Efficient algorithms for whole-body and second-order robot trajectory optimization

Ajay Suresha Sathya[0000−0002−6746−4630], Lander Vanroye[0000−0003−3968−1061], Wilm Decre[0000−0002−9724−8103] and Jan Swevers[0000−0003−2034−5519]

Abstract This chapter develops an approach for achieving fast whole-body optimal control incorporating exact Lagrangian Hessian information. To this end, we concisely derive the three key state-of-the-art algorithmic ingredients of our approach, namely, 1) a linear solver exploiting the optimal control problem (OCP) structure, 2) a constrained forward dynamics algorithm, and 3) efficient differentiation of the OCP up to second order. The OCP's linear solver and the constrained dynamics algorithm are derived in a unified manner since they are both based on Riccati recursion. For OCP differentiation, we combine the adjoint method and automatic differentiation (AD) to obtain significant efficiency improvements compared to vanilla AD. Our approach's potential is demonstrated on a trajectory optimization problem of a seven DoF Kuka LBR iiwa robot arm with full-order dynamics and a horizon length of 50, where we achieve 0.5 ms per OCP iteration with exact Lagrangian Hessian compared to 0.3 ms for Gauss-Newton approximation. Fewer OCP iterations for the exact Hessian approach result in overall faster computation time, challenging the conventional wisdom that exact Hessian methods are prohibitively expensive for fast OCP solvers.

1 Introduction

Trajectory optimization (TO) is a well-established technique to generate high performance and collision-free robot motions [1], and is formulated naturally as an optimal control problem (OCP) [2, 3]. The system limits and obstacles are formulated as the OCP's constraints, while a desired performance criterion is optimized over the OCP

All authors are with Department of Mechanical Engineering, KU Leuven, and Flanders Make@KU Leuven, Leuven, Belgium.
Ajay Suresha Sathya is also with Inria - Département d'Informatique de l'École normale supérieure, PSL Research University.
e-mail: {ajay.sathya, lander.vanroye, wilm.decré, jan.swevers}@kuleuven.be, ajay.sathya@inria.fr.

horizon. Common performance criteria include desired motion execution time, the energy consumed, actuator effort or a trade-off between these terms [4]. In particular, solving minimum-time problems can generate highly dynamic robot behaviors while still satisfying the system constraints. The most widespread approach to solving OCPs employs the so-called direct methods, which discretize the problem in the time domain, using shooting [5, 2] or collocation methods [6, 7], to obtain a finite-dimensional optimization problem, that can be solved using conventional nonlinear programming (NLP) techniques [8].

The computational cost of robot TO depends to a great extent on the fidelity of the kinematics and dynamics model considered. At the kinematics level, the simplest approximation would be to perform only end-effector TO and later track this motion using an inverse kinematics controller. However, this precludes imposing important joint limit constraints and whole-body obstacle avoidance constraints. Due to kinematics computation being relatively inexpensive, modern approaches commonly use the full kinematics model. Typical approximations at the dynamics level are 1) ignoring dynamics, 2) Center-of-Mass (CoM) approaches treating the robot as a point-mass without angular momentum, and 3) centroidal dynamics accounting for angular momentum, but still considering the robot as a single rigid body. Due to the high computational cost of full-order dynamics, a combination of centroidal dynamics with full kinematics model is a currently popular robot TO approach. However, the centroidal dynamics approach does not support imposing torque limits and assumes that desired contact forces can be achieved. Therefore, this approach can require extensive tuning and good initial guesses to generate feasible motions when performing close to system limits. Whole-body TO overcomes these drawbacks by considering the full-order kinematics and dynamics models, which permits the highest expressivity in formulating costs and constraints among the existing TO approaches.

However, the major challenge with whole-body TO is its computational cost. Therefore, whole-body TO is typically limited to generating reference trajectories in an offline manner and for known environments. Fast TO can enable a robot to generate dynamic motions *online* using the latest sensor measurements, allowing it to respond to environmental changes. Speeding up TO requires accelerating its most computationally intensive subproblems, which are, the OCP solver and evaluation of the robot dynamics and its derivatives. The cost of these operations scale cubically with the number of robot's degrees of freedom (DoF) [9], making it especially expensive for humanoids and quadrupeds. To avoid the additional computational cost incurred by computing second-order derivatives, whole-body TO approaches typically restrict themselves to Gauss-Newton Hessian approximations [10], though incorporating exact Lagrangian Hessian may benefit convergence.

Several advancements have been made towards speeding up OCP solvers, which include both algorithmic improvements [11, 12, 13, 14] and high-performance software implementations [15, 16, 17, 18]. In this chapter, we focus on the algorithmic aspects and derive state-of-the-art algorithms for the OCP's linear solver, computing robot dynamics and computing the OCP's derivatives up to second order and show that these developments lead to an overall speed-up of second-order and whole-body

TO. A concise treatment of this wide gamut of topics in a single chapter is made possible, by both the OCP's linear solver and the robot dynamics algorithm being based on a common framework of Riccati recursion, and the derivative computation leveraging a simple combination of the adjoint method [19] and automatic differentiation (AD).

1.1 Organization and contributions

In the remainder of this introduction section, we present a brief survey of relevant literature. The next section on preliminaries presents the notation, formalizes the OCP, and introduces robot dynamics problems and the adjoint method for differentiation. Section 3 presents a generalized Riccati recursion algorithm [12] for an equality constrained linear quadratic regulator (LQR) problem. This algorithm is the inner linear solver of the nonlinear OCP solver FATROP [16] presented in in Section 4. The same Riccati recursion is used to derive a state-of-the-art constrained dynamics algorithm (CDA) [14] in Section 5 before discussing efficient differentiation in Section 6. Section 7 presents computational benchmarking results before discussion and concluding remarks in Sections 8 and 9 respectively.

This chapter aims to provide a concise derivation of the state-of-the-art algorithms comprising our approach to solving whole-body OCPs with exact Lagrangian Hessian information. Therefore, an exhaustive survey of this active field, where new solvers and software continue to be developed, remains out of its scope. Notwithstanding, this chapter's contents capture key algorithmic aspects of existing efficient OCP solvers and can aid readers in understanding the existing literature. In addition to this tutorial nature, the chapter contains novel aspects enumerated below.

1. Extends the adjoint method to compute the terms due to dynamics in OCP Lagrangian's Hessian. This approach provides a simpler derivation for the state-of-the-art algorithm proposed in [9], where each derivative term was manually derived using tensors. Our approach is also more general and readily supports additional costs and constraints on robot accelerations and constraint forces.
2. An alternate dynamic programming [20] (DP) based derivation is provided for the generalized Riccati recursion in [12], that might be more intuitive than the linear algebra based derivation of [12] some readers.
3. Our algorithms and their efficient implementation enable a computational cost of about 500 μs per OCP solver iteration for solving a point-to-point robot arm motion task with an exact Hessian method, full dynamics model and 50 nodes in the horizon. This is significantly faster than existing approaches [21], to the best of our knowledge.

The C++ FATROP library[1] and a and a MATLAB implementation of the constrained dynamics algorithm[2] are open-sourced and made available. The code for

[1] https://github.com/meco-group/fatrop

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

the adjoint method and the benchmark examples of Sec. 7 is also made publicly availabe[3].

Connection between subsequent sections: A whole-body TO problem is modelled as a nonlinear OCP using the CDA from Sec. 5 for the OCP dynamics. This CDA is a special case of our LQR solver introduced in Sec. 3. The OCP is solved using the nonlinear interior point method from Sec. 4 and requires the constraint Jacobian and Lagrangian derivatives up to second-order, which are evaluated using the adjoint method presented in Sec. 2.4. Each barrier sub-problem of the interior point method is solved using LQR solver from Sec. 3.

1.2 Existing work

Optimal control problem solvers: OCPs can be solved using mature off-the-shelf NLP solvers [8] like IPOPT [22] or SNOPT [23], which solve the NLP using Newton iterations. These iterations solve a large sparse linear system resulting from the OCP's KKT conditions [8] using general purpose sparse linear algebra backends, which being general purpose, do not exploit OCP's Markovian structure. Eventually, structure-exploiting algorithms like iterative LQR (iLQR) [24] or differential dynamic programming (DDP) based on Riccati recursion were reported to provide at least an order-of-magnitude speed-up compared to the general purpose solvers. This speed-up is facilitated by Riccati recursion performing matrix operations on small (< 100 rows and columns) blocks, which fit in a modern CPU's cache. Exploiting this idea, the BLASFEO [25] library was developed to accelerate linear algebra for small matrices on widely-used CPU architectures with a tailored implementation. The field has seen extensive activity in recent years, with the early successors of the vanilla iLQR/DDP methods proposed in OCS [26], ALTRO [27] or CROCCODYL [28] toolboxes. The ACADOS [15] library is an especially efficient software (due to BLASFEO linear algebra backend), that uses the sequential quadratic programming (SQP) [8] method and real-time iteration scheme [29] to target MPC applications. Recent works like OCS2 [30], FATROP [16], Katayama's solver [31], PROXDDP [10], MIM solver [32] have attempted adding to structure exploiting solvers the capability to systematically support stage-wise equality/inequality constraints and advanced globalization techniques to bring reliability of the fast OCP solvers closer to the mature off-the-shelf NLP solvers.

This chapter reviews our state-of-the-art fast OCP solver FATROP [16], whose nonlinear interior-point method and advanced globalization techniques closely mirrors that of IPOPT. At FATROP's core is the generalized Riccati recursion algorithm [12] (derived in Sec.3), which by supporting stage-wise state-input mixed equality/inequality constraints and terminal state constraints and making few regularity assumptions, is more general than most existing structure-exploiting solvers.

Robot dynamics algorithms:

[3] https://github.com/meco-group/adjoint_method

A robot's motion can be unconstrained (not counting the joint constraints) or constrained due to environmental contacts, e.g., when a robot arm is contouring a surface. For unconstrained dynamics problems, the articulated body algorithm (ABA) [33, 34, 35] is asymptotically the most efficient algorithm with a worst-case computational complexity of $O(n)$ for kinematic trees, where n is the robot's degrees of freedom (DOFs). For constrained dynamics problems, the most widely used algorithms in simulators are Featherstone's sparsity-exploiting LTL algorithms [36, 37], which have an expensive worst-case computational complexity of $O(nd^2+d^2m+m^2d+m^3)$, where d is the kinematic tree depth and m is constraint dimensionality. However, there exist older, but sparsely used constrained dynamics algorithms [38, 39, 40], that are asymptotically more efficient than the LTL algorithms, with a worst-case computational complexity of $O(n+m^2d+m^3)$. Our recent paper [14] revisited these old algorithms from an LQR perspective and proposed a state-of-the-art constrained dynamics algorithm (CDA) called the PV-early algorithm, with a computational complexity of $O(n+m)$. We will show that a special case of the Riccati recursion algorithm derived in this chapter yields a state-of-the-art $O(n+m)$ complexity constrained dynamics algorithm.

Constrained dynamics is a complex problem on its own, especially with inequality or frictional contact constraints, which are mathematically modeled as a nonlinear complementarity problem. A detailed overview on this topic is provided in [41]. This chapter restricts itself to the simpler equality-constrained dynamics problems.

