, $\delta\mathbf{w} = \mathbf{w} - \mathbf{w}^{\text{prev}}$

$\varepsilon_{\text{pinf}} = \sigma_{\text{pinf}} \cdot \max(\|\delta\mathbf{f}\|_\infty, \|\delta\mathbf{y}\|_\infty, \|\delta\mathbf{w}\|_\infty)$

p1 $:= \| -\delta\mathbf{f}_i + \sum_{j\in\gamma(i)} \delta\mathbf{f}_j + \mathbf{A}_i^\top \delta\mathbf{y}_i\|_\infty \le \varepsilon_{\text{pinf}}, \forall i \in \mathfrak{L}$

p2 $:= \|\mathbf{S}^\top\delta\mathbf{f} + \delta\mathbf{w}\|_\infty \le \varepsilon_{\text{pinf}}$

p3 $:= \mathbf{b}^\top\delta\mathbf{y} \le \varepsilon_{\text{pinf}}$

p4 $:= \boldsymbol{\nu}_{\text{ub}}^\top(\mathbf{w})_+ + \boldsymbol{\nu}_{\text{lb}}^\top(\mathbf{w})_- \le \varepsilon_{\text{pinf}}$

if (**p1 and p2 and p3 and p4**) **then**

  Primal Infeasible = True

For general convex QP problems, dual problem infeasibility must also be checked in addition to primal infeasibility. However, due to the specific problem structure of differential IK of the form (2), where no pure linear terms in the primal decision variables $\mathbf{v}$ and/or $\boldsymbol{\nu}$ appear in the cost, the problem will never become dual infeasible. However, for the sake of completeness, we provide the dual infeasibility checking routine in Appendix A-B, Algorithm 7.

Theoretical guarantees have been provided for solving primal infeasible convex QP problems using Augmented Lagrangian type methods. Particularly, desirable solutions can still be obtained when the primal residual converges (to some value that is the distance of the infeasible problem to the closest feasible problem in the least-squares sense) [4, 14]. Inspired by these theoretical results, we devised a simple tail-solve scheme that, when primal infeasibility is detected, runs tail-solve iterations until primal residual is converged.

Algorithm 5 Infeasibility tail-solve

Require: $\varepsilon_{\text{primal}}$, $\varepsilon_{\text{primal}}^{\text{prev}}$, tail_solve_max_iter, $\sigma_{\text{prim\_tol}}$

for idx **in range**(tail_solve_max_iter) **do**

  if $\|\varepsilon_{\text{primal}} - \varepsilon_{\text{primal}}^{\text{prev}}\| \ge \sigma_{\text{prim\_tol}}$ **then**

    Run Algorithm 1 line 4 to 11

  $\varepsilon_{\text{primal}}^{\text{prev}} := \varepsilon_{\text{primal}}$

C. ADMM Parameter Scheduling

The strategy for updating the ADMM penalty parameter μ from iteration to iteration has a significant impact on the convergence behavior of any ADMM algorithm. It is well-known that fixed-value μs throughout the ADMM iterations often result in slow convergence and other poor solver behavior [9]. Furthermore, [43] suggested using separate μ values for equality and inequality constraints. The μ-update at each ADMM iteration takes on the form of line 2 in Algorithm 1. The reasoning behind such update scheme is that the optimal penalty for any active constraints should be $\mu = \infty$, and for any inactive constraints $\mu = 0$. At convergence, all equality constraints must all be active, therefore we wish to assign large values of μ to equality constraints and help drive primal residual to zero. In addition to separate penalty parameters for equality and inequality constraints, we would also like to keep primal and dual residual decreasing in a uniform manner, that is, if the primal residual is decreasing too fast, we would like the solver to place more emphasis on reducing the dual residual, and vice versa. This can be achieved by increasing or decreasing μ based on the ratio of primal and dual residual. Large values of μ will put more emphasis on the equality constraints in the problem and help reducing primal residual, while small values of μ will put more emphasis on reducing the dual residual. These heuristics for updating μ are presented below:

Algorithm 6 ADMM parameter update

Require: μ, α^μ, σ^μ, $\varepsilon_{\text{primal}}$, $\varepsilon_{\text{dual}}$

if $\varepsilon_{\text{primal}} \ge \sigma^\mu \cdot \varepsilon_{\text{dual}}$ **then**

  $\mu \leftarrow \mu * \sigma^\mu$

else if $\varepsilon_{\text{dual}} \ge \sigma^\mu \cdot \varepsilon_{\text{primal}}$ **then**

  $\mu \leftarrow \mu * \frac{1}{\sigma^\mu}$

$\mu_{\text{task}} = \alpha^\mu \cdot \mu$; $\mu_{\text{box}} = \mu$

D. Complexity Analysis

We analyze the algorithmic complexity of the proposed constrained IK Algorithm 1. Starting with the three-pass recursion Algorithm 2 for the equality-constrained sub-problem. For each pass over the kinematic tree, quantities at each link must be computed. It is straightforward to see that every quantity computed during the **Backward Pass** and **Forward Pass 2** has a fixed computational cost. Therefore, **Backward Pass** and **Forward Pass 2** will each require $O(n_b)$ number of operations to complete, where n_b is the number of links in the system. For the **Forward Pass 1**, every calculation except those on line 3 and 4 have a fixed computational cost for each link. The cost of computing quantities on line 3 and 4 varies depending on the dimension of the motion constraint on a specific link. However, on links where no motion constraints are defined, the cost of computing them are fixed. Therefore, **Forward Pass 1** requires at most $O(n_c)$ + $O(n_b)$ number of operations to complete, where n_c is the number of links with motion constraints

defined. All together, the three-pass recursion algorithm takes up to $O(n_b + n_c)$ number of operations to complete.

