Accelerating Second-Order Differential Dynamic Programming for Rigid-Body Systems

John N. Nganga¹ and Patrick M. Wensing¹

Abstract—This letter presents a method to reduce the computational demands of including second-order dynamics sensitivity information into the Differential Dynamic Programming (DDP) trajectory optimization algorithm. An approach to DDP is developed where all the necessary derivatives are computed with the same complexity as in the iterative Linear Quadratic Regulator (iLQR). Compared to linearized models used in iLQR, DDP more accurately represents the dynamics locally, but it is not often used since the second-order derivatives of the dynamics are tensorial and expensive to compute. This work shows how to avoid the need for computing the derivative tensor by instead leveraging reverse-mode accumulation of derivative information to compute a key vector-tensor product directly. We also show how the structure of the dynamics can be used to further accelerate these computations in rigid-body systems. Benchmarks of this approach for trajectory optimization with multi-link manipulators show that the benefits of DDP can often be included without sacrificing evaluation time, and can be done in fewer iterations than iLQR.

Index Terms—Optimization and Optimal Control; Underactuated Robots; Whole-Body Motion Planning and Control

I. INTRODUCTION

N recent years, online optimal control strategies have gained widespread interest in many applications from motion planning of robots to control of chemical processes [1]–[6]. Rather than relying on manually derived policies, these control strategies optimize a metric of cost that encodes desired task goals. This approach then allows online control performance that is generalizable across tasks or environments. For example, online optimization may enable legged systems to tailor their gaits to sensed terrains and inevitable disturbances or may enable manipulators to rapidly synthesize efficient motions when transporting new objects.

However, for robots with even a few links, the underlying system dynamics are complex, nonlinear, and expensive to evaluate. These features challenge the ability to solve trajectory optimization problems online, particularly in systems with a large number of degrees of freedom (DoFs). Yet, the motivation to perform online optimization is often greater for these very systems, since a high DoF morphology gives the needed flexibility and mobility to adapt to a wider range of situations. Since the curse of dimensionality precludes the ability of exploring the full state space, online optimal control strategies often settle on exploring within a local neighborhood. Even then, the optimization of trajectories is often orders of magnitude slower than real-time. For many

Digital Object Identifier (DOI): see top of this page.

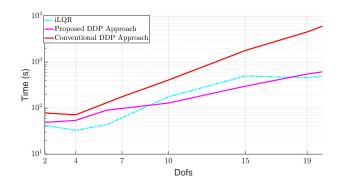


Fig. 1. Trajectory optimization time in MATLAB for an underactuated serial manipulator swing-up task vs. DoFs. All cases start from an initial guess of a dissipative controller. Pink curve: Proposed tensor-free second-order DDP. Red curve: Second-order DDP with explicit computation of second-order dynamics derivative tensors. Blue curve: iLQR method that approximates DDP using first-order dynamics derivatives to save computation time.

years, control approaches in the legged robotics literature have sidestepped this burden by employing simple models to enable faster computation [1].

Recently, whole-body trajectory optimization is becoming more feasible and has gained increased interest as computation power progresses [4], [7]–[9]. For example, [10] used DDP [11] for a humanoid to perform complex tasks such as getting up from an arbitrary pose. DDP exploits the sparsity of an optimal control problem (OCP), and its output includes an optimal trajectory along with a locally optimal feedback policy that can be used to handle disturbances [9]. While DDP natively does not address constraints, many recent approaches using Augmented Lagrangian [12], [13], interior point [14], and relaxed barrier strategies [9], [15] have been proposed to handle general state and control constraints, with specialized approaches considered for control limit constraints [16], [17]. Other work [18]-[21] has considered multi-threading and parallelization of the DDP algorithm to accelerate its computation. Finally, Li et al. [22] combine the advantages of whole-body DDP and simple models by sequentially considering both over the horizon. Collectively, these previous works show broad potential impact from advances to numerical methods for DDP.

Originally described in [11], DDP uses a second-order approximation of the dynamics when constructing a second-order approximation of the optimal cost-to-go. However, in practice (e.g., [7], [10], [23]) many researchers have opted to use a first-order dynamics approximation due to its faster evaluation time, giving rise to the iterative Linear Quadratic Regulator (iLQR). While the second-order dynamics information retains higher fidelity to the full model locally, it is represented by a rank three tensor and is expensive to compute. In this work, we alleviate these computational demands by

¹Authors are with the Department of Mechanical Engineering, University of Notre Dame, Notre Dame, IN 46556, USA {jnganga,pwensing}@nd.edu

describing a new approach that avoids the evaluation of the dynamics derivative tensor (Fig. 1).

A. Specific Contributions

This work presents and combines several advances for reducing the computational complexity of computing secondorder dynamics sensitivity information in DDP. The final result is a method for computing this information with the same computational complexity as first-order dynamics derivatives (e.g., as in iLQR). The contributions are (I) the use of reversemode automatic differentiation (AD) [24] to compute secondorder derivatives needed in DDP. This contribution is general to discrete-time dynamic systems and enables computation reductions compared to methods that explicitly evaluate a derivative tensor for the dynamics. Further, we show (II) how second-order information related to the forward dynamics can be related to associated information from the inverse dynamics, akin to first-order results in [25], [26]; this contribution is specific to rigid-body models. Lastly, we (III) introduce a modification to the Recursive-Newton-Euler Algorithm (RNEA) that supports this process and further reduces computational demands. Figure 1 overviews a benchmark of the proposed methods against iLQR and against DDP approaches that explicitly compute derivative tensors.

II. TRAJECTORY OPTIMIZATION VIA DDP/ILQR

This work considers the efficient solution of a finite-horizon OCP for a rigid-body system such as an articulated robot. This section reviews background on dynamics and trajectory optimization with a focus on the DDP algorithm.