Differentiation of constraints and objectives:

Differentiating dynamics functions is easily the most computationally demanding operation in an OCP solver, especially for larger robots, and needs to be computed efficiently. Automatic differentiation (AD) tools like CasADi [17], CppAD [42] and Julia's Autodiff are widely used since they are significantly more efficient and numerically accurate than black-box methods like finite differences. However, these AD tools do not optimally deal with the $\mathbb{SO}(3)$ manifold while differentiating through the rotation matrices. Tackling this issue and using the implicit function method was shown [13] to considerably speed-up differentiating dynamics functions. This result was extended to constrained dynamics settings with an efficient implementation available in the widely-used Pinocchio [18] library.

Most existing fast whole-body OCP solvers [28, 10, 32] restrict themselves to first-order derivatives and Gauss-Newton Hessian approximation because computing second-order dynamics derivatives [43] (resulting in 3D tensors) is prohibitively expensive. However, OCP solvers only need to compute the Lagrangian Hessian (a 2D matrix), which is significantly cheaper to compute. This insight was combined with the implicit function approach [13] to efficiently compute the Lagrangian Hessian terms due to unconstrained forward dynamics in [44] and later extended to constrained dynamics in [9]. Their exact Hessian approach [44, 45] empirically demonstrated faster convergence compared to the approximate Hessian methods. Adopting similar strategy, this chapter leverages the adjoint method [19] to compute first and second-order derivatives. Our resulting derivation is simpler and arguably more elegant than [9], where the authors hand-derived each partial derivative term and is also more general as we will see in the Sections 2.4 and 6.

2 Preliminaries

In this section, we review notation, formalize OCP, dynamics problems and introduce the adjoint method.

2.1 Notation

Lower-case symbols, lower-case bold-faced symbols and upper-case symbols represent scalars, vectors and matrices respectively. $\mathbb{R}^n$ and $\mathbb{R}^{n\times m}$ are the set of n-dimensional real-valued vectors and $n \times m$ real-valued matrices respectively. $\mathbb{S}^n$, $\mathbb{S}^n_+$ and $\mathbb{S}^n_{++}$ represent the set of $n\times n$ symmetric matrices, positive semi-definite matrices and positive definite matrices respectively. A^T represents transpose of the matrix A. I_n refers to an identity matrix of size $n \times n$. A matrix of size $m \times n$ of zeros or ones is represented by $0_{m\times n}$ and $1_{m\times n}$ respectively. $\frac{\partial y}{\partial x}$ is the partial derivative of a function y w.r.t. x. $\frac{dy}{dx}$ represents the total derivative of y w.r.t. x. For a vector-valued function $\mathbf{y}(\mathbf{x})$, the gradient $\nabla_{\mathbf{x}}\mathbf{y} = \frac{d\mathbf{y}}{d\mathbf{x}}^T$. Let $\boldsymbol{q} \in \mathcal{Q}$ denote a robot configuration, where $\mathcal{Q}$ is the configuration space, $\boldsymbol{\nu} \in \mathcal{T}_{\boldsymbol{q}}\mathcal{Q} \simeq \mathbb{R}^n$ denote the generalized robot velocities, $\dot{\boldsymbol{\nu}}$ denote the generalized robot accelerations and $\boldsymbol{\tau} \in \mathcal{T}^*_{\boldsymbol{q}}\mathcal{Q} \simeq \mathbb{R}^n$ denote the generalized robot forces.

2.2 OCP formulation

This chapter considers discrete-time OCP formulations of the form

$$\underset{\mathbf{x},\mathbf{u}}{\textbf{minimize}} \quad \sum_{k=0}^{K-1} \{l_k(\mathbf{x}_k, \mathbf{u}_k)\} + l_K(\mathbf{x}_K), \tag{1a}$$

$$\textbf{subject to} \quad \mathbf{x}_{k+1} = \mathbf{f}_{d,k}(\mathbf{x}_k, \mathbf{u}_k),\ k = 0, 1, \ldots, K-1, \tag{1b}$$

$$\mathbf{g}_{s,k}(\mathbf{x}_k, \mathbf{u}_k) \leq \mathbf{0}, \quad \mathbf{h}_{s,k}(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{0}, \quad k = 0, 1, \ldots, K-1, \tag{1c}$$

$$\mathbf{g}_e(\mathbf{x}_K) \leq \mathbf{0}, \quad \mathbf{h}_e(\mathbf{x}_K) = \mathbf{0}, \tag{1d}$$

where $\mathbf{x}_k \in \mathbb{R}^{n_x}$ and $\mathbf{u}_k \in \mathbb{R}^{n_u}$ are the state and control variables at time-step k respectively, K is the horizon length, $l_k : \mathbb{R}^{n_x\times n_u} \mapsto \mathbb{R}$ and $l_K : \mathbb{R}^{n_x} \mapsto \mathbb{R}$ are the stage-wise and terminal cost functions respectively, $\mathbf{f}_{d,k} : \mathbb{R}^{n_x\times n_u} \mapsto \mathbb{R}^{n_x}$ is the discretized dynamics function at time-step k, $\mathbf{g}_{s,k} : \mathbb{R}^{n_x\times n_u} \mapsto \mathbb{R}^{n_{g,k}}$ and $\mathbf{g}_e : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_{g,K}}$ are the stage and terminal inequality constraint functions respectively. Analogously, $\mathbf{h}_{s,k} : \mathbb{R}^{n_x\times n_u} \mapsto \mathbb{R}^{n_{h,k}}$ and $\mathbf{h}_e : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_{h,K}}$ are the stage and terminal equality constraint functions respectively. All the functions above are assumed to be twice continuously differentiable.

2.3 Robot dynamics algorithms

Robot dynamics problems are broadly classified as inverse dynamics (ID) or forward dynamics (FD) problems, which can further be unconstrained or constrained.
Unconstrained dynamics is formulated by the Euler-Lagrange equations

$$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \mathbf{c}(\boldsymbol{q}, \boldsymbol{\nu}) = \boldsymbol{\tau}, \tag{2}$$

where $M \in \mathbb{S}^n_{++}$ is the joint-space inertia matrix (JSIM) and $\mathbf{c} \in \mathbb{R}^n$ is the generalized force vector due to gravity, centripetal, and centrifugal effects.

Formally, an unconstrained ID problem is the mapping $\text{ID}(\boldsymbol{q}, \boldsymbol{\nu}, \dot{\boldsymbol{\nu}}) \mapsto \boldsymbol{\tau}$, that can be solved by simply evaluating the left-hand-side of Eq. (2). M and $\mathbf{c}$ can be efficiently computed using the composite rigid body algorithm (CRBA) [46] algorithm. However, the recursive Newton-Euler algorithm (RNEA) [35] is the most efficient unconstrained ID algorithm with an $O(n)$ computational complexity. RNEA avoids explicitly computing M, which requires at least $O(n^2)$ operations.

Formally, unconstrained FD problem is the mapping $\text{FD}(\boldsymbol{q}, \boldsymbol{\nu}, \boldsymbol{\tau}) \mapsto \dot{\boldsymbol{\nu}}$, that solves the linear system in Eq. (2) for $\dot{\boldsymbol{\nu}}$. Naively factorizing M requires $O(n^3)$ operations. However, for tree-structured robots, the LTL algorithm [36] exploits the branching-induced sparsity in M to compute its Cholesky factorization with a reduced $O(nd^2)$ complexity, where d is the tree depth. The fastest unconstrained FD algorithm is the recursive articulated body algorithm (ABA) [33, 34, 47], with a $O(n)$ complexity, which, similarly to RNEA, avoids explicitly computing the JSIM.
Constrained dynamics problems, that are required to satisfy the equality constraint

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

are formulated at the acceleration level with the following two equations

$$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \mathbf{c}(\boldsymbol{q}, \boldsymbol{\nu}) + J^T\boldsymbol{\lambda} = \boldsymbol{\tau}, \tag{3a}$$

$$J(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{J}(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu} = \mathbf{a}^*_c, \tag{3b}$$

where $J \in \mathbb{R}^{m\times n}$ is the constraint Jacobian, $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the Lagrange multiplier vector, $\mathbf{a}^*_c \in \mathbb{R}^m$ is the affine constraint acceleration terms, and $\dot{J}$ is J's total time derivative. These terms are obtained from holonomic and non-holonomic constraints by differentiating them w.r.t. time once or twice respectively to get Eq. (3b).

Constrained ID problem is the mapping, $\text{ID}_c(\boldsymbol{q}, \boldsymbol{\nu}, \dot{\boldsymbol{\nu}}, \boldsymbol{\lambda}) \mapsto \boldsymbol{\tau}$, which can be solved by simply evaluating the left-hand-side of Eq. (3a). The unconstrained ID algorithm, RNEA can be easily modified to include constraint forces $J^T\boldsymbol{\lambda}$, and remains the fastest constrained ID algorithm with an $O(n + m)$ complexity. The constrained RNEA version computes neither M nor J explicitly. Assuming that J is full row-rank, constrained FD problem is the mapping, $\text{FD}_c(\boldsymbol{q}, \boldsymbol{\nu}, \boldsymbol{\tau}) \mapsto (\boldsymbol{\lambda}, \dot{\boldsymbol{\nu}})$, which requires simultaneously solving the two linear equations in Eq. (3). Computing constrained FD is significantly costlier than unconstrained FD (often by a factor ~2), which is in turn significantly costlier than computing the ID problems (again by a factor of ~2).

Note 1 This chapter assumes that J has full row-rank. This assumption is violated if the motion constraints are redundantly or incorrectly specified. Then, there will be either no solution for $(\boldsymbol{\lambda}, \dot{\boldsymbol{\nu}})$ or an infinite number of solutions for $\boldsymbol{\lambda}$. In practice, this issue is handled via Tikhonov regularization [48], truncated SVD or proximal iterations [49]. We avoid these details for simplicity.

For kinematic trees, it turns out that J has a branching-induced sparsity pattern similar to M. This was exploited in an extension of the LTL algorithm [37] to obtain a CDA with an expensive $O(nd^2+m^2d+md^2+m^3)$ complexity, and is widely used in existing simulators like Pinocchio [18], Raisim [50], MuJoCo [48], etc. Our recent work proposes a faster recursive algorithm [14] similar to ABA with a low $O(n+m)$ complexity. In Section 5, we will show that a special case of the Riccati recursion from Section 3 yields an $O(n+m)$ complexity constrained dynamics algorithm.