The three-pass recursion only gets called once every ADMM iteration, and the solver will converge/terminate in at most **max_iter** number of iterations. Furthermore, all the remaining computations within a single ADMM iteration require a fixed number of operations. Therefore, we can conclude that the proposed constrained IK Algorithm 1, has a complexity of $O(n_b + n_c)$, that is to say that it scales linearly with respect to the number of links in the system and the number of motion constraints defined in the IK problem. The derived complexity claim is further verified on benchmark problems of varying scale in Sec. V.

V. Experimental Validation and Benchmarks

We evaluate the performance of differential IK solvers in a benchmark of inverse kinematics scenarios, which we plan to release as open source software after peer-review of this work [2]. We build upon the `robot_descriptions.py` project [12] that indexes robot descriptions publicly available under open source licenses and provides a Python library to load them directly into robotics software. The index includes a variety of robot structures, including arms, mobile-base, quadruped and humanoid structures, with a present total of 78 descriptions. While the aim of the public benchmark will be to encompass all available descriptions and enable new users to add their own scenarios, as an initial stage in this work we focus on two structures: arms (11 descriptions) and humanoids (12 descriptions) for a total of 23 scenarios. Scenarios are defined as task target trajectories that evolve independently from the robot configuration. We design them to reproduce realistic applications for each kind of robot. Table I lists these robot descriptions and their degrees of freedom (DOF).

A. Robotic arm scenarios

Scenarios for arms consist of a single task on the end-effector pose traveling back and forth between two reference configurations, as in *e.g.,* traveling phases in pick-and-place scenarios. The target interpolates on a straight-line path in $\mathbb{R}^3 \times SO(3)$ rather than on $SE(3)$ [31, Section 9.2.1], with positions in the inertial frame going back and forth between two values $\mathbf{p}_0$ and $\mathbf{p}_1$, scaled by a geometric factor to adapt to the dimensions of each arm:

$$\mathbf{p}_i^*(t) = \mathit{scale} \times \left[\mathbf{p}_0 + \sin(t)^2(\mathbf{p}_1 - \mathbf{p}_0)\right] \tag{23}$$

Orientations $\mathbf{R}_i^*(t)$ interpolate similarly between two boundary values $\mathbf{R}_0$ and $\mathbf{R}_1$. The tuple $(\mathbf{R}_i^*(t), \mathbf{p}_i^*(t))$ thus defines a target $\mathbf{T}_i^*(t) \in SE(3)$ for the end-effector task i. We place robotic arms at the origin of the initial frame for the scaling factor to make sense. Across all scenarios, this factor ranges from 8% for the smallest arm (Poppy Ergo Jr) to 220% for the largest one (M-710iC).

[2]Benchmark videos will be made available at https://simple-robotics.github.io/publications/low-order-ik/

Table I: Robot descriptions included in benchmark scenarios.

Robot	Type	DOF (△)
e.DO	Arm	6
M-710iC	Arm	6
Gen2	Arm	6
Gen3	Arm	6
Poppy Ergo Jr	Arm	6
UR3	Arm	6
UR5	Arm	6
UR10	Arm	6
Z1	Arm	6
iiwa	Arm	7
Panda	Arm	8
SigmaBan	Humanoid	26
Draco3	Humanoid	31
Atlas v3 (DRC)	Humanoid	36
Atlas v4	Humanoid	36
iCub	Humanoid	38
TALOS	Humanoid	38
JVRC-1	Humanoid	40
Romeo	Humanoid	43
JAXON	Humanoid	44
Robonaut 2	Humanoid	62
ergoCub	Humanoid	63
Valkyrie	Humanoid	65

B. Humanoid scenarios

Scenarios for humanoids reproduce linear inverted pendulum tracking [27] where a center-of-mass trajectory is followed by inverse kinematics alongside foot trajectories. We reproduce the feedforward part of an optimization-based formulation of that approach [10, Fig. 2], with a center-of-mass trajectory produced by open-loop model predictive control [48]. Each benchmark scenario then consists of three tasks:

- position $\mathbf{p}_{com}^*(t)$ for the upper-body target where the robot should place its center of mass (as in [27], we use a fixed location in the torso frame calculated from the center of mass position in standing configuration), and
- poses $\mathbf{T}_{lf}^*(t)$ and $\mathbf{T}_{rf}^*(t)$ for the left and right feet, alternating between contact and swing phases.

Humanoids walk forward in all scenarios, with a stationary center-of-mass height computed from the robot's initial configuration. Step length and swing foot height are adjusted for each description, ranging from $(20\ \mathrm{cm}, 5\ \mathrm{cm})$ for the smallest humanoid (SigmaBan) to $(50\ \mathrm{cm}, 15\ \mathrm{cm})$ for the largest one (Atlas v3).