A. Dynamics

The inverse dynamics (ID) of a rigid-body system are

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \tau_a(\mathbf{q}) \triangleq \mathrm{ID}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{a}_a),$$
 (1)

where $\mathbf{M}(\mathbf{q})$ represents the mass matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ Coriolis and centrifugal terms, and $\tau_g(\mathbf{q})$ the gravity term with \mathbf{a}_g the gravitational acceleration. We group the Coriolis, centrifugal, and gravity terms together as $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \tau_g(\mathbf{q})$. The RNEA [27], [28] can evaluate ID with $\mathcal{O}(n)$ complexity where n is the number of DoFs in the system. The forward dynamics (FD) of the system can be formulated as

$$\ddot{\mathbf{q}} = \mathbf{M}(\mathbf{q})^{-1} (\boldsymbol{\tau} - \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})) \triangleq \mathrm{FD}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}, \mathbf{a}_q).$$

When the fourth argument is omitted for ID or FD, gravity of 9.81 m/s^2 downward is assumed. The Articulated-Body Algorithm (ABA) [28] can compute FD in $\mathcal{O}(n)$ complexity and is an efficient alternative to $\mathcal{O}(n^3)$ algorithms that calculate and invert the mass matrix to carry out FD (e.g., [29]). Continuous trajectories for the state $\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$ and control input $\boldsymbol{\tau}$ are discretized herein using a numerical integration scheme. While the strategies proposed are applicable for use with any explicit integration scheme, forward Euler integration is assumed to simplify the remaining development such that:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \triangleq \mathbf{x}_k + h \begin{bmatrix} \dot{\mathbf{q}} \\ \mathrm{FD}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}) \end{bmatrix},$$
 (2)

where h is the integration stepsize.

B. Differential Dynamic Programming

Herein, DDP [11] and iLQR [23] are used to solve an OCP with a cost function of the form

$$V_0(\mathbf{x}_0, \mathbf{U}_0) = \ell_f(\mathbf{x}_N) + \sum_{k=0}^{N-1} \ell_k(\mathbf{x}_k, \mathbf{u}_k)$$
(3)

where $\ell_k(\mathbf{x}_k, \mathbf{u}_k)$ represents the running cost, $\ell_f(\mathbf{x}_N)$ represents the terminal cost incurred at the end of a horizon, and $\mathbf{U}_0 = [\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{N-1}]$ is the control sequence over the horizon. A cost-to-go function $V_k(\mathbf{x}_k, \mathbf{U}_k)$ can be similarly defined from any time point as the partial sum of costs from time k to N. The cost-to-go function $V_0(\mathbf{x}_0, \mathbf{U}_0)$ in (3) is minimized with respect to \mathbf{U}_0 , with states subject to the discrete system dynamics (2), providing

$$\mathbf{U}_0^{\star}(\mathbf{x}_0) = \underset{\mathbf{U}_0}{\operatorname{argmin}} \ V_0(\mathbf{x}_0, \mathbf{U}_0).$$

Throughout the paper, the star superscript refers to an optimal value. DDP optimizes over each control \mathbf{u}_i separately, working backwards in time by approximating Bellman's equation

$$V_k^{\star}(\mathbf{x}_k) = \min_{\mathbf{u}_k} \left[\underbrace{\ell_k(\mathbf{x}_k, \mathbf{u}_k) + V_{k+1}^{\star}(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k))}_{Q_k(\mathbf{x}_k, \mathbf{u}_k)} \right]$$
 (4)
where $V_N^{\star}(\mathbf{x}_N) = \ell_f(\mathbf{x}_N)$.

Consider a perturbation to Q_k due to small perturbations around a nominal state-control pair $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{u}}_k$ such that

$$\delta Q_k(\delta \mathbf{x}_k, \delta \mathbf{u}_k) = Q_k(\bar{\mathbf{x}}_k + \delta \mathbf{x}_k, \bar{\mathbf{u}}_k + \delta \mathbf{u}_k) - Q_k(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k).$$

Expanding the δQ -function to second order leads to

$$\delta Q_k \approx \frac{1}{2} \begin{bmatrix} 1\\ \delta \mathbf{x}_k\\ \delta \mathbf{u}_k \end{bmatrix}^T \begin{bmatrix} 0 & Q_{\mathbf{x}}^T & Q_{\mathbf{u}}^T\\ Q_{\mathbf{x}} & Q_{\mathbf{xx}} & Q_{\mathbf{ux}}^T\\ Q_{\mathbf{u}} & Q_{\mathbf{ux}} & Q_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} 1\\ \delta \mathbf{x}_k\\ \delta \mathbf{u}_k \end{bmatrix}$$
(5)

where subscripts indicate partial derivatives. Omitting the time index k for conciseness, these quantities are given by:

$$Q_{\mathbf{x}} = \ell_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^T V_{\mathbf{x}}' \tag{6a}$$

$$Q_{\mathbf{u}} = \ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^T V_{\mathbf{x}}' \tag{6b}$$

$$Q_{\mathbf{x}\mathbf{x}} = \ell_{\mathbf{x}\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^T V_{\mathbf{x}\mathbf{x}}' \mathbf{f}_{\mathbf{x}} + V_{\mathbf{x}}' \cdot \mathbf{f}_{\mathbf{x}\mathbf{x}}$$
 (6c)

$$Q_{\mathbf{u}\mathbf{u}} = \ell_{\mathbf{u}\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^{T} V_{\mathbf{x}\mathbf{x}}' \mathbf{f}_{\mathbf{u}} + V_{\mathbf{x}}' \cdot \mathbf{f}_{\mathbf{u}\mathbf{u}}$$
 (6d)

$$Q_{\mathbf{u}\mathbf{x}} = \ell_{\mathbf{u}\mathbf{x}} + \mathbf{f}_{\mathbf{u}}^{T} V_{\mathbf{x}\mathbf{x}}' \mathbf{f}_{\mathbf{x}} + V_{\mathbf{x}}' \cdot \mathbf{f}_{\mathbf{u}\mathbf{x}}.$$
 (6e)