2.4 Adjoint-method for differentiation

We now review the adjoint method, which will later speed up differentiating dynamics functions. Refer [19] for a tutorial introduction to the adjoint method.
First derivatives: Consider a twice-differentiable function $f(\mathbf{x}, \mathbf{y}(\mathbf{x})) : \mathbb{R}^{n_x \times n_y} \mapsto \mathbb{R}$, where $\mathbf{x} \in \mathbb{R}^{n_x}$ is an independent variable, and $\mathbf{y}(\mathbf{x}) \in \mathbb{R}^{n_y}$ is a dependent variable. Our problem is to compute $\frac{df}{d\mathbf{x}}$, which from the chain rule is

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} + \frac{\partial f}{\partial \mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}}. \tag{4}$$

However, computing $\frac{d\mathbf{y}}{d\mathbf{x}}$ can be expensive and/or the function $\mathbf{y}(\mathbf{x})$ may not even be explicitly available for differentiation. Suppose that $\mathbf{y}(\mathbf{x})$ is implicitly defined by a twice-differentiable function $\mathbf{g}(\mathbf{x}, \mathbf{y}(\mathbf{x})) : \mathbb{R}^{n_x \times n_y} \mapsto \mathbb{R}^{n_y}$, for which both $\frac{\partial \mathbf{g}}{\partial \mathbf{y}}$ is invertible and $\mathbf{g}(\mathbf{x}, \mathbf{y}(\mathbf{x})) = \mathbf{0}_{n_y}$ *everywhere*, i.e. $\forall \mathbf{x} \in \mathbb{R}^{n_x}$. It follows that

$$\frac{d\mathbf{g}}{d\mathbf{x}} = \mathbf{0}_{n_y}, \tag{5}$$

expanding which using chain rule gives

$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}} + \frac{\partial \mathbf{g}}{\partial \mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{x}} = \mathbf{0}_{n_y}, \tag{6}$$

where using $\frac{\partial \mathbf{g}}{\partial \mathbf{y}}$'s invertibility, we have

$$\frac{d\mathbf{y}}{d\mathbf{x}} = -\left(\frac{\partial \mathbf{g}}{\partial \mathbf{y}}\right)^{-1} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}. \tag{7}$$

The result above is the well-known implicit function theorem [51]. Substituting this in Eq. (4) gives

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} - \frac{\partial f}{\partial \mathbf{y}} \left(\frac{\partial \mathbf{g}}{\partial \mathbf{y}}^{-1} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \right). \tag{8}$$

Evaluating $\frac{d\mathbf{y}}{d\mathbf{x}}$ above (the expression in the parenthesis) requires an expensive matrix-matrix operation. This can be avoided by re-ordering the parentheses

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} - \left(\frac{\partial f}{\partial \mathbf{y}} \frac{\partial \mathbf{g}}{\partial \mathbf{y}}^{-1} \right) \frac{\partial \mathbf{g}}{\partial \mathbf{x}}, \tag{9}$$

upon observing that $\frac{\partial f}{\partial \mathbf{y}}$ is a $1 \times n_y$ row vector using the associativity property of the matrix product operation. Let $\boldsymbol{\mu}^{\boldsymbol{T}} \in \mathbb{R}^{n_y}$ denote the expression in the parentheses above, which is evaluated by solving the linear system

$$\frac{\partial \mathbf{g}}{\partial \mathbf{y}}^{T} \boldsymbol{\mu} = \frac{\partial f}{\partial \mathbf{y}}^{T}. \tag{10}$$

The adjoint method gets its name from having to solve the transpose of $\frac{\partial \mathbf{g}}{\partial \mathbf{y}}$ above. Substituting the value of $\boldsymbol{\mu}$ in Eq. (9) gives an efficient expression to compute $\frac{df}{d\mathbf{x}}$,

$$\frac{df}{d\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} - \boldsymbol{\mu}^{T} \frac{\partial \mathbf{g}}{\partial \mathbf{x}}. \tag{11}$$

Second derivatives: To compute f's Hessian, let us concatenate Eq. (11) and Eq. (10)

$$\begin{pmatrix} \frac{df}{d\mathbf{x}}^{T} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial \mathbf{z}}^{T} - \frac{\partial \mathbf{g}}{\partial \mathbf{z}}^{T} \boldsymbol{\mu} \end{pmatrix}, \tag{12}$$

where $\mathbf{z} = \left[\mathbf{x}^T, \mathbf{y}^T\right]^T$. Differentiating the equation above again w.r.t. $\mathbf{x}$ gives

$$\begin{pmatrix} \frac{d^2 f}{d\mathbf{x}^2} \\ 0 \end{pmatrix} = \left(\frac{\partial^2 f}{\partial \mathbf{z}^2} - \frac{\partial^2 \left(\mathbf{g}^T \tilde{\boldsymbol{\mu}} \right)}{\partial \mathbf{z}^2} \right) \begin{pmatrix} I \\ \frac{d\mathbf{y}}{d\mathbf{x}} \end{pmatrix} - \left(\frac{\partial \mathbf{g}}{\partial \mathbf{z}} \right)^T \frac{d\boldsymbol{\mu}}{d\mathbf{z}} \begin{pmatrix} I \\ \frac{d\mathbf{y}}{d\mathbf{x}} \end{pmatrix}, \tag{13}$$

where, $\tilde{\boldsymbol{\mu}}$ is fixed to the numerical value of $\boldsymbol{\mu}$ during differentiation, and computing $\frac{d\boldsymbol{\mu}}{d\mathbf{z}}$ requires expensively differentiating through the solution of Eq. (10). However, this can be avoided. Restating Eq. (6) gives

$$\frac{\partial \mathbf{g}}{\partial \mathbf{z}} \begin{pmatrix} I_{n_x} \\ \frac{d\mathbf{y}}{d\mathbf{x}} \end{pmatrix} = 0,$$

using which, pre-multiplying the Hessian equation in Eq. (13) by $\left(I_{n_x} \ \frac{d\mathbf{y}}{d\mathbf{x}}^T \right)$ simplifies the Hessian expression to

$$\frac{d^2 f}{d\mathbf{x}^2} = \begin{pmatrix} I_{n_x} & \frac{d\mathbf{y}}{d\mathbf{x}}^T \end{pmatrix} \left(\frac{\partial^2 f}{\partial \mathbf{z}^2} - \frac{\partial^2 \left(\mathbf{g}^T \tilde{\boldsymbol{\mu}} \right)}{\partial \mathbf{z}^2} \right) \begin{pmatrix} I_{n_x} \\ \frac{d\mathbf{y}}{d\mathbf{x}} \end{pmatrix}. \tag{14}$$

3 Efficient LQR solver

We now derive the generalized Riccati recursion algorithm [12] for a stage-wise equality constrained LQR problem, that will form the basis of both the fast OCP solver and constrained dynamics algorithms discussed later. Our dynamic programming based derivation of [12] is different from the linear algebra based derivation in [12] and may be more intuitive to some readers. Consider an LQR problem of the following form

$$\underset{\mathbf{x},\mathbf{u}}{\textbf{minimize}} \quad \sum_{k=0}^{K-1} \left\{ \frac{1}{2} \begin{pmatrix} \mathbf{u}_k \\ \mathbf{x}_k \end{pmatrix}^T \begin{pmatrix} R_k & S_k^T \\ S_k & Q_k \end{pmatrix} \begin{pmatrix} \mathbf{u_k} \\ \mathbf{x}_k \end{pmatrix} + \begin{pmatrix} \mathbf{r}_k \\ \mathbf{q}_k \end{pmatrix}^T \begin{pmatrix} \mathbf{u}_k \\ \mathbf{x}_k \end{pmatrix} \right\} + \mathbf{x}_K^T Q_K \mathbf{x}_K + \mathbf{q}_K^T \mathbf{x}_K, \tag{15a}$$

$$\textbf{subject to} \quad \mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k + \mathbf{b}_k, \ k = 0, 1, \ldots, K-1, \tag{15b}$$

$$G_{k,x} \mathbf{x}_k + G_{k,u} \mathbf{u}_k = -\mathbf{g}_k, \quad k = 0, 1, \ldots, K-1, \tag{15c}$$

$$G_{K,x} \mathbf{x}_K = -\mathbf{g}_K. \tag{15d}$$

Due to the additional stage-wise and terminal equality constraints in Eq. (15c) and Eq. (15d), vanilla Riccati recursion [2] using the so-called cost-to-go function does not suffice. We will instead use a 'constrained cost-to-go' problem. Suppose that the optimal 'constrained cost-to-go' problem at stage k can be parameterized as

$$V_k^*(\mathbf{x}_k) = \frac{1}{2} \mathbf{x}_k^T P_k \mathbf{x}_k + \mathbf{p}_k^T \mathbf{x}_k, \tag{16a}$$

$$\textbf{subject to} \quad \hat{G}_{k,x} \mathbf{x}_k = -\hat{\mathbf{g}}_k. \tag{16b}$$

Bellman recurrence relation: Using Bellman's recurrence relation, the optimal 'constrained cost-to-go' problem at stage k is written as the solution to

$$\underset{\mathbf{x}_{k+1},\mathbf{u}_k}{\textbf{minimize}} \quad \left\{ \frac{1}{2} \begin{pmatrix} \mathbf{u}_k \\ \mathbf{x}_k \end{pmatrix}^T \begin{pmatrix} R_k & S_k^T \\ S_k & Q_k \end{pmatrix} \begin{pmatrix} \mathbf{u_k} \\ \mathbf{x}_k \end{pmatrix} + \begin{pmatrix} \mathbf{r}_k \\ \mathbf{q}_k \end{pmatrix}^T \begin{pmatrix} \mathbf{u}_k \\ \mathbf{x}_k \end{pmatrix} \right\} + \frac{1}{2} \mathbf{x}_{k+1}^T P_{k+1} \mathbf{x}_{k+1} + \mathbf{p}_{k+1}^T \mathbf{x}_{k+1} \tag{17a}$$

$$\textbf{subject to} \quad \mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k + \mathbf{b}_k, \tag{17b}$$

$$G_{k,x} \mathbf{x}_k + G_{k,u} \mathbf{u}_k = -\mathbf{g}_k, \tag{17c}$$

$$\hat{G}_{k+1,x} \mathbf{x}_{k+1} = -\hat{\mathbf{g}}_{k+1}. \tag{17d}$$

Eliminating system dynamics: We first eliminate the dynamics constraints in Eq. (17b) through substitution to get

$$\underset{\mathbf{u}_k}{\textbf{minimize}} \quad \frac{1}{2}\begin{pmatrix}\mathbf{u}_k\\ \mathbf{x}_k\end{pmatrix}^T\begin{pmatrix}\bar{R}_k & \bar{S}_k^T\\ \bar{S}_k & \bar{Q}_k\end{pmatrix}\begin{pmatrix}\mathbf{u_k}\\ \mathbf{x}_k\end{pmatrix}+\begin{pmatrix}\bar{\mathbf{r}}_k\\ \bar{\mathbf{q}}_k\end{pmatrix}^T\begin{pmatrix}\mathbf{u}_k\\ \mathbf{x}_k\end{pmatrix} \tag{18a}$$

$$\textbf{subject to} \quad \bar{G}_{k,x}\mathbf{x}_k+\bar{G}_{k,u}\mathbf{u}_k=-\bar{\mathbf{g}}_k, \tag{18b}$$

where

$$\bar{R}_k=R_k+B_k^TP_{k+1}B_k,\quad \bar{S}_k=S_k+A_k^TP_{k+1}B_k,\quad \bar{Q}_k=Q_k+A_k^TP_{k+1}A_k,$$

$$\bar{\mathbf{r}}_k=\mathbf{r}_k+B_k^T\left(\mathbf{p}_{k+1}+P_{k+1}\mathbf{b}_k\right),\quad \bar{\mathbf{q}}_k=\mathbf{q}_k+A_k^T\left(\mathbf{p}_{k+1}+P_{k+1}\mathbf{b}_k\right),$$

$$\bar{G}_{k,x}=\begin{pmatrix}G_{k,x}\\ \hat{G}_{k+1,x}A_k\end{pmatrix},\quad \bar{G}_{k,u}=\begin{pmatrix}G_{k,u}\\ \hat{G}_{k+1,x}B_k\end{pmatrix},\quad \bar{\mathbf{g}}_k=\begin{pmatrix}\mathbf{g}_k\\ \hat{\mathbf{g}}_{k+1}+\hat{G}_{k+1,x}\mathbf{b}_k\end{pmatrix}.$$