C. Inverse kinematics formulation

We implement Loik in C++ using the Pinocchio library [13] for rigid-body dynamics and Eigen [22] for linear algebra. Our benchmark unrolls target trajectories for each scenario, with a time step $\delta t = 5$ ms and a trajectory duration of 10 seconds, large enough to yield two thousand IK problems per scenario. We use the Pink library [11] to formulate first-order differential inverse kinematics [32, 41] with tasks. A task consists of two components: a target, as detailed in the latter two sections for the scenarios in this benchmark, and dynamics. For a task over frame i with target $\mathbf{T}_i^*$ and whose transform in the current configuration $\mathbf{q} \in \mathcal{C}$ is $\mathbf{T}_i(\mathbf{q})$, the task dynamics computes a

velocity expressed locally in F as:

$$\mathbf{v}_i^*(\mathbf{q}) = \frac{k_p}{\delta t} \log(\mathbf{T}_i^{-1}(\mathbf{q})\mathbf{T}_i^*) \tag{24}$$

with $k_p \in (0, 1)$ the gain parameter indicating how much we trust the local linearization of the nonlinear kinematics.

LOIK works in maximal coordinates, and we can therefore forward this task velocity directly as a motion constraint (2c) $\mathbf{A}_i\mathbf{v}_i = \mathbf{b}_i$ with:

$$\mathbf{A}_i^{\text{LOIK}} = \mathbf{I}_6 \qquad \mathbf{b}_i^{\text{LOIK}} = \mathbf{v}_i^*(\mathbf{q}) \tag{25}$$

QP-based approaches work in minimal coordinates, meaning their motion constraints are expressed as $\mathbf{A}_i\boldsymbol{\nu} = \mathbf{b}_i$. They solve the same underlying problem by computing the Jacobian matrix $\mathbf{J}_i(\mathbf{q})$ of the frame at the current configuration, and setting:

$$\mathbf{A}_i^{\text{QP}} = \mathbf{J}_i(\mathbf{q}) \qquad \mathbf{b}_i^{\text{QP}} = \mathbf{v}_i^*(\mathbf{q}) \tag{26}$$

In multitask scenarios, the quadratic programming formulation casts all tasks into a single cost:

$$\underset{\boldsymbol{\nu}}{\text{minimize}} \sum_i \|\mathbf{A}_i^{\text{QP}}\boldsymbol{\nu} - \mathbf{b}_i^{\text{QP}}\|^2_{\mathbf{H}_i^{\text{QP}}} \tag{27}$$

In multitask humanoid scenarios, we apply equal weights $\mathbf{H}_i^{\text{QP}} = \mathbf{I}_6$, meaning all tasks have equal importance and an 1 rad orientation error is penalized on par with a 1 m position error. Following the steps (24), (26), (27), QP matrices and vectors are uniquely defined from task targets and the current robot configuration (see *e.g.* the documentation of [11] for further details).

For another point of reference to the state of the art, we also consider the quadratic programming formulation from DRAKE [45] where an additional optimization variable α allows for an optional downscale of the velocities from (24):

$$\underset{\boldsymbol{\nu},\alpha}{\text{minimize}} \sum_i \|\mathbf{P}(\boldsymbol{\nu} - K(\mathbf{q}^* \ominus \mathbf{q}))\|^2 - 100\alpha \tag{28}$$

$$\text{s.t.} \forall i, \mathbf{A}_i^{\text{QP}}\boldsymbol{\nu} = \alpha\mathbf{b}_i^{\text{QP}} \tag{29}$$

$$0 \leq \alpha \leq 1 \tag{30}$$

where $\ominus$ is the configuration-space difference operator (reciprocal to the integrator $\mathbf{q}_{t+\delta t} = \mathbf{q}_t \oplus (\boldsymbol{\nu}\delta t)$), $\mathbf{q}^*$ is the neutral configuration and we set $K = 10^{-6}$ Hz. The rows of $\mathbf{P}$ form an orthonormal basis for the nullspace of the matrix $\mathbf{A}^{\text{QP}}$ stacking all $\{\mathbf{A}_i^{\text{QP}}\}$ vertically (computed in DRAKE by an additional LU decomposition). See [45] for details.

During the rollout phase, our benchmark starts from a prescribed initial configuration $\mathbf{q}_0$ and integrates $\mathbf{q}_{t+\delta t} = \mathbf{q}_t \oplus (\boldsymbol{\nu}\delta t)$ where $\boldsymbol{\nu}$ is the solution computed by LOIK or QP. We perform one rollout integrating solutions from one method (calling both methods on each configuration $\mathbf{q}_t$) and one rollout integrating solutions from the other.

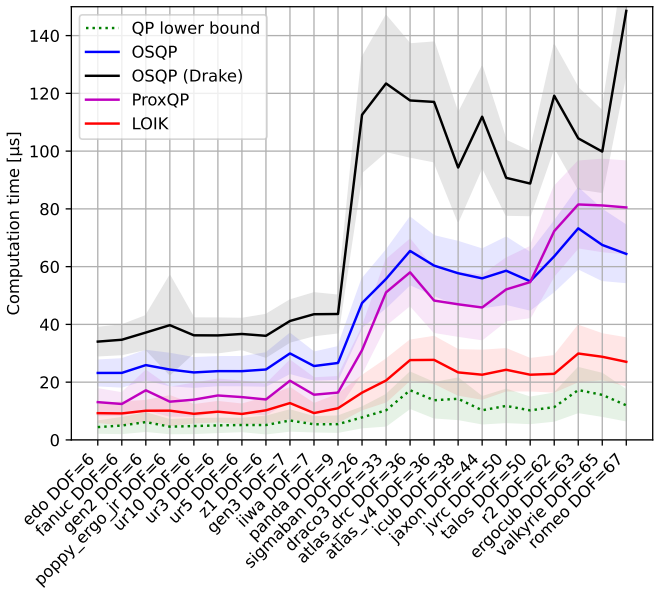

Figure 7: Computation times for each method, on average over the 10 seconds of motion of each scenario (x-axis) with confidence intervals of ± one standard deviation. The additional "QP lower bound" measures the time taken to compute frame placement and Jacobian matrices, a prerequisite-requisite to build QP matrices.