The prime in (6) denotes the next time step, i.e., $V'_{xx} = V_{xx}(k+1)$. The last terms in (6c - 6e) denote contraction with a tensor and are ignored in iLQR, representing the main difference between DDP and iLQR. These coefficients (6) could alternatively be viewed through derivatives of the Hamiltonian

$$H_k(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell_k(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{u}),$$
 (7)

where $\lambda \triangleq V_{\mathbf{x}}'$ is the co-state vector. For example, [11]

$$Q_{\mathbf{x}} = \nabla_{\mathbf{x}} H_k(\mathbf{x}_k, \mathbf{u}_k, V_{\mathbf{x}}') \quad \text{and}$$

$$Q_{\mathbf{u}\mathbf{x}} = \nabla_{\mathbf{u}\mathbf{x}}^2 H_k(\mathbf{x}_k, \mathbf{u}_k, V_{\mathbf{x}}') + \mathbf{f}_{\mathbf{u}}^T V_{\mathbf{x}\mathbf{x}}' \mathbf{f}_{\mathbf{x}}.$$

Minimizing (5) over $\delta \mathbf{u}_k$ attains the incremental control

$$\delta \mathbf{u}_{k}^{\star} = \underset{\delta \mathbf{u}_{k}}{\operatorname{argmin}} \ \delta Q_{k}(\delta \mathbf{x}_{k}, \delta \mathbf{u}_{k})$$

$$= -\underbrace{Q_{\mathbf{u}\mathbf{u}}^{-1}Q_{\mathbf{u}}}_{:=\mathbf{K}_{k}} - \underbrace{Q_{\mathbf{u}\mathbf{u}}^{-1}Q_{\mathbf{u}\mathbf{x}}}_{:=\mathbf{K}_{k}} \delta \mathbf{x}_{k},$$
(8)

where $\delta \mathbf{x} = \mathbf{x} - \bar{\mathbf{x}}$ denotes the deviation from the nominal state. When this control is substituted in (5), the quadratic approximation of the value function can be constructed as

$$\operatorname{ER}(k) = \frac{1}{2} Q_{\mathbf{u}}^{T} Q_{\mathbf{u}\mathbf{u}}^{-1} Q_{\mathbf{u}} + \operatorname{ER}(k+1)$$
 (9a)

$$V_{\mathbf{x}}(k) = Q_{\mathbf{x}} - Q_{\mathbf{u}}^T Q_{\mathbf{u}\mathbf{u}}^{-1} Q_{\mathbf{u}\mathbf{x}}$$

$$\tag{9b}$$

$$V_{\mathbf{x}\mathbf{x}}(k) = Q_{\mathbf{x}\mathbf{x}} - Q_{\mathbf{x}\mathbf{u}}Q_{\mathbf{u}\mathbf{u}}^{-1}Q_{\mathbf{u}\mathbf{x}},\tag{9c}$$

where the ER(k) is the expected reduction in cost-to-go if $\delta \mathbf{x}_k = 0$ and \mathbf{U}_k were chosen optimally. This process is repeated until a value function approximation is obtained at time k = 0, constituting the backward sweep of DDP.

Following this backward sweep, a forward sweep proceeds by simulating the system forward in time under the incremental control policy (8), resulting in a new state-control trajectory [11]. This optimal control law (8) is typically modified by a backtracking line-search parameter, $0 < \epsilon < 1$ such that $\mathbf{u}_k = \bar{\mathbf{u}}_k - \epsilon \kappa_k - \mathbf{K}_k \delta \mathbf{x}_k$. The line-search parameter ensures that DDP/iLQR takes steps that result in a reduction of the total cost. The resulting trajectory serves as a new nominal trajectory, with the above backward and forward sweeps repeated until some convergence criteria is met.

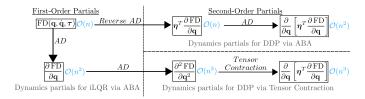
C. DDP and iLQR: Conceptual Comparison

Many factors influence the relative performance of iLQR and DDP, with the main difference again being that the tensorial terms in (6c - 6e) are ignored in iLQR. The effect is that while DDP experiences quadratic convergence for trajectories that are sufficiently close to local optimality [10], iLQR only experiences super-linear convergence (i.e., it converges more slowly). However, if the running and terminal costs are strictly convex, iLQR can be simplified relative to DDP since the terms (6c - 6e) in iLQR then ensure that Q_{uu} is always positive definite. By comparison, the addition of the tensor terms (6c - 6e) in DDP can render $Q_{\rm nm}$ indefinite, requiring regularization [10], [30], which incurs additional computational cost. The choice of DDP and iLQR in application then becomes a cost-benefit analysis among these differences with iLQR favored in recent work [7], [10], [23]. The derivatives of the dynamics are the most computationally expensive terms in DDP, motivating methods for their efficient evaluation.

III. EFFICIENT COMPUTATION OF SECOND-ORDER DERIVATIVES FOR DDP

This section considers how dynamics partials enter the partials of the Hamiltonian (7). The co-state vector is partitioned as $\lambda = [\boldsymbol{\xi}^T, \boldsymbol{\eta}^T]^T$ where $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ are the co-states associated with \mathbf{q} and $\dot{\mathbf{q}}$ respectively. Via (2), the Hamiltonian is then:

$$H_k(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}) = \ell_k(\mathbf{x}, \mathbf{u}) + \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\eta} \end{bmatrix}^T \left(\mathbf{x} + h \begin{bmatrix} \dot{\mathbf{q}} \\ \mathrm{FD}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}) \end{bmatrix} \right).$$



3

Fig. 2. Computational approach for the partials needed in *iLQR via ABA*, *DDP via ABA*, and *DDP via Tensor Contraction* methods

Focusing on the second-order partials of $H_k(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda})$ with respect to \mathbf{q} , and dropping the arguments for conciseness, the partials can be written as

$$\frac{\partial^{2} H}{\partial q_{i} \partial q_{j}} = \frac{\partial^{2} \ell}{\partial q_{i} \partial q_{j}} + h \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\eta} \end{bmatrix}^{T} \frac{\partial}{\partial q_{i}} \left(\frac{\partial}{\partial q_{j}} \begin{bmatrix} \dot{\mathbf{q}} \\ \text{FD} \end{bmatrix} \right)
= \frac{\partial^{2} \ell}{\partial q_{i} \partial q_{j}} + h \boldsymbol{\eta}^{T} \frac{\partial}{\partial q_{i}} \begin{bmatrix} \frac{\partial}{\partial q_{j}} & \text{FD} \end{bmatrix},$$
(10)

where the simplification occurs since $\partial \dot{\mathbf{q}}/\partial q_i = 0$.