Decomposing the problem into the range and nullspace of $\bar{G}_{k,u}$: To optimize Eq. (18a) over $\mathbf{u}_k$ subject to the constraint in Eq. (18b), we compute the nullspace of $\bar{G}_{k,u}$, using the following decomposition.

$$\bar{G}_{k,u}=T_{k,L}\begin{pmatrix}I_{\rho_k} & \\ & 0_{m_k-\rho_k\times n_u-\rho_k}\end{pmatrix}T_{k,R}, \tag{19}$$

where $T_{k,L}\in\mathbb{R}^{m_k\times m_k}$ and $T_{k,R}\in\mathbb{R}^{n_u\times n_u}$ are invertible matrices and ρ_k is the rank of $\bar{G}_{k,u}$. We compute this decomposition using completely pivoted LU factorization [12, Section 2.3] for computational efficiency. Other slower, but potentially more numerical stable, algorithms like singular value decomposition (SVD) or QR decomposition could also be used. Let

$$\mathbf{u}_k=T_{k,R}^{-1}\begin{pmatrix}\tilde{\mathbf{u}}_k\\ \hat{\mathbf{u}}_k\end{pmatrix}, \tag{20}$$

where $\tilde{\mathbf{u}}_k\in\mathbb{R}^{\rho_k}$ and $\hat{\mathbf{u}}_k\in\mathbb{R}^{n_u-\rho_k}$. Substituting Eq. (20) into Eq. (18a) gives

$$\frac{1}{2}\begin{pmatrix}\tilde{\mathbf{u}}_k\\ \hat{\mathbf{u}}_k\\ \mathbf{x}_k\end{pmatrix}^T\begin{pmatrix}\tilde{R}_k & \hat{\tilde{R}}_k^T & \tilde{S}_k^T\\ \hat{\tilde{R}}_k & \hat{R}_k & \hat{S}_k'^T\\ \tilde{S}_k & \hat{S}_k' & \bar{Q}_k\end{pmatrix}\begin{pmatrix}\tilde{\mathbf{u}}_k\\ \hat{\mathbf{u}}_k\\ \mathbf{x}_k\end{pmatrix}+\begin{pmatrix}\tilde{\mathbf{r}}_k\\ \hat{\mathbf{r}}_k'\\ \bar{\mathbf{q}}_k\end{pmatrix}^T\begin{pmatrix}\tilde{\mathbf{u}}_k\\ \hat{\mathbf{u}}_k\\ \mathbf{x}_k\end{pmatrix}, \tag{21}$$

where

$$\begin{pmatrix}\tilde{R}_k & \hat{\tilde{R}}_k^T\\ \hat{\tilde{R}}_k & \hat{R}_k\end{pmatrix}=T_{k,R}^{-T}\bar{R}_kT_{k,R}^{-1},\quad \begin{pmatrix}\tilde{S}_k & \hat{S}_k'\end{pmatrix}=\bar{S}_kT_{k,R}^{-1},\quad \begin{pmatrix}\tilde{\mathbf{r}}_k\\ \hat{\mathbf{r}}_k'\end{pmatrix}=T_{k,R}^{-T}\bar{\mathbf{r}}_k.$$

Substituting Eq. (20) into Eq. (18b) and left-multiplying it with $T_{k,L}^{-1}$ gives

$$\begin{pmatrix}I_{\rho_k} & \\ & 0_{(n_u-\rho_k)\times(n_u-\rho_k)}\end{pmatrix}\begin{pmatrix}\tilde{\mathbf{u}}_k\\ \hat{\mathbf{u}}_k\end{pmatrix}=-\begin{pmatrix}\tilde{G}_{k,x}\\ \hat{G}_{k,x}\end{pmatrix}\mathbf{x}_k-\begin{pmatrix}\tilde{\mathbf{g}}_k\\ \hat{\mathbf{g}}_k\end{pmatrix}, \tag{22}$$

where

$$\begin{pmatrix}\tilde{G}_{k,x}\\ \hat{G}_{k,x}\end{pmatrix} = T_{k,L}^{-1}\bar{G}_{k,x}, \quad \begin{pmatrix}\tilde{\mathbf{g}}_k\\ \hat{\mathbf{g}}_k\end{pmatrix} = T_{k,L}^{-1}\bar{\mathbf{g}}_k.$$

Eliminating $\tilde{\mathbf{u}}_k$: From Eq. (22), we readily have that

$$\tilde{\mathbf{u}}_k = -\tilde{G}_{k,x}\mathbf{x}_k - \tilde{\mathbf{g}}_k, \tag{23}$$

which is substituted in Eq. (21) to get the following objective function

$$\frac{1}{2}\begin{pmatrix}\hat{\mathbf{u}}_k\\ \mathbf{x}_k\end{pmatrix}^T\begin{pmatrix}\hat{R}_k & \hat{S}_k^T\\ \hat{S}_k & \hat{Q}_k\end{pmatrix}\begin{pmatrix}\hat{\mathbf{u}}_k\\ \mathbf{x}_k\end{pmatrix} + \begin{pmatrix}\hat{\mathbf{r}}_k\\ \hat{\mathbf{q}}_k\end{pmatrix}^T\begin{pmatrix}\hat{\mathbf{u}}_k\\ \mathbf{x}_k\end{pmatrix}, \tag{24}$$

where

$$\hat{Q}_k = \bar{Q}_k + \tilde{G}_{k,x}^T\tilde{R}_k\tilde{G}_{k,x} - (\tilde{S}_k\tilde{G}_k + \tilde{G}_k^T\tilde{S}_k^T), \quad \hat{S}_k = \hat{S}_k^{'} + \tilde{G}_{k,x}^T\hat{\tilde{R}}_k^T$$

$$\hat{r}_k = \hat{r}_k^{'} - \hat{\tilde{R}}_k\tilde{g}_k, \quad \hat{q}_k = \hat{q}_k^{'} + \tilde{G}_{k,x}^T\tilde{R}_k\tilde{g}_k - \tilde{S}_k\tilde{g}_k - \tilde{G}_{k,x}^T\tilde{r}_k.$$

Solving for $\hat{\mathbf{u}}_k$: Assuming that the reduced Hessian of the constrained LQR problem [12, Definition 1] is positive definite, $\hat{R}_k$ is invertible [12], which is anyway a necessary condition for the LQR problem to have a unique minimizer [8, Thm. 16.2]. Finally $\hat{\mathbf{u}}_k$, which is unconstrained, can be minimized and eliminated using the Schur complement from the objective function in Eq. (24) to get

$$\frac{1}{2}\mathbf{x}_k^T P_k\mathbf{x}_k + \mathbf{p}_k^T\mathbf{x}_k, \tag{25}$$

where

$$P_k = \hat{Q}_k - \hat{S}_k\hat{R}_k^{-1}\hat{S}_k^T, \quad \mathbf{p}_k = \hat{q}_k - \hat{S}_k^T\hat{R}_k^{-1}\hat{r}_k,$$

with the optimal $\hat{\mathbf{u}}_k$ given by

$$\hat{\mathbf{u}}_k = -\hat{R}_k^{-1}\left(\hat{S}_k^T\mathbf{x}_k + \hat{r}_k\right). \tag{26}$$

The objective function in Eq. (25) combined with the uneliminated constraint on $\mathbf{x}_k$ from Eq. (22)

$$\hat{G}_{k,x}\mathbf{x}_k = -\hat{\mathbf{g}}_k$$

gives the optimal constrained cost-to-go function at stage k with the parametric form assumed in Eq. (16). We now have a recursive set of equations to compute $V_k^*(\mathbf{x}_k)$ from $V_{k+1}^*(\mathbf{x}_{k+1})$. The constrained cost-to-go function at the final stage K is already in the parametric form of Eq. (16) with $P_K = Q_K$ and $\mathbf{p}_K = \mathbf{q}_K$. The computations derived above can be performed till the initial state recursively to obtain

$$\begin{aligned} V_0^*(\mathbf{x}_0) =& \frac{1}{2}\mathbf{x}_0^T P_0\mathbf{x}_0 + \mathbf{p}_0^T\mathbf{x}_0, \\ &\textbf{subject to} \quad \hat{G}_{0,x}\mathbf{x}_0 = -\hat{\mathbf{g}}_0, \end{aligned} \tag{27}$$

whereupon the initial state $\mathbf{x}_0$ is computed by solving a small equality constrained QP. Often, the initial state is fully-constrained from estimator feedback, in which case, solving this QP is skipped.

Forward rollout: Now we have all the ingredients to compute the optimal $\hat{\mathbf{u}}_k$, $\tilde{\mathbf{u}}_k$, $\mathbf{u}_k$ and $\mathbf{x}_k$ using the equations 26, 23, 20, and 15b respectively in a forward rollout from $k = 0$ to $k = K - 1$.

Recovering optimal Lagrange multipliers: Suppose that $\boldsymbol{\eta}_{d,k}$, $\boldsymbol{\eta}_{c,k}$ and $\boldsymbol{\eta}_{T,k+1}$ refer respectively to the Lagrange multipliers of dynamics, stage-wise and terminal constraint (see Eqs. (17b), (17c) and (17d) respectively) of the Bellman recurrence subproblem. From the dual feasibility KKT conditions by taking gradient w.r.t $\mathbf{x}_{k+1}$ in Eq. (17), it can be shown that the optimal $\boldsymbol{\eta}_{d,k}$ are given by

$$\boldsymbol{\eta}_{d,k} = P_{k+1}\mathbf{x}_{k+1} + \hat{G}^T_{k+1,x}\boldsymbol{\eta}_{T,k+1} + \mathbf{p}_{k+1}. \tag{28}$$

Due to the constraint frame transformation with nullspace decomposition, we get

$$\begin{bmatrix} \boldsymbol{\eta}_{c,k} \\ \boldsymbol{\eta}_{T,k+1} \end{bmatrix} = T^{-T}_{k,L} \begin{bmatrix} \tilde{\boldsymbol{\eta}}_{c,k} \\ \boldsymbol{\eta}_{T,k} \end{bmatrix}. \tag{29}$$

The dual feasibility condition of the problem in Eq. (21) by differentiating w.r.t $\tilde{\mathbf{u}}_\mathbf{k}$ gives

$$\tilde{\boldsymbol{\eta}}_{c,k} = \tilde{R}_k\tilde{\mathbf{u}}_k + \tilde{S}_k\mathbf{x}_k + \hat{\tilde{R}}_k\hat{\mathbf{u}}_\mathbf{k} + \tilde{\mathbf{r}}_k. \tag{30}$$

$\boldsymbol{\eta}_{T,0}$ is computed at the end of backward Riccati recursion while solving the optimization problem in Eq.(27), following which Eqs.(30), (29), (28) are evaluated in that order at every k during the roll-out to compute the optimal Lagrange multipliers.

Relation to existing LQR solvers: Our LQR solver makes fewer regularity assumptions than related equality-constrained LQR solvers [52, 53, 54] in literature by only requiring the reduced Hessian to be positive definite, which is anyway a necessary condition for the LQR problem to be well-posed. [53] and [54] assume that the full space Hessian is positive definite, which implies that all the constraints are linearly independent. [52] assumes that R_k is positive definite and requires $G_{k,u}$ to be full row-rank, which implicitly restricts the stage-wise constraints to be fewer than the control input's dimension. This prevents, for example, imposing terminal state constraints on an underactuated robot platform. Other closely related algorithm [55] uses expensive SVD operations to compute Eq. (19) in contrast to the LU decomposition used by us and [55], implemented in MATLAB, is not accompanied by code release. Recent work [10] using proximal algorithms [56] requires the reduced Hessian to be only positive semidefinite, but it performs expensive Schur complement on larger matrices than us and is an iterative algorithm unlike our LQR solver. Furthermore, our C++ implementation uses the tailored BLASFEO linear algebra library, further contributing to its speed. Readers are referred to [12] for detailed benchmarking of our LQR solver.