D. Comparison to QP-based inverse kinematics

With the parameters we have described, the benchmark produces 92,000 IK problems. We solve each problem with LOIK, either OSQP or ProxQP run on the same quadratic program (27), and OSQP run on the DRAKE quadratic program (28)–(30), denoted by "OSQP (Drake)". We leave QP solvers with their respective default settings and tolerances. The resulting computation times are reported in Figure 7.

a) Computation times: we did not include the overhead in measuring computation times due to Python instructions. We implemented LOIK as a single C++ function call with low binding overhead. For QP formulations, computation times include (1) the time taken to build problem matrices and (2) the time the solver takes to return a solution. We only counted in (1) the time taken to compute frame placements and Jacobian matrices, which are implemented in C++ in the same framework as LOIK.[3] Since computing these quantities is a prerequisite-requisite to build an IK quadratic program, we include these timings as well as "QP lower bound" in Figure 7. For (2), we used timings computed internally by each solver. Both decisions yield lower bounds on actual timings for the QP-based approach.

We observe in these results that, while LOIK is 1.5–2× faster than QP-based approaches on single-task arm scenarios, it scales more favorably when moving on to multi-task humanoid ones, where its computation times are 2–3× shorter. LOIK scales essentially like the "QP lower bound" of frame Jacobian computations (another linear-time algorithm), with

[3]This means in particular that, for "OSQP (Drake)", (1) does not include the time taken to compute the additional LU decomposition.

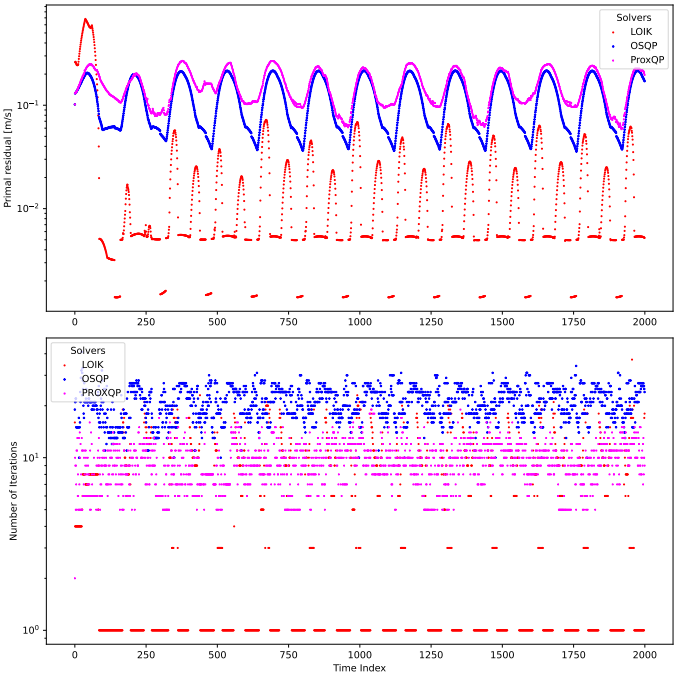

Figure 8: Comparison of solution quality from LOIK, OSQP, and ProxQP for the 67-DOF Romeo humanoid scenario. Top: primal residual at each time step. Bottom: number of iterations each solver took to converge at each time step.

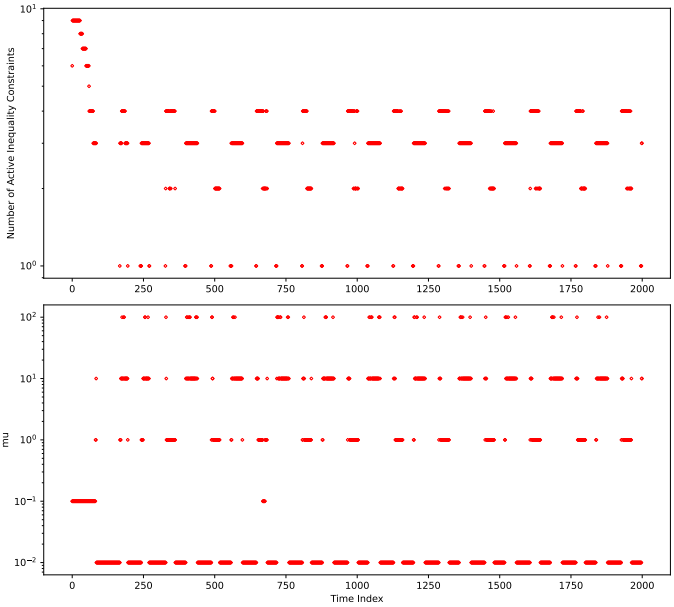

Figure 9: Additional solver information from LOIK for the 67-DOF Romeo humanoid scenario. Top: number of active inequality constraints at termination for each time step. Bottom: value of the ADMM parameter μ at termination for each time step.

roughly a factor of two.

b) Residuals: We take a closer look at the performance of LOIK, OSQP and ProxQP on the 67-DOF Romeo humanoid scenario, which is one of the most challenging of the benchmark. We look at the primal residual $\max_i(\|\mathbf{A}_i\boldsymbol{\nu} - \mathbf{b}_i\|_\infty)$ for tracking tasks over the entire trajectory, as well as the number of iterations required by each solver to converge. These comparisons are depicted in Fig. 8. In this scenario, LOIK converges to a solution in less iterations than ProxQP and OSQP. The hyperparameters used with LOIK are reported in Sec. B. For OSQP and ProxQP, the default hyperparameters were used, with one exception for OSQP where we enforced checking termination at every iteration rather than the default every 25 iterations. A major reason for LOIK requiring fewer iterations was found to be our relative scaling factor of $\alpha^\mu = 10^4$, which weighs equality constraints higher than inequality constraints. This appears to be more effective than OSQP's approach of not weighting the equality and inequality constraints differently.