In (10), the second-order partial of FD is the most expensive term to compute. It represents a bottleneck in DDP since computing it for all possible i and j results in a tensor $\frac{\partial^2 \mathrm{FD}}{\partial \mathbf{q}^2}$ with n^3 elements. These elements can be computed with total complexity $\mathcal{O}(n^3)$ before being contracted with $\boldsymbol{\eta}$ at an additional $\mathcal{O}(n^3)$ total cost. This conventional strategy is denoted as DDP via Tensor Contraction.

A. Reverse-mode Accumulation to Efficiently Compute (10)

To motivate our approach for accelerating the evaluation of (10), consider any vector-valued function $\mathbf{g}(\mathbf{x}): \mathbb{R}^n \to \mathbb{R}^m$. Given any fixed vector $\gamma \in \mathbb{R}^m$, reverse-mode derivative accumulation [24] provides an approach to evaluate the vector-Jacobian product

$$\gamma^{T} \frac{\partial \mathbf{g}}{\partial \mathbf{x}} = \sum_{j=1}^{m} \gamma_{j} \frac{\partial g_{j}}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \left[\gamma^{T} \mathbf{g} \right]$$
 (11)

with the same complexity as evaluating g(x). It is implemented in many AD packages (e.g., CasADi [31]). In practice, the cost of evaluating (11) can be bounded above by a small constant times the cost of evaluating g(x), and the constant is often three to four [24, Section 3.3].

Returning to (10), since the partials of FD are contracted with the vector $\boldsymbol{\eta}$ on the left, the desired partials can be computed efficiently using reverse-mode approaches, as diagrammed in Fig. 2. Since FD can be calculated in $\mathcal{O}(n)$, reverse-mode AD can be used to compute $\boldsymbol{\eta}^T \frac{\partial \mathrm{FD}}{\partial \mathbf{q}}$ in $\mathcal{O}(n)$ operations as well. This result is then differentiated further, achieving the necessary result $\frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\eta}^T \frac{\partial \mathrm{FD}}{\partial \mathbf{q}} \right]$ in $\mathcal{O}(n^2)$ operations – the same complexity as the first-order partials for FD itself. When partials for DDP are obtained with this approach, we denote the method as DDP via ABA since ABA is first used to evaluate FD. This general strategy also applies to any dynamic system $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ for computing $\frac{\partial}{\partial \mathbf{x}}[\lambda^T \mathbf{f}_{\mathbf{x}}] = \lambda \cdot \mathbf{f}_{\mathbf{x}\mathbf{x}}$ in DDP. While the use of reverse-mode accumulation to compute Hessians is a standard option in AD packages, its use to accelerate DDP here is new.

B. First-Order Derivatives of Rigid-Body Dynamics

The structure of rigid-body dynamics further enables efficiency gains for derivative evaluation. The iLQR and DDP algorithms require first-order derivatives of the dynamics, and these can be computed in $\mathcal{O}(n^2)$ operations with AD tools applied to the ABA. When dynamics derivatives are computed in this manner, the resulting iLQR algorithm is denoted as *iLQR via ABA*. This approach is diagrammed on the left side of Fig. 2. Derivatives of FD are, however, analytically related to the derivatives of ID by [25], [26]

$$\frac{\partial FD}{\partial \mathbf{z}}\Big|_{\mathbf{q}_0, \dot{\mathbf{q}}_0, \boldsymbol{\tau}_0} = -\mathbf{M}^{-1}(\mathbf{q}_0) \left. \frac{\partial ID}{\partial \mathbf{z}} \right|_{\mathbf{q}_0, \dot{\mathbf{q}}_0, \boldsymbol{\tau}_0} \quad \text{and} \quad (12)$$

$$\frac{\partial \operatorname{FD}}{\partial \boldsymbol{\tau}} \bigg|_{\mathbf{q}_0, \dot{\mathbf{q}}_0, \boldsymbol{\tau}_0} = \mathbf{M}^{-1}(\mathbf{q}_0) = \left(\frac{\partial \operatorname{ID}}{\partial \ddot{\mathbf{q}}} \bigg|_{\mathbf{q}_0, \dot{\mathbf{q}}_0, \ddot{\mathbf{q}}_0} \right)^{-1}, \quad (13)$$

where \mathbf{z} can either represent the vector \mathbf{q} or $\dot{\mathbf{q}}$, and where $\mathbf{q}_0, \dot{\mathbf{q}}_0, \ddot{\mathbf{q}}_0$, and $\boldsymbol{\tau}_0$ refer to the point of linearization.

These relationships provide alternate methods for the firstorder partials of FD, shown on the left side of Fig. 3. The term $\frac{\partial \operatorname{ID}}{\partial \mathbf{z}}$ can be computed in $\mathcal{O}(n^2)$ complexity with AD tools or specialized algorithms (e.g., [25], [26], [32], [33]). The explicit computation of the mass matrix inverse can be avoided in (12) by instead applying it indirectly via n calls to the ABA algorithm $(\mathcal{O}(n))$ with the columns of $\frac{\partial \operatorname{ID}}{\partial \mathbf{z}}$ as inputs for τ . This approach evaluates (12) with total complexity $\mathcal{O}(n^2)$. Since (12) relies on RNEA to obtain ID, we denote this method as iLOR via RNEA. As an alternate approach, the partials of FD are computed via (12) with $O(n^3)$ complexity as follows. The partials of ID are computed with $O(n^2)$ complexity, the mass matrix inverse is computed once with $O(n^2)$ complexity (e.g., via [34]), and a dense matrix-matrix multiply in (12) finally sets the complexity at $O(n^3)$. Since matrix multiplications are optimized on modern hardware, this approach [25] can be faster than the lower-order one. We next extend the relation (12) to the second-order case.