4 Fatrop

We now briefly review our state-of-the-art nonlinear OCP solver Fatrop. As mentioned in Section. 1.2, Fatrop aims to combine the speed of structure-exploiting solvers with the relative robustness of mature off-the-shelf NLP solvers. To achieve this, we have chosen to emulate the nonlinear interior point strategy and the advanced globalization from the widely-used IPOPT [22] solver in Fatrop, thereby inheriting IPOPT's global and quadratic local convergence properties. Our choice is informed by our prior positive experience with IPOPT on robotics problems.

Applying IPOPT's interior point method to solve the OCP from Eq. 1 results in a linear system that is then solved using the Riccati recursion derived in the previous section. We now show how to derive this linear system from the OCP. Due to several derivations in this section, it was difficult to avoid notational overlap between this section and the rest of the section. We apologize for this overlap and ask readers to consider this section's notation to be self-contained in a 'local scope'.

Let the OCP from Eq. 1 be reformulated as the following NLP with slack variables, without any loss of generality

$$\underset{\mathbf{x},\mathbf{s}}{\textbf{minimize}} \quad f(\mathbf{x}) \tag{31a}$$

$$\textbf{subject to} \quad \mathbf{h}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{g}(\mathbf{x}) - \mathbf{s} = \mathbf{0}, \quad \mathbf{s} \geq \mathbf{0}, \tag{31b}$$

where $\mathbf{x}$ includes all the state and control variables from Eq. 1 and $\mathbf{s}$ denotes the slack variables. The positivity constraint on the slack variables is enforced by introducing a logarithmic barrier term in the objective function:

$$\underset{\mathbf{x},\mathbf{s}}{\textbf{minimize}} \quad f(\mathbf{x}) - \mu_j \sum_i \log(s_i) \tag{32a}$$

$$\textbf{subject to} \quad \mathbf{h}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{g}(\mathbf{x}) - \mathbf{s} = \mathbf{0}. \tag{32b}$$

After introducing additional auxiliary variables $z_i = \mu_j / s_i$, the problem above is solved via Newton iterations over the set of nonlinear equations

$$\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{0}, \quad \nabla_{\mathbf{s}} \mathcal{L} = \mathbf{0}, \tag{33a}$$

$$\mathbf{h}(\mathbf{x}) = \mathbf{0}, \quad \mathbf{g}(\mathbf{x}) - \mathbf{s} = \mathbf{0}, \tag{33b}$$

$$\operatorname{diag}(\mathbf{z})\mathbf{s} = \mu_j \mathbf{1}, \tag{33c}$$

which together with $\mathbf{x}, \mathbf{z} \geq \mathbf{0}$ constitute the first-order necessary optimality conditions or the KKT conditions [8] associated with Eq. (32). The Lagrangian function $\mathcal{L}$ is defined as

$$\mathcal{L} := f(\mathbf{x}) + \boldsymbol{\eta}_{\mathbf{h}}^T \mathbf{h}(\mathbf{x}) + \boldsymbol{\eta}_{\mathbf{g}}^T (\mathbf{g}(\mathbf{x}) - \mathbf{s}) - \mathbf{z}^T \mathbf{s}.$$

The centering equation in Eq. (33c) is a relaxed version of the complementarity conditions of the original problem in Eq. (31). Note that these KKT conditions tend to KKT conditions of the original problem as the barrier parameter μ_j decreases to zero. Therefore, a homotopy is used that starts with a non-zero μ_j, solves the

nonlinear systems from Eq. 33c using Newton's method and decreases the μ_j value until convergence. This is the essential idea behind the primal-dual interior-point method. Each Newton iteration solves the linear system

$$\begin{pmatrix} \nabla^2_{\mathbf{xx}}\mathcal{L} & & \mathbf{J_h}^T & \mathbf{J_g}^T \\ & & & -\boldsymbol{I} \quad -\boldsymbol{I} \\ \mathbf{J_h} & & & \\ \mathbf{J_g} & -\boldsymbol{I} & & \\ & \mathbf{Z} & & \mathbf{S} \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{s} \\ \Delta\boldsymbol{\eta}_\mathbf{h} \\ \Delta\boldsymbol{\eta}_\mathbf{g} \\ \Delta\mathbf{z} \end{pmatrix} = - \begin{pmatrix} \nabla_\mathbf{x}\mathcal{L} \\ \nabla_\mathbf{s}\mathcal{L} \\ \mathbf{h} \\ \mathbf{g} - \mathbf{s} \\ \mathbf{Sz} - \mu_j\mathbf{e} \end{pmatrix}, \tag{34}$$

where a capital symbol such as $\mathbf{S}$ is a diagonal matrix with the vector $\mathbf{s}$ constituting its diagonal. $\Delta\mathbf{s}$, $\Delta\boldsymbol{\eta_g}$ and $\Delta\mathbf{z}$ can be eliminated to obtain a symmetric (possibly indefinite) linear system

$$\begin{pmatrix} \nabla^2_{\mathbf{xx}}\mathcal{L} + \mathbf{J_g}^T\mathbf{S}^{-1}\mathbf{Z}\mathbf{J_g} & \mathbf{J_h}^T \\ \mathbf{J_h} & \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\eta_h} \end{pmatrix} = - \begin{pmatrix} \boldsymbol{\gamma} \\ \mathbf{h} \end{pmatrix}, \tag{35}$$

which we call the reduced primal-dual system, where

$$\boldsymbol{\gamma} = \nabla_\mathbf{x}\mathcal{L} + \mathbf{J_g}^T(-\eta_\mathbf{g} - \mathbf{S}^{-1}\left(\mu_j\mathbf{1} - \mathbf{Z}(\mathbf{g} - \mathbf{s})\right)).$$

From the reduced system's solution, $\Delta\mathbf{s}$, $\Delta\boldsymbol{\eta_g}$ and $\Delta\mathbf{z}$ can be recovered using

$$\Delta\mathbf{s} = \mathbf{J_g}\Delta\mathbf{x} + \mathbf{g} - \mathbf{s}, \tag{36a}$$

$$\Delta\boldsymbol{\eta_g} = -\boldsymbol{\eta_g} - \mathbf{S}^{-1}(\mu_j\mathbf{e} - \mathbf{Z}\Delta\mathbf{s}), \tag{36b}$$

$$\Delta\mathbf{z} = -\mathbf{z} + \mathbf{S}^{-1}(\mu_j\mathbf{e} - \mathbf{Z}\Delta\mathbf{s}). \tag{36c}$$

Due to the stage-wise constraint structure assumed in Eq. (1), differentiating these stage-wise functions gives the corresponding reduced primal-dual system in Eq. (35). This reduced system can be shown to have a sparsity pattern that matches that of the equality-constrained LQR problem in Eq. (15). The terms $\nabla^2_{\mathbf{xx}}\mathcal{L} + \mathbf{J_g}^T\mathbf{S}^{-1}\mathbf{Z}\mathbf{J_g}$ and $\boldsymbol{\gamma}$ correspond to the LQR's quadratic and linear cost terms respectively, while $\mathbf{J_h}$ and $\mathbf{h}$ corresponds to LQR's equality constraints. Apart from the cost of computing the terms in Eq. (34), solving the reduced system in Eq. (35) is typically the most computationally expensive step for an OCP solver and this step is greatly accelerated in Fatrop using Riccati recursion derived in the previous section. Similarly to Acados, Fatrop also uses the Blasfeo [25] library, which is highly optimized for its linear algebra operations. Note that for OCPs that lead to an indefinite reduced Hessian, during the Riccati recursion $\hat{R}_k$'s factorization fails at the Schur complement step in Eq. (25). To address this, we employ Hessian regularization similarly to IPOPT [22].

For a detailed introduction and analysis of primal-dual interior-point methods we refer readers to [8]. Fatrop currently implements nearly all the advanced globalized features of IPOPT [22], except for the restoration phase. For more details on Fatrop and its benchmarking compared to other OCP solvers, we refer readers to [16].

5 Constrained dynamics as an LQR problem

This section formulates the constrained dynamics problem as an equivalent equality-constrained LQR problem, which turns out to be a special case of the LQR formulation considered in Section. 3. Instead of the generalized coordinate formulation in Eq. (3), we formulate the constrained dynamics problem differently here and will shortly explain its connection to Eq. (3). We use Gauss' principle of least constraint (GPLC) [57, 58], a formulation of mechanics according to which, the acceleration of a constrained system rigid bodies under the influence of external forces is as close as possible to the unconstrained acceleration in a weighted least-squares sense. This optimization problem expressed in the so-called maximal coordinates is

$$\underset{\mathbf{a}_1,\ldots,\mathbf{a}_n,\dot{\boldsymbol{\nu}}}{\textbf{minimize}} \quad \sum_{i=1}^{n} \frac{1}{2} \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right)^T H_i \left(\mathbf{a}_i - H_i^{-1}\mathbf{f}_i\right), \tag{37a}$$

$$\textbf{subject to} \quad \mathbf{a}_i = \mathbf{a}_{i-1} + B_i\dot{\boldsymbol{\nu}}_i + \mathbf{a}_{b,i},\ i = 1, 2, \ldots, n, \tag{37b}$$

$$K_i\mathbf{a}_i = \mathbf{k}_i,\ i = 1, 2, \ldots, n, \tag{37c}$$

$$\mathbf{a}_0 = -\mathbf{a}_{\text{grav}}, \tag{37d}$$

where we follow the standard spatial algebra notation of Featherstone [35]. $\mathbf{a}_i \in \mathbb{R}^6$, $H_i \in \mathbb{S}^6_{++}$ and $\mathbf{f}_i \in \mathbb{R}^6$ are respectively the ith body's spatial accelerations, spatial inertia tensor and the spatial forces acting on the ith rigid body. $\mathbf{f}_i$ includes the bias forces $(-\mathbf{v}_i \times^* H_i\mathbf{v}_i)$, where $\mathbf{v}_i$ is the ith body's spatial velocity. $\mathbf{a}_{b,i} = \mathbf{v}_i \times B_i\boldsymbol{\nu}_i$ and $\mathbf{a}_{\text{grav}}$ are the bias accelerations and the spatial acceleration-due-to-gravity vectors. $K_i \in \mathbb{R}^{m_i \times 6}$ and $\mathbf{k}_i \in \mathbb{R}^{m_i}$ are the constraint matrix and desired constraint accelerations on the ith link. $\times$ and $\times^*$ generalizes the notion of a cross-product to spatial motion vectors and force vectors. The spatial algebra notation enables treating 6D motions and forces as elements of a dual vector-space, enabling their addition and subtraction. This greatly simplifies deriving and implementing dynamics algorithms. We refer readers to [35] for a detailed introduction. All the physical quantities here are defined w.r.t. and expressed in an inertial frame for notational simplicity.