To inspect further how the method performs, Fig. 9 lists the number of active inequality constraints and the value of the ADMM penalty parameter at the time of termination for LOIK over the same Romeo humanoid scenario. In our benchmark, this scenario is among those with the highest counts of active inequality constraints, with three or more constraints active most of the time. We check in particular that the ADMM penalty parameter μ remains within a reasonable range even on the infeasible problems at the beginning of the trajectory, owing to the infeasibility detection and termination strategy discussed in Sec. IV-B.

E. Limitations

While we have assessed the effectiveness of LOIK over a wide range of robots, we note that, at present, its expressivity presents a couple of limitations. First, LOIK does not support robot topologies with internal closed loops, as its recursive derivation relies on a tree topology. Supporting closed-loops is a relevant future research direction, since several recent robots include them to improve some mechanical properties. Second, LOIK assumes that all task constraints involve a single link each, thus not supporting tasks that involve multiple links at once. This precludes us from, for instance, constraining the sum of the velocity of two links or constraining the velocity of the center of mass, which is the inertia-weighted sum of all link velocities (that is why we approximated the center of mass by a fixed upper-body target in humanoid scenarios).

VI. Conclusion

In this paper, we have proposed a linear complexity differential inverse kinematics solver, LOIK, which is capable of handling both linear equality and inequality constraints. This was only made possible by effectively exploiting the kinematic tree-induced sparsity via the proposed three-pass recursion algorithm that solves the equality-constrained subproblem within an ADMM loop. The ADMM formulation enabled inequality constraints to be integrated into the solver at minimal cost, as these constraints are resolved via constraint set projection during each ADMM update. We have also demonstrated the speed and scalability of our algorithm when compared to the current state-of-the-art QP solvers on real-world IK problems across a wide range of robotics platforms with varying kinematic complexity. We expect this work will contribute to drastically reducing the computational burden of

existing IK-based controllers, thus facilitating their usage in applications where computational resources are limited. Other interesting future directions include integrating LOIK within larger control architectures for computing global IK [6] and constrained motion planning [7], as well as exploring the natural parallelization opportunities that ADMM provides.

Acknowledgements

This work was supported in part by the French government under the management of Agence Nationale de la Recherche (ANR) as part of the "Investissements d'avenir" program, references ANR-19-P3IA-0001 (PRAIRIE 3IA Institute) and ANR-22-CE33-0007 (INEXACT), the European project AGIMUS (Grant 101070165), the Louis Vuitton ENS Chair on Artificial Intelligence and the Casino ENS Chair on Algorithmic and Machine Learning. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Erwin Aertbeliën and Joris De Schutter. etasl/etc: A constraint-based task specification language and robot controller using expression graphs. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1540–1546. IEEE, 2014.

[2] Dae-Sung Bae and Edward J Haug. A recursive formulation for constrained mechanical system dynamics: Part i. open loop systems. *Journal of Structural Mechanics*, 15(3):359–382, 1987.

[3] Antoine Bambade, Fabian Schramm, Sarah El Kazdadi, Stéphane Caron, Adrien Taylor, and Justin Carpentier. PROXQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond. working paper or preprint, September 2023. URL https://inria.hal.science/hal-04198663.

[4] Antoine Bambade, Fabian Schramm, Adrien Taylor, and Justin Carpentier. Qplayer: efficient differentiation of convex quadratic optimization. *International Conference on Learning Representations (ICLR)*, 2024.

[5] Goran Banjac, Paul Goulart, Bartolomeo Stellato, and Stephen Boyd. Infeasibility detection in the alternating direction method of multipliers for convex optimization. *Journal of Optimization Theory and Applications*, 183: 490–519, 2019.

[6] Patrick Beeson and Barrett Ames. Trac-ik: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 928–935. IEEE, 2015.

[7] Dmitry Berenson, Siddhartha Srinivasa, and James Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[8] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[9] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[10] Stéphane Caron, Abderrahmane Kheddar, and Olivier Tempier. Stair climbing stabilization of the hrp-4 humanoid robot using whole-body admittance control. In *2019 International conference on robotics and automation (ICRA)*, pages 277–283. IEEE, 2019.

[11] Stéphane Caron, Yann De Mont-Marin, Rohan Budhiraja, and Seung Hyeon Bang. Pink: Python inverse kinematics based on Pinocchio, January 2024. URL https://github.com/stephane-caron/pink.

[12] Stéphane Caron, Giulio Romualdi, Lev Kozlov, Daniel Ordonez, Hugo Tadashi Kussaba, and Seung Hyeon Bang. robot_descriptions.py: Robot descriptions in Python, January 2024. URL https://github.com/robot-descriptions/robot_descriptions.py.

[13] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*, 2019.

[14] A. Chiche and J.C. Gilbert. How the augmented lagrangian algorithm can deal with an infeasible convex quadratic optimization problem. *Journal of Convex Analysis*, 2016.