C. Second-Order Partials of Rigid-Body Dynamics

As the main technical contribution of the paper, this section presents an efficient way to include the second-order dynamics partials in DDP by employing reverse-mode AD and the relationship between first-order sensitivities (12). Figure 3 is diagrammed as a companion road-map to the following technical development. The derivation makes use of the identity

$$\frac{\partial^2 ID}{\partial q_i \partial \ddot{\mathbf{q}}} = \frac{\partial}{\partial q_i} \mathbf{M}(\mathbf{q}) = \frac{\partial \mathbf{M}}{\partial q_i}$$
(14)

and the fact that for a fixed \mathbf{q} , $\dot{\mathbf{q}}$, and $\boldsymbol{\tau}$, $\mathrm{ID}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is implicitly dependent on FD through composition via $\ddot{\mathbf{q}} = \mathrm{FD}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau})$. Therefore, the partials of ID include the partials of FD through chain rule. We start by using reconsidering the

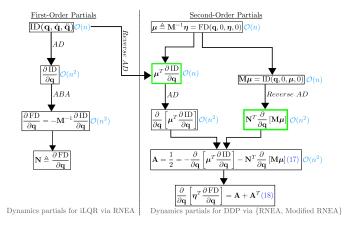


Fig. 3. Computational approach for the partials needed in *iLQR via RNEA*, *DDP via RNEA*, and *DDP via Modified RNEA*. For *DDP via Modified RNEA*, the highlighted blocks (in green) can be accelerated using the Modified RNEA (see Algo. 1)

second order partial in (10) via the use of (12):

$$\boldsymbol{\eta}^{T} \frac{\partial}{\partial q_{j}} \left[\frac{\partial \operatorname{FD}}{\partial q_{i}} \right] = -\boldsymbol{\eta}^{T} \frac{\partial}{\partial q_{j}} \left[\mathbf{M}^{-1} \frac{\partial \operatorname{ID}}{\partial q_{i}} \bigg|_{\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}} = \operatorname{FD}(\mathbf{q}, \dot{\mathbf{q}}, \tau)} \right]$$

$$= \boldsymbol{\eta}^{T} \mathbf{M}^{-1} \left[\frac{\partial \mathbf{M}}{\partial q_{j}} \mathbf{M}^{-1} \frac{\partial \operatorname{ID}}{\partial q_{i}} - \frac{\partial^{2} \operatorname{ID}}{\partial q_{i} \partial q_{j}} - \frac{\partial^{2} \operatorname{ID}}{\partial q_{i} \partial \ddot{\mathbf{q}}} \frac{\partial \operatorname{FD}}{\partial q_{j}} \right]$$
(15)

where we used that $\partial \mathbf{M}^{-1}/\partial q_j = -\mathbf{M}^{-1}(\partial \mathbf{M}/\partial q_j)\mathbf{M}^{-1}$, and the last term in the overall result is from the chain rule for the second-order partials of ID. The identity (14) allows for the rewrite of (15) such that

$$\boldsymbol{\eta}^{T} \frac{\partial}{\partial q_{j}} \left[\frac{\partial FD}{\partial q_{j}} \right]$$

$$= \boldsymbol{\eta}^{T} \mathbf{M}^{-1} \left[\frac{\partial \mathbf{M}}{\partial q_{j}} \mathbf{M}^{-1} \frac{\partial ID}{\partial q_{i}} - \frac{\partial^{2} ID}{\partial q_{i} \partial q_{j}} - \frac{\partial \mathbf{M}}{\partial q_{i}} \frac{\partial FD}{\partial q_{j}} \right].$$
(16)

The matrix-vector product $\mathbf{M}^{-1}\boldsymbol{\eta}$ can be computed efficiently using the ABA algorithm by ignoring gravity and the Coriolis term and using $\boldsymbol{\eta}$ as an input in place of $\boldsymbol{\tau}$. Once \mathbf{q} and $\boldsymbol{\eta}$ are given, we fix the value of $\boldsymbol{\mu} \triangleq \mathbf{M}^{-1}\boldsymbol{\eta} = \mathrm{FD}(\mathbf{q},0,\boldsymbol{\eta},0)$ (i.e., once fixed it no longer depends on \mathbf{q}).

We also fix each $\nu_i \triangleq \frac{\partial \text{FD}}{\partial q_i}$, and re-use this information in (16) as follows

$$\begin{split} \boldsymbol{\eta}^T \frac{\partial}{\partial q_j} \left[\frac{\partial \operatorname{FD}}{\partial q_j} \right] \\ &= -\boldsymbol{\mu}^T \frac{\partial}{\partial q_i} \left[\mathbf{M} \boldsymbol{\nu}_i \right] - \frac{\partial}{\partial q_i} \left[\boldsymbol{\mu}^T \frac{\partial \operatorname{ID}}{\partial q_i} \right] - \boldsymbol{\mu}^T \frac{\partial}{\partial q_i} \left[\mathbf{M} \boldsymbol{\nu}_j \right]. \end{split}$$

Note that the key reorganization $\frac{\partial}{\partial q_j} [\boldsymbol{\mu}^T \frac{\partial \operatorname{ID}}{\partial q_i}] = \boldsymbol{\mu}^T \frac{\partial^2 \operatorname{ID}}{\partial q_i \partial q_j}$ is possible since $\boldsymbol{\mu}$ is treated as constant after its initial evaluation. The resulting equation has a unique structure in that the terms $\frac{\partial}{\partial q_i} [\mathbf{M} \boldsymbol{\nu}_j]$ and $\frac{\partial}{\partial q_j} [\mathbf{M} \boldsymbol{\nu}_i]$ are related by index permutation. This observation allows us to define $\mathbf{N} \triangleq \frac{\partial \operatorname{FD}}{\partial \mathbf{q}}$ such that

$$oldsymbol{\mu}^T rac{\partial}{\partial q_j} \left[\mathbf{M} oldsymbol{
u}_i
ight] = \left[\mathbf{N}^T rac{\partial}{\partial \mathbf{q}} \left[\mathbf{M} oldsymbol{\mu}
ight]
ight]_{ij} \,.$$