Connection to Eq. (3): The M and $\mathbf{c}$ terms are typically computed from the $\boldsymbol{q}$, H_i, and $\boldsymbol{\nu}$ terms. Efficient algorithms to do so are RNEA [59] and CRBA [46] for $\mathbf{c}$ and M respectively, as explained in [35, Section 6.1]. The J, $\dot{J}$ and $\mathbf{a}_c^*$ terms from Eq. (3) are also typically computed from the K_i and $\mathbf{k}_i$ terms above using each constrained link's kinematic Jacobian as explained in [14, Eq. 3]. Finally, the connection between $\mathbf{f}_i$ above and $\boldsymbol{\tau}$ is shown in [14, Eq. 23]. Note that the formulation above can be considered to be a lower-level formulation of the constrained dynamics problem since the terms of the generalized coordinate formulation in Eq. (3) are computed from the terms in formulation above.

Connecting GPLC to the LQR problem: The objective function in (37a) can be further simplified after ignoring constant terms to get

$$\sum_{i=1}^{n} \frac{1}{2}\mathbf{a}_i^T H_i \mathbf{a}_i - \mathbf{a}_i^T \mathbf{f}_i,$$

comparing which to (15), we see that the GPLC is a special case of the equality-constrained LQR problem in Section 3 with

$$\begin{aligned} \mathbf{x}_i = \mathbf{a}_{i-1},\ \mathbf{u}_i = \dot{\boldsymbol{\nu}}_i,\ Q_i = H_i,\ R_i = 0_{n_u \times n_u},\ S_i = 0_{6\times n_u},\ \mathbf{q}_i = -\mathbf{f}_i,\ \mathbf{r}_i = 0_{n_u}, \\ A_i = I_{6\times 6},\ B_i = B_i,\ \mathbf{b}_i = \mathbf{a}_{b,i},\ G_{k,x} = K_i,\ G_{k,u} = 0_{m_i \times n_u},\ \mathbf{g}_k = -\mathbf{k}_i. \end{aligned} \tag{38}$$

With this, Fatrop's LQR solver can be called to solve the constrained dynamics problems efficiently with a computational complexity of $O(n+m)$, where m is the number of constraints on the system.

Comparison with existing constrained dynamics solvers: Constrained dynamics-LQR connection was already made in the 1970s, and used to derive the PV algorithm in [60, 38]. However, their algorithm first eliminated all the primal variables and then the dual variables to get a hig computational complexity of $O(n+m^2d+m^3)$, where d is the kinematic tree depth. However, this LQR connection was sparsely known in the robotics community, resulting in most existing simulators using the relatively more inefficient LTL algorithms [36, 37] with the worst case computational complexity of $O(nd^2+m^2d+dm^2+m^3)$. The LQR connection had not been exploited to solve constrained dynamics until our recent work [14], which proposed a closely related Riccati recursion algorithm called PV-early algorithm with the computational complexity of $O(n+m)$ similarly to FATROP's LQR solver. The PV-early algorithm is currently the most efficient algorithm reported in robot dynamics literature. It can provide over 2x speed-up compared to the LTL algorithms for humanoid robots. Readers are referred to [14] for more detailed benchmarking results.

The primary difference between the PV-early algorithm and Fatrop's LQR solver, is that PV-early computes the Schur complement to eliminate $\mathbf{u}_i$ before eliminating the constraints, whereas this order is reversed in Section 3. PV-early's algorithm is always feasible for mechanics problems since P_i from Section 3 is guaranteed to be positive definite as it corresponds to an inertial quantity, whereas Fatrop's linear solver makes fewer assumptions by requiring only the reduced Hessian to be positive definite [12]. We refer readers to [14] for insight on the physical interpretation of the Riccati recursion terms when applied to dynamics problems. This suggests that Fatrop's LQR solver could be more efficient than PV-early because it eliminates the smaller $\tilde{\mathbf{u}}_i$ vector during the Schur complement step. Therefore, it is an interesting future research direction to implement a specialized version of Fatrop's LQR solver to compute constrained dynamics problems.

6 Efficient constrained dynamics derivatives

We now apply the implicit function theorem and the adjoint method covered in Sec 2.4 to efficiently compute the constrained dynamics Jacobian and the constrained dynamics' contribution to the Lagrangian's gradient and Hessian. We start with the unconstrained FD Jacobian, before moving on to the more complex constrained setting. Finally, we discuss computing the first and the second derivatives of the Lagrangian function.

Unconstrained FD: Similarly to [13], let us define an implicit function

$$\mathbf{g}_u(\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau}, \dot{\boldsymbol{v}}) = \mathrm{ID}(\boldsymbol{q}, \boldsymbol{v}, \dot{\boldsymbol{v}}) - \boldsymbol{\tau} = \mathbf{0}_n, \tag{39}$$

comparing which with the implicit function $\mathbf{g}(\mathbf{x}, \mathbf{y}(\mathbf{x}))$ introduced in Sec 2.4, $\mathbf{x} = [\boldsymbol{q}^T, \boldsymbol{v}^T, \boldsymbol{\tau}^T]^T$ and $\mathbf{y} = \dot{\boldsymbol{v}} = \mathrm{FD}(\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau})$. This choice of $\mathbf{g}_u$ is valid since it is zero everywhere by definition and $\frac{\partial \mathbf{g}_u}{\partial \mathbf{y}} = \frac{\partial \mathrm{ID}}{\partial \dot{\boldsymbol{v}}} = M$ is positive definite and hence invertible. Applying the implicit function theorem from (7) readily gives

$$\frac{\partial \mathrm{FD}}{\partial\, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau}} = -M^{-1} \frac{\partial \mathbf{g}_u}{\partial\, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau}}. \tag{40}$$

Evaluating the right-hand-side of the expression above involves differentiating ID, which is significantly cheaper than directly differentiating FD functions. $M^{-1} = \frac{\partial \mathrm{FD}}{\partial \boldsymbol{\tau}}$ is obtained by differentiating FD (the PV-early algorithm in our case) w.r.t. $\boldsymbol{\tau}$, which is an exception to the rule that differentiating through FD is expensive. Computing M^{-1} this way is more efficient than computing M and factorizing/inverting it.

Constrained FD: Analogously to the previous section, we define an implicit function

$$\mathbf{g}_c(\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau}, \boldsymbol{\lambda}, \dot{\boldsymbol{v}}) = \begin{pmatrix} \mathbf{f}_{c,a}(\boldsymbol{q}, \boldsymbol{v}, \dot{\boldsymbol{v}}) \\ \mathrm{ID}_c(\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\lambda}, \dot{\boldsymbol{v}}) - \boldsymbol{\tau} \end{pmatrix} = \mathbf{0}_{n+m}, \tag{41}$$

where $\mathbf{f}_{c,a}$ encodes motion constraints at the acceleration level from Eq. (3b). Comparing the equation above with the implicit function $\mathbf{g}(\mathbf{x}, \mathbf{y}(\mathbf{x}))$ from Section. 2.4, we have $\mathbf{x} = [\boldsymbol{q}^T, \boldsymbol{v}^T, \boldsymbol{\tau}^T]^T$ and $\mathbf{y} = [\boldsymbol{\lambda}^T, \dot{\boldsymbol{v}}^T]^T = \mathrm{FD}_c(\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau})$. Similarly to the unconstrained setting, $\mathbf{g}_c$ is zero everywhere by definition, and we have that

$$\frac{\partial \mathbf{g}_c}{\partial\, \mathbf{y}} = \begin{pmatrix} 0_{m\times m} & J \\ J^T & M \end{pmatrix}, \tag{42}$$

is invertible from Note 1 and positive definiteness of M. Applying the implicit function theorem from (7) gives

$$\frac{\partial \mathrm{FD}_c}{\partial\, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau}} = -\begin{pmatrix} 0_{m\times m} & J \\ J^T & M \end{pmatrix}^{-1} \frac{\partial \mathbf{g}_c}{\partial\, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{\tau}}. \tag{43}$$

The matrix $\begin{pmatrix} 0_{m\times m} & J \\ J^T & M \end{pmatrix}^{-1}$ can be efficiently computed by evaluating the term $\frac{\partial \mathrm{FD}_c}{\partial \begin{bmatrix} \mathbf{k}^T & \boldsymbol{\tau}^T \end{bmatrix}^T}$ using the PV-early solver. $\mathbf{k}$ is the concatenated vector of all the right-hand-side of the motion constraints from Eq. 37d.

Gradient and Hessian of the OCP Lagrangian: The OCP solver in (34) requires the Lagrangian Hessian $\nabla^2_{\mathbf{xx}}\mathcal{L}$. Computing this term by directly differentiating through constrained FD algorithms is expensive. The adjoint method mitigates this issue and can be readily applied using Eqs. 11 and 14. The f in these equations refers to the FD function's contribution to the Lagrangian $\mathcal{L}$. For example, $f = \boldsymbol{\eta}^T\dot{\boldsymbol{\nu}}\Delta t$ for an explicit Euler integrator scheme, where $\boldsymbol{\eta}$ is the corresponding Lagrangian multiplier and Δt is the time-step. Constrained FD can also be a part of the objective function, e.g., suppose user chooses to regularize $\boldsymbol{\nu}$. Then, for the adjoint method $f = \dot{\boldsymbol{\nu}}^T\dot{\boldsymbol{\nu}}$. It can be more complex for other integration methods like exponential map of spatial velocity [61]. However, the key aspect is that $\frac{\partial^2(f-\mathbf{g}_c^T\tilde{\boldsymbol{\mu}})}{\partial \mathbf{z}^2}$ and $\frac{\partial f}{\partial \mathbf{y}}$ can be computed conveniently using AD software. The remaining terms needed to compute the Lagrangian gradient and Hessian are already available from the Jacobian computation ($\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$) discussed above in both unconstrained and constrained settings. The proposed adjoint method is listed as an algorithm in Algorithm 1, where the notation $.\mathbf{z}$ implies that the elements of $\mathbf{z}$ are appropriately unpacked as inputs to the corresponding functions.