[15] Alberto De Marchi. On a primal-dual newton proximal method for convex quadratic programs. *Preprint*, 12 2020. Submitted.

[16] Rosen Diankov. *Automated construction of robotic manipulation programs*. PhD thesis, Carnegie Mellon University, The Robotics Institute Pittsburgh, 2010.

[17] Dimitar Dimitrov, Alexander Sherikov, and Pierre-Brice Wieber. Efficient resolution of potentially conflicting linear constraints in robotics. Submitted to IEEE TRO (05/August/2015), August 2015. URL https://inria.hal.science/hal-01183003.

[18] Adrien Escande, Nicolas Mansard, and Pierre-Brice Wieber. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. *The International Journal of Robotics Research*, 33(7):1006–1028, 2014.

[19] Roy Featherstone. The calculation of robot dynamics using articulated-body inertias. *The international journal of robotics research*, 2(1):13–30, 1983.

[20] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[21] R. Glowinski and A. Marroco. Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de probl'emes de Dirichlet non linéaires. *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(R2):41–76, 1975.

[22] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[23] B. Hermans, A. Themelis, and P. Patrinos. QPALM: A Newton-type Proximal Augmented Lagrangian Method for Quadratic Programs. In *58th IEEE Conference on Decision and Control*, Dec. 2019.

[24] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:302–320, 1969.

[25] Taylor A Howell, Brian E Jackson, and Zachary Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019.

[26] Wilson Jallet, Antoine Bambade, Nicolas Mansard, and Justin Carpentier. Constrained differential dynamic programming: A primal-dual augmented lagrangian approach. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13371–13378. IEEE, 2022.

[27] Shuuji Kajita, Mitsuharu Morisawa, Kanako Miura, Shin'ichiro Nakaoka, Kensuke Harada, Kenji Kaneko, Fumio Kanehiro, and Kazuhito Yokoi. Biped walking stabilization based on linear inverted pendulum tracking. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4489–4496. IEEE, 2010.

[28] Oussama Kanoun, Florent Lamiraux, and Pierre-Brice Wieber. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. *IEEE Transactions on Robotics*, 27(4):785–792, 2011.

[29] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous robots*, 40:429–455, 2016.

[30] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5 (47):eabc5986, 2020.

[31] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017.

[32] Yoshihiko Nakamura. *Advanced robotics: redundancy and optimization*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[33] Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 2006.

[34] Je P Popov, Anatolij Fedorovic Vereshchagin, and Stanislav Leonidovic Zenkevic. *Manipuljacionnyje roboty: Dinamika i algoritmy*. Nauka, 1978.

[35] M. J. D. Powell. A method for nonlinear constraints in minimization problems. *Optimization*, page 283–298, 1969.

[36] James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

[37] R. Tyrrell Rockafellar and Roger J. B. Wets. *Variational Analysis*. Springer Berlin, Heidelberg, 1997.

[38] Layale Saab, Nicolas Mansard, François Keith, Jean-Yves Fourquet, and Philippe Souères. Generation of dynamic motion for anthropomorphic systems under prioritized equality and inequality constraints. In *2011 IEEE International Conference on Robotics and Automation*, pages 1091–1096. IEEE, 2011.

[39] Ajay Suresha Sathya and Justin Carpentier. Constrained articulated body dynamics algorithms.

[40] Ajay Suresha Sathya, Herman Bruyninckx, Wilm Decré, and Goele Pipeleers. Efficient constrained dynamics algorithms based on an equivalent lqr formulation using gauss' principle of least constraint. *IEEE Transactions on Robotics*, 2023.

[41] Jean-Jacques Slotine and B Siciliano. A general framework for managing multiple tasks in highly redundant robotic systems. In *proceeding of 5th International Conference on Advanced Robotics*, volume 2, pages 1211–1216, 1991.

[42] Ruben Smits, H Bruyninckx, and E Aertbeliën. Kdl: Kinematics and dynamics library, 2011.

[43] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[44] Tomomichi Sugihara. Solvability-unconcerned inverse kinematics based on levenberg-marquardt method with robust damping. In *2009 9th IEEE-RAS international conference on humanoid robots*, pages 555–560. IEEE, 2009.

[45] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

[46] AF Vereshchagin. Computer simulation of the dynamics of complicated mechanisms of robot-manipulators. *Eng. Cybernet.*, 12:65–70, 1974.

[47] Anatolii Fedorovich Vereshchagin. Modeling and control of motion of manipulational robots. *Soviet Journal of Computer and Systems Sciences*, 27(5):29–38, 1989.

[48] Pierre-Brice Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 137–142. IEEE, 2006.

[49] Pierre-Brice Wieber, Adrien Escande, Dimitar Dimitrov, and Alexander Sherikov. Geometric and numerical aspects of redundancy. *Geometric and Numerical Foundations of Movements*, pages 67–85, 2017.