As a result, the following definition

$$\mathbf{A} := -\frac{1}{2} \frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\mu}^T \frac{\partial \mathbf{ID}}{\partial \mathbf{q}} \right] - \mathbf{N}^T \frac{\partial}{\partial \mathbf{q}} \left[\mathbf{M} \boldsymbol{\mu} \right]$$
(17)

provides

$$\frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\eta}^T \frac{\partial FD}{\partial \mathbf{q}} \right] = \mathbf{A} + \mathbf{A}^T.$$
 (18)

Both terms in **A** can be computed in $\mathcal{O}(n^2)$ as follows. For the $\frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\mu}^T \frac{\partial \, \mathrm{ID}}{\partial \mathbf{q}} \right]$ term, $\boldsymbol{\mu}^T \frac{\partial \, \mathrm{ID}}{\partial \mathbf{q}}$ can first be computed using reverse-mode AD, then AD can be used again for the derivative of that result (Fig. 3). For the $\mathbf{N}^T \frac{\partial}{\partial \mathbf{q}} \left[\mathbf{M} \boldsymbol{\mu} \right]$ term, RNEA can be used to evaluate $\mathbf{M} \boldsymbol{\mu} = \mathrm{ID}(\mathbf{q}, 0, \boldsymbol{\mu}, 0)$ by ignoring gravity and the Coriolis terms. This calculation is an $\mathcal{O}(n)$ operation. Considering the n columns of \mathbf{N} to contract on the left of $\mathbf{N}^T \frac{\partial}{\partial \mathbf{q}} \left[\mathbf{M} \boldsymbol{\mu} \right]$ provides an $\mathcal{O}(n^2)$ method via reverse-mode AD. The first-order partials in \mathbf{N} have to be computed for any iLQR/DDP method, and are thus not an additional cost. Even though the formulas (17), (18) result in the same complexity as applying reverse-mode AD to ABA, the new approach is faster since RNEA is simpler than ABA.

Formulas for the other second-order partials are as follows. Using the same approach as before, we can show that

$$\frac{\partial}{\partial \dot{\mathbf{q}}} \left[\boldsymbol{\eta}^T \frac{\partial \operatorname{FD}}{\partial \dot{\mathbf{q}}} \right] = -\frac{\partial}{\partial \dot{\mathbf{q}}} \left[\boldsymbol{\mu}^T \frac{\partial \operatorname{ID}}{\partial \dot{\mathbf{q}}} \right] \; .$$

For the mixed partials of \mathbf{q} and $\dot{\mathbf{q}}$, we can show that

$$\frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\eta}^T \frac{\partial \operatorname{FD}}{\partial \dot{\mathbf{q}}} \right] = - \boldsymbol{\Psi}^T \frac{\partial}{\partial \mathbf{q}} \left[\mathbf{M}(\mathbf{q}) \boldsymbol{\mu} \right] - \frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\mu}^T \frac{\partial \operatorname{ID}}{\partial \dot{\mathbf{q}}} \right] \,,$$

where $\Psi=\frac{\partial\,\mathrm{FD}}{\partial\dot{\mathbf{q}}}.$ Finally, for the mixed partials of \mathbf{q} and $\boldsymbol{ au},$ the resulting relationship is

$$\frac{\partial}{\partial \mathbf{q}} \left[\boldsymbol{\eta}^T \frac{\partial \operatorname{FD}}{\partial \boldsymbol{\tau}} \right] = - \boldsymbol{\Xi}^T \frac{\partial}{\partial \mathbf{q}} \left[\mathbf{M}(\mathbf{q}) \boldsymbol{\mu} \right],$$

where $\Xi=\frac{\partial\,\mathrm{FD}}{\partial au}$. Using reverse-mode tools, all these partials can be computed with $\mathcal{O}(n^2)$ complexity. We denote DDP algorithms that use this approach as *DDP via RNEA*.

Across these derivations, the term $\mu^T \frac{\partial \operatorname{ID}}{\partial \mathbf{q}}$ is common. To evaluate $\mu^T \frac{\partial \operatorname{ID}}{\partial \mathbf{q}}$ we can either (1) use reverse AD with the RNEA to calculate result or (2) provide a method to compute the term $\mu^T \operatorname{ID}$, and then calculate its gradient with AD. While the two approaches are similar, we present a refactoring of the RNEA to reduce computation requirements for $\mu^T \operatorname{ID}$ before applying AD.

D. Modified RNEA Algorithm

Here, we follow spatial vector algebra, notation, and body numbering conventions in [28]. Consider a poly-articulated tree-structured system of N_B rigid bodies. We denote p(i) as the parent body of body i, and use the notation $j \succeq i$ to indicate when body j is after body i in the kinematic tree. Sums over pairs of related bodies can be carried out in either of the following ways:

$$\sum_{i=1}^{N_B} \sum_{j|j \succ i} = \sum_{j=1}^{N_B} \sum_{i|j \prec j} . \tag{19}$$

The modified RNEA output μ^T ID satisfies $\mu^T \tau = \sum_{i=1}^{N} \mu_i^T \tau_i$. The torque τ_i at joint i is given as

$$oldsymbol{ au}_i = \mathbf{S}_i^T \sum_{j|j\succeq i} {}^j \mathbf{X}_i^T oldsymbol{\mathcal{F}}_j \,,$$

where \mathbf{S}_i gives joint *i*'s free-modes, $\mathcal{F}_j = \mathbf{I}_j \mathbf{a}_j + (\mathbf{v}_j \times^*) \mathbf{I}_j \mathbf{v}_j$ the inertial force of body j, \mathbf{v}_j its spatial velocity, \mathbf{a}_j its spatial acceleration, and \times^* a cross product for spatial vectors [28]. The term $\boldsymbol{\mu}^T \boldsymbol{\tau}$ then satisfies

$$oldsymbol{\mu}^Toldsymbol{ au} = \sum_{i=1}^{N_B} \sum_{j|j \succ i} oldsymbol{\mu}_i^T \mathbf{S}_i^{Tj} \mathbf{X}_i^T oldsymbol{\mathcal{F}}_j.$$