Algorithm 1 Adjoint method

with functions $\mathrm{FD}_c(\boldsymbol{q},\ \boldsymbol{\nu},\ \boldsymbol{\tau}, \mathbf{k})$, $\mathbf{g}_c(\boldsymbol{q},\ \boldsymbol{\nu},\ \boldsymbol{\tau}\ \dot{\boldsymbol{\nu}},\ \boldsymbol{\lambda}, \mathbf{k})$, $f(\boldsymbol{q},\ \boldsymbol{\nu},\ \boldsymbol{\tau},\ \dot{\boldsymbol{\nu}},\ \boldsymbol{\lambda},\ \boldsymbol{\eta})$, and with $\mathbf{x} = [\boldsymbol{q}^T\ \boldsymbol{\nu}^T\ \boldsymbol{\tau}^T]^T$, $\mathbf{y} = [\boldsymbol{\lambda}^T\ \dot{\boldsymbol{\nu}}^T]^T$, $\mathbf{z} = [\mathbf{x}^T\ \mathbf{y}^T]^T$

Input: $\boldsymbol{q}, \boldsymbol{\nu}, \boldsymbol{\tau}, \boldsymbol{\eta}, \mathbf{k}$

1: $\mathbf{y} \leftarrow \mathrm{FD}_c(.\mathbf{x}, \mathbf{k})$

2: Solve for $\boldsymbol{\mu}$: $\left(\left.\frac{\partial \mathbf{g}_c}{\partial \mathbf{y}}\right|_{\mathbf{z},\mathbf{k}}\right)^T \boldsymbol{\mu} = \left(\left.\frac{\partial f}{\partial \mathbf{y}}\right|_{\mathbf{z},\boldsymbol{\eta}}\right)^T$ ▷ AD

3: $\left.\frac{\partial \mathrm{FD}_c}{\partial \mathbf{x}}\right|_{\mathbf{z},\mathbf{k}} \leftarrow \left.\frac{\partial \mathrm{FD}_c}{\partial \left[\mathbf{k}^T\ \boldsymbol{\tau}^T\right]^T}\right|_{\mathbf{x},\mathbf{k}} \left.\frac{\partial \mathbf{g}_c}{\partial \mathbf{x}}\right|_{\mathbf{z},\mathbf{k}}$ ▷ AD

4: $\left.\frac{\partial^2 f}{\partial \mathbf{x}^2}(.\mathbf{z}, .\mathrm{FD}_c(.\mathbf{z}, \mathbf{k}), \boldsymbol{\eta})\right|_{\mathbf{z},\mathbf{k},\boldsymbol{\eta}} \leftarrow G^T \left.\frac{\partial^2(f-\mathbf{g}_c^T\boldsymbol{\mu})}{\partial \mathbf{z}^2}\right|_{\mathbf{z},\mathbf{k},\boldsymbol{\eta}} G$ ▷ AD

with $G = \begin{pmatrix} \boldsymbol{I} \\ \left.\frac{\partial \mathrm{FD}_c}{\partial \mathbf{x}}\right|_{\mathbf{z},\mathbf{k}} \end{pmatrix}$

Output: $\dot{\boldsymbol{\nu}}$, $\left.\frac{\partial \mathrm{FD}_c}{\partial \mathbf{x}}\right|_{\mathbf{z},\mathbf{k}}$, $\left.\frac{\partial^2 f}{\partial \mathbf{x}^2}(.\mathbf{z}, .\mathrm{FD}_c(.\mathbf{z}, \mathbf{k}), \boldsymbol{\eta})\right|_{\mathbf{z},\mathbf{k},\boldsymbol{\eta}}$

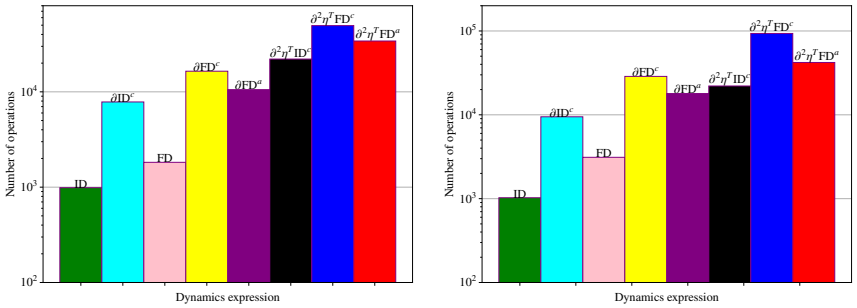
Relation to existing work: [9] also proposes an approach to compute the exact Lagrangian Hessian. It can be shown that the Hessian computation from Eq. (14) obtained using the adjoint method is equivalent to [9]'s approach. However, the derivation in [9] uses 4-D tensors and is significantly more complex than ours and involved manually differentiating FD_c w.r.t. to each term. Moreover, we allow f to be any twice-differentiable function of $\mathbf{y}$ while it was assumed to be $\boldsymbol{\eta}^T\dot{\boldsymbol{\nu}}$ in [9]. [9] proposes a modified RNEA, which requires one computation sweep compared to classical RNEA's two sweeps, to compute and differentiate $\mathbf{g}_c$. Modified RNEA was found give up to $\sim 25\%$ speed-up [44] compared to RNEA and can also be adopted in our framework when $f = \boldsymbol{\eta}\dot{\boldsymbol{\nu}}$. For simplicity, this is left for future work.

7 Computational Benchmarking

We implemented a point-to-point motion (P2P) trajectory optimization task for a 7 DoF Kuka LBR iiwa arm using FATROP [16] and PV-early algorithm [14] as the OCP solver and CDA respectively. The Lagrangian's derivatives are computed using the adjoint method from Section 6 and the AD tool CASADI [17]. Computational timings are benchmarked on a `13th Gen Intel® Core™ i9-13950HX` laptop CPU running an `Ubuntu 22.04LTS` operating system.

Since previous papers [16, 14] have benchmarked FATROP and PV-early algorithm, we benchmark the adjoint method in this section. We first compare the operation counts for various functions in unconstrained and constrained settings, followed by description of the point-to-point motion problem on which we subsequently benchmark the OCP solution timings.

Operation counts of derivative computation:



(a) iiwa - unconstrained case.

(b) iiwa - 6D constraint on the end-effector.

Fig. 1: Operation count for different dynamics functions and their derivatives.

Table 1: Computational speed-up obtained by the adjoint method compared to CASADI.

	Unconstrained	Constrained
∂FD	1.56x	1.60x
$\partial^2 \eta^T$FD	1.45x	2.21x

Figure 1 lists the number of operations (floating-point addition, multiplication, division, sin/cos, etc.) required to compute the different dynamics functions, where we used CASADI to get the operation count. ∂ID^c and ∂FD^c refer to ID and FD

Jacobians respectively computed by CasADi, while ∂FD^a refers to FD Jacobian computed using the implicit function approach. Similarly, $\partial^2\eta^T\text{ID}^c$ and $\partial^2\eta^T\text{FD}^c$ refer to the Hessian of $\boldsymbol{\eta}^T\text{ID}$ and $\boldsymbol{\eta}^T\text{FD}$ respectively computed using CasADi, while $\partial^2\eta^T\text{FD}^a$ is computed using the adjoint method. Table 1 lists the speed-up obtained by the adjoint method compared to using only CasADi.

Benchmarking OCP solution times:

Table 2: Benchmarking the computational cost of the adjoint method while solving a point-to-point OCP for a Kuka LBR iiwa robot arm using the Fatrop solver. The cost per iteration is presented in square brackets, the percentage of cost spent in function evaluations in parentheses and the number of iterations taken in the curly brackets. All computations are performed on a single CPU core with `TurboBoost` enabled.

	Unconstrained	Constrained
Exact Hessian - CasADi	10.9 ms [0.64 ms] (65%) {17}	7.6 ms [0.95 ms] (82%) {8}
Exact Hessian - adjoint	**8.7 ms** [0.51 ms] (56%) {17}	**4.1 ms** [0.52 ms] (64%) {8}
No second-order dyn - CasADi	10.4 ms [0.37 ms] (50%) {28}	7.5 ms [0.47 ms] (67%) {19}
No second-order dyn - adjoint	8.8 ms [0.31 ms] (39%) {28}	6.4 ms [0.34 ms] (52%) {19}

We now benchmark the adjoint method's influence on OCP solution timings. Consider a point-to-point motion task for a 7 DoF Kuka LBR iiwa robot arm, with the desired joint position imposed with a terminal equality constraint. Stage-wise inequality constraints were imposed on the joint positions, velocities and torques, leading to 21 inequality constraints per stage. Explicit Euler method was used to simulate the system forward by a time-step of 50 ms, for a total of 50 steps in the OCP to get a total horizon period of 2.5 s. Stage-wise costs were simply quadratic regularization on the joint torques and joint velocities. We tested both the exact-Hessian method, where the second derivatives of the dynamics terms are computed and used in the Lagrangian, and the Hessian approximation, where the second derivatives of the dynamics terms are omitted in the Lagrangian Hessian. We tested the method for unconstrained FD, where the arm is moving in free-space, and also in a constrained setting, where the end-effector is rigidly fixed with a 6D constraint. In the constrained dynamics setting, the terminal equality constraint for the desired joint position was relaxed with a quadratic penalty as the constraint prevented it from reaching desired goal joint position exactly.

Table 2 lists the obtained results. The adjoint method provides a sizeable speed-up in the OCP solving compared to using only CasADi both for the exact Hessian and the approximate Hessian method. The adjoint method's speed-up method was higher for the constrained problem, matching the observation in the Table 1. As expected, the exact Hessian method took fewer iterations to converge. However, in contrast to the conventional wisdom in robotics, we found that the exact Hessian method was even faster than the approximate Hessian method, when using the adjoint method. Table 3 also compares the first and second-order methods as the weight on torque

regularization is increased for the unconstrained dynamics setting. Higher weight makes the problem more challenging since it incentivizes minimizing the highly nonlinear gravity compensation torques over the horizon. For higher weights, exact Hessian approach converges much faster than the approximate Hessian method.

Table 3: Comparing the exact Hessian method with the approximate Hessian method for an that becomes increasingly challenging for higher torque regularization weight. Computation times are presented in ms and the iteration count in parentheses. For higher weights, the approximate Hessian even exceeded maximum iterations limit.

Torque regularization weight	1e-3	2e-3	5e-3	1e-2	5e-2
No second-order dyn - adjoint	8.9 ms (28)	15.0 ms (53)	19.4 ms (63)	22.9 ms (75)	57 ms (212)
Exact Hessian - adjoint	8.8 ms (17)	9.0 ms (18)	13.8 ms (28)	15.8 ms (32)	16.7 ms (32)

Comparison with a general purpose solver: Not to belabor the point that structure-exploiting solvers are fast, we now compare FATROP with CASADI-IPOPT on the point-to-point motion OCP in unconstrained setting with a torque regularization weight of 1e-3. CASADI-IPOPT with default settings takes about 5.5 seconds (625x more expensive than FATROP-Adjoint method). Switching to using purely `SX` CASADI expressions brings the computation time down to 156 ms (18x more expensive). This high cost is due to a combination of IPOPT being slow as well as dynamics functions not being code-generated and compiled. Because CASADI code-generates OCPs without modularity, this resulted in C-files larger than 100 MB making their compilation infeasible. Manually installing the non-default linear solvers like MA57 [62] can further improve IPOPT's performance, however Riccati recursion has been found to be over 10x faster than MA57 as well [16].

8 Discussion

The 0.5 ms computation time per OCP iteration on a single CPU-core for the 7 DoF iiwa robot when using full-order dynamics and exact Hessian method and 50 horizon steps, is considerably faster than the results in existing literature [21, 32]. [21] achieves such rates only for fewer horizon steps (30 compared to our 50 steps) despite ignoring the second-order derivatives. This speed-up was achievable due to the cumulative speed-ups brought about by all the three algorithms presented in the chapter, namely the Riccati recursion used in both FATROP's LQR solver and PV-early algorithm as well as the adjoint method.

The promising results obtained open up exciting possibilities of future application-oriented research. With our second-order methods, generating complex trajectories for a humanoid-sized robots in a few seconds instead of minutes appears to be feasible, which will naturally be the focus of our future work. There exist several low-hanging fruits that can be exploited to obtain further speed-ups, such as parallelizing

the function evaluations in our multiple-shooting formulation. With parallel function evaluation, the linear solver is likely to become the computational bottleneck and the inherently sequential Riccati recursion is not trivially parallelizable.

Inverse dynamics versus forward dynamics: Recent works [63, 64] have explored ID-based OCP formulations, in which case expensive constrained FD algorithms and the adjoint method are not required. However, this approach models contact forces as additional control inputs and the motion constraints as OCP constraints, making function evaluations cheaper at the expense of more expensive Riccati recursion due to a larger KKT system. With Riccati recursion being more difficult to parallelize than function evaluation, it is unclear if using ID is overall a better choice and requires further investigation.