APPENDIX A
LOIK ALGORITHMIC DETAILS

A. Lagrangian Saddle Point Conditions

To derive the saddle point conditions for Lagrangian (20), we note that for the differential IK problem of the form (2), the quadratic objective function is smooth (therefore continuous), and all the linear equality constraints form a convex constrained set; additionally, the inequality constraint sets $\mathcal{K}_i$ are all assumed to be convex, under these conditions, the saddle point conditions for the Lagrangian (20) is provided by [37] [Thm 6.12], [Cor 11.51], which in turn provides the first order optimality conditions of the original problem (when it is feasible):

$$-\partial_{\mathbf{v}_i}\mathcal{L}_{\text{IK}} = -\left(\mathbf{H}_i^{\text{ref}}\mathbf{v}_i - \mathbf{H}_i^{\text{ref}^\top}\mathbf{v}_i^{\text{ref}} + \mathbf{A}_i^\top\mathbf{y}_i - \mathbf{f}_i + \sum_{j\in\gamma(i)}\mathbf{f}_j\right) = \mathbf{0} \tag{31a}$$

$$-\partial_{\boldsymbol{\nu}_i}\mathcal{L}_{\text{IK}} = -\left(\mathbf{S}_i^\top\mathbf{f}_i + \mathbf{w}_i\right) = \mathbf{0} \tag{31b}$$

$$-\partial_{\mathbf{z}_i}\mathcal{L}_{\text{IK}} = \mathbf{w}_i \in \mathcal{N}_{\mathcal{K}_i}(\mathbf{z}_i) \tag{31c}$$

$$\mathbf{z}_i \in \mathcal{K}_i \tag{31d}$$

$$\partial_{\mathbf{f}_i}\mathcal{L}_{\text{IK}} = \mathbf{v}_{\pi(i)} + \mathbf{S}_i\boldsymbol{\nu}_i - \mathbf{v}_i = \mathbf{0} \tag{31e}$$

$$\partial_{\mathbf{y}_i}\mathcal{L}_{\text{IK}} = \mathbf{A}_i\mathbf{v}_i - \mathbf{b}_i = \mathbf{0} \tag{31f}$$

$$\partial_{\mathbf{w}_i}\mathcal{L}_{\text{IK}} = \boldsymbol{\nu}_i - \mathbf{z}_i = \mathbf{0} \tag{31g}$$

We now show that using the ADMM updates proposed in (6c) and (6e) (for box constraint sets), first-order optimality conditions (31c) and (31d) will be automatically satisfied. Verifying (31d) is straightforward, as (7) has shown that primal z-update (6c) boils down to a projection of $\mathbf{z}_i$ onto the constraint set $\mathcal{K}_i$, therefore $\mathbf{z}_i^{k+1} \in \mathcal{K}_i$ after each ADMM iteration. For optimality condition (31c), note that when $\mathcal{K}_i$s are box constraints, for any element in the interior of $\mathcal{K}_i$, $\mathcal{N}_{\mathcal{K}_i}(\mathbf{z}_i)$ is the zero cone, i.e. $\mathcal{N}_{\mathcal{K}_i}(\mathbf{z}_i) = \{\mathbf{0}\}$; for any element that has components on the lower boundary of $\mathcal{K}_i$, i.e., $\mathbf{z}_i[j] = \mathbf{z}_{\mathbf{lb}}[j]$ the normal cone $\mathcal{N}_{\mathcal{K}_i}(\mathbf{z}_i)$ consists of elements, n, whose jth component is non-positive $n[j] \leq 0$; likewise, for any element that has components on the upper boundary of $\mathcal{K}_i$, i.e. $\mathbf{z}_i[j] = \mathbf{z}_{\mathbf{ub}}[j]$, the normal cone $\mathcal{N}_{\mathcal{K}_i}(\mathbf{z}_i)$ consists of elements, n, whose jth component is non-negative $n[j] \geq 0$. When $\mathbf{z}_i^{k+1}$ is strictly in the interior of $\mathcal{K}_i$, $\mathbf{z}_i^{k+1} = \Pi_{\mathcal{K}_i}\left(\boldsymbol{\nu}_i^{k+1} + \frac{1}{\mu_{\text{box}}^k}\mathbf{w}_i^k\right) = \boldsymbol{\nu}_i^{k+1} + \frac{1}{\mu_{\text{box}}^k}\mathbf{w}_i^k$. Substituting back into $\mathbf{w}_i$-update (6e): $\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + \mu_{\text{box}}^k(\boldsymbol{\nu}_i^{k+1} - (\boldsymbol{\nu}_i^{k+1} + \frac{1}{\mu_{\text{box}}^k}\mathbf{w}_i^k)) = \mathbf{0}$. Which proves that $\mathbf{w}_i^{k+1} \in \mathcal{N}_{\mathcal{K}_i}(\mathbf{z}_i^{k+1})$ when $\mathbf{z}_i^{k+1}$ is strictly in the interior of $\mathcal{K}_i$. The other two cases where $\mathbf{z}_i^{k+1}$ has components on the boundaries of $\mathcal{K}_i$ can be similarly verified.

B. Dual Infeasibility Detection

Below, we present the dual infeasibility detection algorithm for differential IK problems of the form (2)

Algorithm 7 Dual infeasibility detection

Require: $\mathbf{v}_i$s, $\mathbf{v}_i^{\text{prev}}$s, $\boldsymbol{\nu}$, $\boldsymbol{\nu}^{\text{prev}}$, $\mathbf{A}_i$s, $\mathbf{H}_i^{\text{ref}}$s, $\mathbf{v}_i^{\text{ref}}$s, σ_{dinf}

$\delta\mathbf{v} = \mathbf{v} - \mathbf{v}^{\text{prev}}$, $\delta\boldsymbol{\nu} = \boldsymbol{\nu} - \boldsymbol{\nu}^{\text{prev}}$