The summation above is then refactored using (19) as

$$\boldsymbol{\mu}^T \operatorname{ID} = \sum_{j=1}^{N_B} \left[\sum_{i|i \leq j} {}^{j} \mathbf{X}_i \mathbf{S}_i \boldsymbol{\mu}_i \right]^T \boldsymbol{\mathcal{F}}_j.$$
 (20)

This refactoring leads to the modified RNEA algorithm (see Algo. 1). The result is that μ^T ID can be computed in a single pass rather than two passes as in RNEA. The algorithm is still an $\mathcal{O}(n)$ algorithm but leads to a simpler computation graph for reverse-mode AD. Whenever this method is used in DDP, it is referred to as *DDP via Modified RNEA*. The computation workflow in this case is given as in Fig. 3, where the modified RNEA is used to accelerate the blocks highlighted in green.

E. Summary

Before proceeding to the presentation of comparative results, we briefly review the methods introduced for incorporating partials into iLQR and DDP. When only using first-order dynamics partials, we have *iLQR via ABA* and *iLQR via RNEA*. For full second-order methods, we have *DDP via ABA*, *DDP via RNEA*, and *DDP via Modified RNEA* methods, all avoiding explicitly computing the second-order dynamics derivative tensor. Finally the conventional DDP method *DDP via Tensor Contraction* involves computing the second-order derivative tensor. The computation approaches of these methods have been diagrammed in Fig. 2 and Fig. 3.

IV. RESULTS

To test the performance and scalability of our proposed methods, we evaluate their application to solve an OCP for

Algorithm 1 Modified RNEA Algorithm

Require: $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \boldsymbol{\mu}, model$

1:
$$\mathbf{v}_0 = 0, \mathbf{a}_0 = -\mathbf{a}_g, \mathbf{w}_0 = 0, s = 0$$

2: for i = 1 to N do

3:
$$\mathbf{v}_i = {}^i\mathbf{X}_{p(i)}\,\mathbf{v}_{p(i)} + \mathbf{S}_i\,\dot{\mathbf{q}}_i$$

4:
$$\mathbf{w}_i = {}^i\mathbf{X}_{p(i)}\,\mathbf{w}_{p(i)} + \mathbf{S}_i\,\boldsymbol{\mu}_i$$

5:
$$\mathbf{a}_i = {}^i \mathbf{X}_{p(i)} \, \mathbf{a}_{p(i)} + (\mathbf{v}_i \times) \, \mathbf{S}_i \dot{\mathbf{q}}_i + \mathbf{S}_i \ddot{\mathbf{q}}_i$$

6:
$$s + = \mathbf{w}_i^T (\mathbf{I}_i \mathbf{a}_i + (\mathbf{v}_i \times^*) \mathbf{I}_i \mathbf{v}_i)$$

7: end for

8: **return**
$$s = \boldsymbol{\mu}^T \boldsymbol{\tau}$$

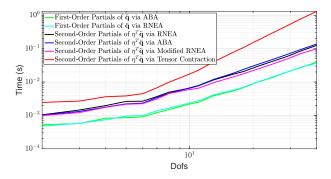


Fig. 4. The computation time for evaluating the partials of the dynamics in MATLAB with CasADi.

an underactuated n-link pendubot. The trajectory optimization problem here was a swing-up problem and this goal was encoded via design of running and terminal costs. These costs were similar for all proposed methods regardless of the number of links in the model. As the number of links is increased, their nonlinear couplings on one another present additional challenge for solving the OCP. To further make the control problem challenging, the final link in the system was left un-actuated. For all the proposed methods, the same convergence criteria was used with convergence indicated by a negligible ($< 10^{-9}$) reduction in the cost function between iterations. We first compare the computation time of the dynamics partials (Section IV-A) using the methods described previously and then compare the addition of those partials within DDP/iLQR optimization frameworks as appropriate (see Section IV-B). This work was implemented in MATLAB¹ alongside the CasADi [31] Toolkit which allows for rapid and efficient testing of AD approaches. Since the partials are evaluated in the CasADi virtual machine through MATLAB, merits of the methods should be assessed via comparison between them, while future work will study improving absolute timing numbers via C/C++ implementation.

A. Dynamics Partials

Within DDP/iLQR, the computation of the partials of the dynamics is the most computationally expensive part of the optimization process. Figure 4 compares the computation time of the different methods for evaluating these partials. As shown, evaluation of the second-order partials by tensor contraction takes the longest time whereas all other second-order partials have the same computational complexity as the first-order dynamic partials (as indicated by the slope on the log-log plot). The most competitive second-order approach requires approximately only 2.5 times more computation time than first-order partials. Second-order partials via RNEA/modified RNEA were faster than second-order partials via ABA since RNEA is simpler than ABA, with the modified RNEA outperforming RNEA. These results indicate that when compared to conventional second-order tensorial dynamic partials, the proposed methods have the potential to reduce the computational overhead of DDP to be competitive with iLQR.

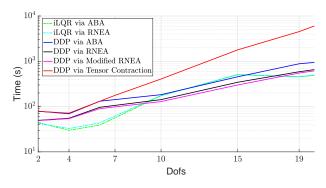


Fig. 5. Computation time for solving a swing-up OCP for an n-link pendubot in MATLAB.