Limitations: Some limitations of the current approach include that we rely on CasADi's C-code generation capabilities for evaluating the derivatives efficiently. For larger robots like humanoids, this leads to large code-generated files of about 100 MB, which are prohibitively time-consuming to compile and also suffer from reduced efficiency of the compiled code. Therefore a recursive implementation of the Hessian computation similarly to the dynamics derivatives implementation in C++ like in Pinocchio can reduce the compilation time or even eliminate this inconvenient step. We assumed linear independence of the motion constraints, which can be relaxed using the proximal algorithms explored in [49], which we will be our future work.

9 Conclusions

We derived three state-of-the-art algorithms for efficiently solving whole-body robot OCPs, pertaining to solving the OCP's KKT system, computing constrained dynamics and differentiating the dynamics functions. Each of these algorithms can be used stand-alone in different applications. Implementing them efficiently and combining them together provides a particularly efficient robot trajectory optimization framework due to the cumulative benefits. The adjoint method provided over 1.5x speed-up in unconstrained setting and over 2x in constrained setting compared to CasADi for differentiating Kuka LBR iiwa's dynamics. This speed-up appears to make the exact Hessians competitive with the first order dynamics approximations, which is contrary to the conventional wisdom in robotics. Since exact Hessian methods often converge more reliably and in fewer iterations than inexact Hessian methods for challenging NLPs, as was also confirmed in our results, our contribution encourages their usage for robot TO.

Compared to existing structure-exploiting OCP solvers, Fatrop provides state-of-the-art generality (by supporting equality/inequality constraints that are both stage-wise and terminal), constraint-handling (its nonlinear interior point method ensuring constraint satisfaction), reliability (exact Hessian and it implements most of IPOPT's advanced globalization strategies) and speed (fast computation timings due to its efficient BLASFEO implementation of the LQR solver). This chapter, by combining

FATROP and efficient dynamics and exact Hessian computation, offers a promising framework to unlock fast robot trajectory optimization applications in challenging tasks for high-dimensional robots like quadrupeds and humanoids. Such applications will naturally be the focus of our future work along with overcoming some limitations discussed in the previous section.

Acknowledgements The authors gratefully acknowledge funding from European Research Council (ERC) Advanced Grant (GA: ROBOTGENSKILL No. 788298), Research Foundation Flanders (FWO) (GA No. G0D1119N) and FlandersMake SBO project - ARENA, and by the European Union through the AGIMUS project (GA no.101070165). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Competing Interests The authors have no conflicts of interest to declare that are relevant to the content of this chapter.

References

1. John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.
2. James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, 2017.
3. Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton university press, 2011.
4. Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009.
5. Hans Georg Bock. Recent advances in parameteridentification techniques for ode. In *Numerical Treatment of Inverse Problems in Differential and Integral Equations: Proceedings of an International Workshop, Heidelberg, Fed. Rep. of Germany, August 30—September 3, 1982*, pages 95–121. Springer, 1983.
6. Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.
7. David A Benson, Geoffrey T Huntington, Tom P Thorvaldsen, and Anil V Rao. Direct trajectory optimization and costate estimation via an orthogonal collocation method. *Journal of Guidance, Control, and Dynamics*, 29(6):1435–1440, 2006.
8. Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.
9. John N Nganga and Patrick M Wensing. Accelerating hybrid systems differential dynamic programming. *ASME Letters in Dynamic Systems and Control*, 3(1):011002, 2023.
10. Wilson Jallet, Antoine Bambade, Etienne Arlaud, Sarah El-Kazdadi, Nicolas Mansard, and Justin Carpentier. Proxddp: Proximal constrained trajectory optimization. 2023.
11. David Q Mayne. Differential dynamic programming–a unified approach to the optimization of dynamic systems. In *Control and dynamic systems*, volume 10, pages 179–254. Elsevier, 1973.
12. Lander Vanroye, Joris De Schutter, and Wilm Decré. A generalization of the riccati recursion for equality-constrained linear quadratic optimal control. *Optimal Control Applications and Methods*, n/a(n/a), 2023.
13. Justin Carpentier and Nicolas Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Proc. Robot., Sci. Syst.*, 2018.
14. 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*, pages 1–20, 2023.
15. Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. acados—a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14(1):147–183, 2022.
16. Lander Vanroye, Ajay Sathya, Joris De Schutter, and Wilm Decré. Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10036–10043, 2023.
17. Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.
18. 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 *2019 IEEE/SICE International Symposium on System Integration (SII)*, pages 614–619. IEEE, 2019.

19. Andrew M Bradley. Pde-constrained optimization and the adjoint method. 2019. *URL https://cs. stanford. edu/˜ ambrad/adjoint_tutorial. pdf*, 2021.
20. Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
21. Sébastien Kleff, Avadesh Meduri, Rohan Budhiraja, Nicolas Mansard, and Ludovic Righetti. High-frequency nonlinear model predictive control of a manipulator. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7330–7336. IEEE, 2021.
22. Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.
23. Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.
24. Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.
25. Gianluca Frison, Dimitris Kouzoupis, Tommaso Sartor, Andrea Zanelli, and Moritz Diehl. Blasfeo: Basic linear algebra subroutines for embedded optimization. *ACM Transactions on Mathematical Software (TOMS)*, 44(4):1–30, 2018.
26. Markus Giftthaler, Michael Neunert, Markus Stäuble, and Jonas Buchli. The control toolbox—an open-source c++ library for robotics, optimal and model predictive control. In *2018 IEEE international conference on simulation, modeling, and programming for autonomous robots (SIMPAR)*, pages 123–129. IEEE, 2018.
27. 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.
28. Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2536–2542. IEEE, 2020.
29. Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on control and optimization*, 43(5):1714–1736, 2005.
30. Farbod Farshidian et al. OCS2: An open source library for optimal control of switched systems. [Online]. Available: https://github.com/leggedrobotics/ocs2.
31. Sotaro Katayama and Toshiyuki Ohtsuka. Structure-exploiting newton-type method for optimal control of switched systems. *International Journal of Control*, (just-accepted):1, 2023.
32. Armand Jordana, Sébastien Kleff, Avadesh Meduri, Justin Carpentier, Nicolas Mansard, and Ludovic Righetti. Stagewise implementations of sequential quadratic programming for model-predictive control. 2023.
33. AF Vereshchagin. Computer simulation of the dynamics of complicated mechanisms of robot-manipulators. *Eng. Cybernet.*, 12:65–70, 1974.
34. Roy Featherstone. The calculation of robot dynamics using articulated-body inertias. *Int. J. Robot. Res.*, 2(1):13–30, 1983.
35. Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.
36. Roy Featherstone. Efficient factorization of the joint-space inertia matrix for branched kinematic trees. *Int. J. Robot. Res.*, 24(6):487–500, 2005.
37. Roy Featherstone. Exploiting sparsity in operational-space dynamics. *Int. J. Robot. Res.*, 29(10):1353–1368, 2010.
38. Anatolii Fedorovich Vereshchagin. Modeling and control of motion of manipulational robots. *Soviet Journal of Computer and Systems Sciences*, 27(5):29–38, 1989.
39. Dae-Sung Bae and Edward J Haug. A recursive formulation for constrained mechanical system dynamics: Part ii. closed loop systems. *Journal of Structural Mechanics*, 15(4):481–506, 1987.
40. M Otter, H Brandl, and R Johanni. An algorithm for the simulation of multibody systems with kinematic loops. In *Proceedings of the 7th World Congress on Theory of Machines and Mechanisms, IFToMM, Sevilla, Spain*, 1987.

41. Quentin Le Lidec, Wilson Jallet, Louis Montaut, Ivan Laptev, Cordelia Schmid, and Justin Carpentier. Contact models in robotics: a comparative analysis. *arXiv preprint arXiv:2304.06372*, 2023.
42. Brad Bell. Cppad: a package for c++ algorithmic differentiation, 2018.
43. Shubham Singh, Ryan P Russell, and Patrick M Wensing. On second-order derivatives of rigid-body dynamics: Theory & implementation. *IEEE Transactions on Robotics*, 2024.
44. John N Nganga and Patrick M Wensing. Accelerating second-order differential dynamic programming for rigid-body systems. *IEEE Robotics and Automation Letters*, 6(4):7659–7666, 2021.
45. John N Nganga, He Li, and Patrick M Wensing. Second-order differential dynamic programming for whole-body mpc of legged robots. *IFAC-PapersOnLine*, 56(3):499–504, 2023.
46. Michael W Walker and David E Orin. Efficient dynamic computer simulation of robotic mechanisms. 1982.
47. Helmut Brandl, Rainer Johanni, and Martin Otter. A very efficient algorithm for the simulation of robots and similar multibody systems without inversion of the mass matrix. *IFAC Proceedings Volumes*, 19(14):95–100, 1986.
48. Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proc. IEEE/RSJ Int. Conf. Int. Robots. Syst.*, pages 5026–5033. IEEE, 2012.
49. Justin Carpentier, Rohan Budhiraja, and Nicolas Mansard. Proximal and sparse resolution of constrained dynamic equations. In *Proc. Robot., Sci. Syst.*, 2021.
50. Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. *IEEE Robot. Autom. Lett.*, 3(2):895–902, 2018.
51. Steven George Krantz and Harold R Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.
52. Markus Giftthaler and Jonas Buchli. A projection approach to equality constrained iterative linear quadratic optimal control. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 61–66. IEEE, 2017.
53. Athanasios Sideris and Luis A Rodriguez. A riccati approach to equality constrained linear quadratic optimal control. In *Proceedings of the 2010 American Control Conference*, pages 5167–5172. IEEE, 2010.
54. Alexander Domahidi, Aldo U Zgraggen, Melanie N Zeilinger, Manfred Morari, and Colin N Jones. Efficient interior point methods for multistage problems arising in receding horizon control. In *2012 IEEE 51st IEEE conference on decision and control (CDC)*, pages 668–674. IEEE, 2012.
55. Forrest Laine and Claire Tomlin. Efficient computation of feedback control for equality-constrained lqr. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 6748–6754. IEEE, 2019.
56. Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.
57. Carl Friedrich Gauß. Über ein neues allgemeines grundgesetz der mechanik. 1829.
58. Herman Bruyninckx and Oussama Khatib. Gauss' principle and the dynamics of redundant and constrained manipulators. In *Proc. IEEE Int. Conf. Robot. Autom.*, volume 3, pages 2563–2568. IEEE, 2000.
59. John YS Luh, Michael W Walker, and Richard PC Paul. On-line computational scheme for mechanical manipulators. 1980.
60. Je P Popov, Anatolij Fedorovič Vereshchagin, and Stanislav Leonidovič Zenkevič. *Manipuljacionnyje roboty: Dinamika i algoritmy*. Nauka, 1978.
61. Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.
62. Iain S Duff. Ma57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software (TOMS)*, 30(2):118–144, 2004.
63. Sotaro Katayama and Toshiyuki Ohtsuka. Efficient solution method based on inverse dynamics for optimal control problems of rigid body systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2070–2076. IEEE, 2021.
64. Carlos Mastalli, Saroj Prasad Chhatoi, Thomas Corbéres, Steve Tonneau, and Sethu Vijayakumar. Inverse-dynamics mpc via nullspace resolution. *IEEE Transactions on Robotics*, 2023.