$\varepsilon_{\text{dinf}} = \sigma_{\text{dinf}} \cdot \max(\|\delta\mathbf{v}\|_\infty, \|\delta\boldsymbol{\nu}\|_\infty)$

d1 $:= \|\mathbf{H}^{\text{ref}}\delta\mathbf{v}\|_\infty \leq \varepsilon_{\text{dinf}}$

d2 $:= \|\mathbf{v}^{\text{ref}^\top}\mathbf{H}^{\text{ref}}\delta\mathbf{v}\|_\infty \leq \varepsilon_{\text{dinf}}$

d3 $:= -\varepsilon_{\text{dinf}} \preccurlyeq \delta\mathbf{v}_{\pi(i)} + \mathbf{S}_i\delta\boldsymbol{\nu}_i - \delta\mathbf{v}_i \preccurlyeq \varepsilon_{\text{dinf}}, \forall i \in \mathfrak{L}$

d4 $:= -\varepsilon_{\text{dinf}} \preccurlyeq \mathbf{A}\delta\mathbf{v} \preccurlyeq \varepsilon_{\text{dinf}}$

d5 $:= -\varepsilon_{\text{dinf}} \preccurlyeq \delta\boldsymbol{\nu} \preccurlyeq \varepsilon_{\text{dinf}}$

if (**d1 and d2 and d3 and d4 and d5**) **then**

  Dual Infeasible = True

APPENDIX B
LOIK HYPERPARAMETER VALUES

Table II stores hyperparameter values used within LOIK to generate all experimental results presented in this paper.

Table II: LOIK hyperparameter values for all experiments

Parameter	Value
$\rho_{\mathbf{v}}$	1e-5
$\rho_{\boldsymbol{\nu}}$	1e-5
μ	1e-2
α^μ	1e4
σ_{abs}	1e-3
σ_{rel}	1e-3
σ_{pinf}	1e-2
σ_{dinf}	1e-2
max_iter	100
tail_solve_max_iter	100

APPENDIX C
RANDOM INITIALIZATION AND WARM-STARTING

To quantify LOIK's performance characteristics against initial guesses, we tested both warm-starting and random initialization strategies at the most difficult problem for each robot IK scenarios. This is achieved by selecting a time instant during the tracking trajectory for each robot at which the highest number of iterations is required. At these time instances, we provided poor initial guesses for each decision variable, sampled from a uniform distribution supported over [-1e5, 1e5]. We then performed this random sampling a thousand times for each robot scenario at these most difficult problems, and collected the average number of iterations (and standard deviation) required into Figure 11. Additionally, we also gathered the number of active inequality constraints at these time instances for each robot into Figure 12 to verify that these problems are indeed non-trivial.

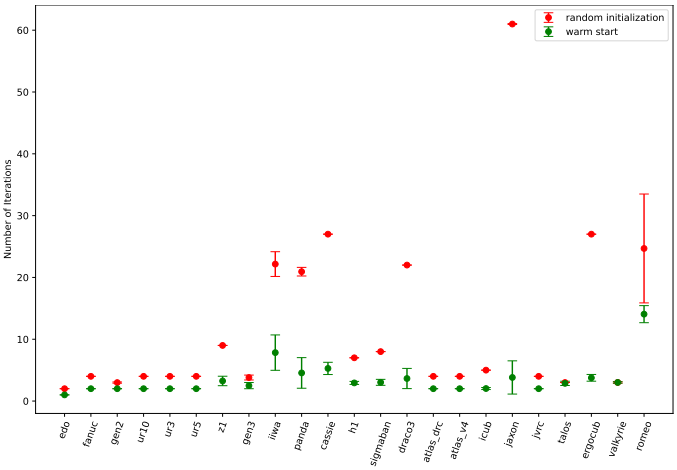

Figure 11: LOIK iteration count with random initialization and warm-starting

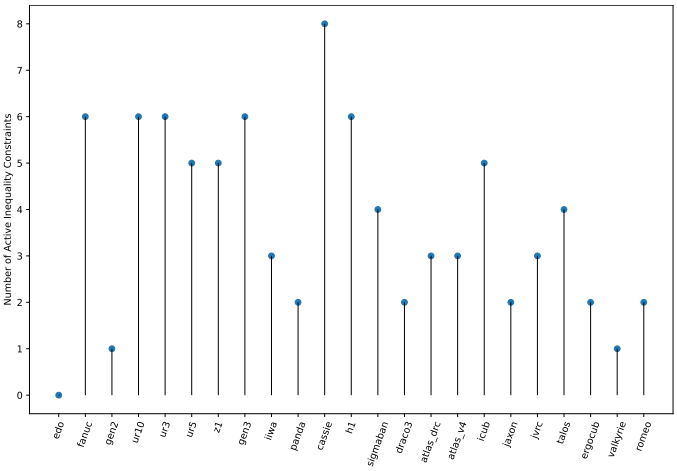

Figure 12: Number of active inequality constraints associated with the problem set solved in Figure 11

It can be seen that LOIK's iteration count is quite stable despite the extremely poor initial guesses. This is perhaps unsurprising since we are solving convex QPs, which, as a problem class, are relatively insensitive to initial guesses. The number of active inequality constraints indicates that these problems were non-trivial. The problem above was even infeasible for some robots, with LOIK returning the closest feasible solution (in weighted l-2 norm) that minimizes the task function error, as explained in Section IV-B.

Additionally, please note that if LOIK was warm-started with the solution from the previous instance, the number of iterations required to converge is significantly reduced, as is expected from augmented Lagrangian methods, which are known to be amenable to warm-starting. We also demonstrate this superb warm-starting property in Figure 11, even for the most difficult problems.