B. Trajectory Optimization: DDP/iLQR Framework

We then include the dynamics partials in DDP/iLQR as appropriate and evaluate the OCP for the pendubot. Note that for an OCP with a horizon of N timesteps, each iteration of iLQR/DDP must evaluate the derivatives N times, motivating the need for their rapid evaluation. Figure 5 illustrates the time required to solve the OCP to convergence with DDP and iLQR variants. Tensor-free DDP variants had evaluation times comparable to the iLQR ones, while DDP via Tensor Contraction takes longer to optimize due to its higher cost of evaluating derivatives. The comparative performance of the fastest iLQR and DDP variants depends on the problem instance. Trajectory updates performed for either algorithm depend on the non-linearity of the system considered, and on the initial guess, which prevents uniformly recommending iLQR or DDP over the other.

We also evaluate the mean time to compute the DDP/iLQR variants for a 7-link pendubot model. We use a dissipative controller as the initial control trajectory and randomize the initial state vector around the downward configuration of the pendubot. This is a difficult problem, as it forces the control to pump energy into the system in order to drive it to the upright configuration. Figure 6 illustrates the time to solve the OCP with DDP/iLQR variants. As shown, DDP via Tensor Contraction took the longest time to converge, whereas the tensor-free DDP strategies took more time compared to iLQR in this case. A portion of the additional time for DDP is attributed to repeats of the backward sweep due to the regularization needed for DDP in this case. While this motivates focus on these aspects in future work, it is worth noting that the running cost in this case was convex, and this provides benefit to iLQR in terms of avoiding regularization.

Finally, we optimize a trajectory for a 7-link KUKA LBR manipulator (available in MATLAB's Robotics Toolbox) using DDP/iLQR methods. Figure 7 illustrates suboptimality vs. iterations for iLQR and DDP when applied to the manipulator. The suboptimality measures the difference between the current cost function value and its value at the end of the iterations. As illustrated, the DDP variants featured quadratic convergence whereas the iLQR variants featured super-linear convergence. This figure also illustrates that the DDP variants took fewer iterations compared to iLQR counterparts.

Since the solution of an OCP is dependent on the initial

¹Open-source code: https://tinyurl.com/468ynkuu

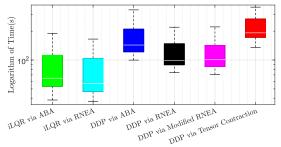


Fig. 6. Time to converge for a 7-link model with a random sampling of the initial conditions around the downward configuration of the pendubot.

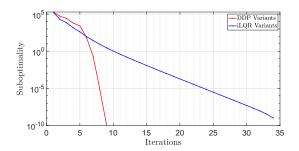


Fig. 7. Cost convergence over iterations for a 7-link KUKA LBR model.

conditions, we randomize the initial controller using Ornstein-Uhlenbeck process noise for the 7-link KUKA LBR manipulator and solved the OCP problem using either iLQR/DDP in 40 separate instances. The first pane of Fig. 8 illustrates the progression of that manipulator from an initial configuration to the balanced upright configuration along an optimal trajectory. Subplot (a) of Fig. 8 illustrates the empirical probability density function (pdf) of the number of iterations over those optimizations. Subplot (b) illustrates the pdf of the log of the final cost of each optimal solution. As illustrated in subplot (a), in 95% of the simulations, iLQR had a higher number of iterations. On average, iLQR had three times as many iterations as DDP. From subplot (b), we note that most of the simulations regardless of DDP/iLQR converged to similar solution; in fact, DDP converged to a different solution than iLQR in only three instances. Figure 8 illustrates that the inclusion of second-order information in DDP will result in similar converged solution and in fewer iterations. This result is powerful in that the addition of second-order information results in algorithm that converges in less iterations and to the same minima.

C. Discussion

As compared to conventional approaches to DDP that explicitly compute second-order derivatives of the dynamics, the proposed DDP variants presented herein show a marked improvement in their computational complexity and computation time. This is especially important as second-order information retains better local fidelity to the original model and therefore a second-order approximation better captures the nonlinear effects of the system. This feature is expected to be important for complex systems such as quadrupeds whose coupled nonlinear dynamics might not be accurately captured by a first-order approximation. This second-order information

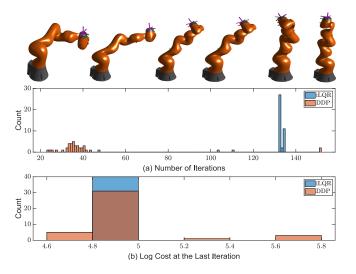


Fig. 8. Optimization results for a 7-link KUKA LBR model with randomized initial control inputs from an Ornstein-Uhlenbeck noise process. The LBR was simulated using MATLAB's Robotics toolbox.

could be of value in critical circumstances, for example, the increased fidelity could help a quadruped prevent falls or handle disturbances when traversing unstructured terrain.

We noted previously that DDP includes a regularization scheme to ensure that $Q_{\mathbf{u}\mathbf{u}}$ remains positive-definite during the backward pass. This regularization incurs additional computational cost as compared to iLQR methods, and this additional cost cannot be anticipated before running the optimization. In Fig. 6, the proposed DDP variants needed more computation time than the iLQR variants, though we expected those evaluation times to be more similar based on other tests (Fig. 1). We attribute a portion of the computational cost to repeats of the backward sweep when regularization fails, as most of the computation in DDP is spent evaluating the backward sweep.

Lastly, our results showed that iLQR typically has more iterations but overall had comparable computation time as proposed DDP variants. Therefore, the benefits of DDP can be included in trajectory optimization without the previously significant sacrifice in evaluation time, and can be done in fewer iterations. This may lead to more robust model-predictive control wherein warm staring online may keep DDP iterations within the quadratic convergence well.

V. CONCLUSION AND FUTURE DIRECTIONS

This work has made use of reverse-mode AD tools to quickly evaluate different approaches for obtaining the derivatives needed by DDP. We also extended the relationship between the first-order sensitivities of ID and FD [25], [26] for rigid-body systems to the second-order case. The combination of this new approach and AD tools allows for the evaluation of the needed derivatives in DDP with the same complexity as iLQR. Lastly, we introduce a restructuring of RNEA to derive a modified RNEA that returns μ^T ID in $\mathcal{O}(n)$ complexity, and enables the fastest DDP algorithm.

While AD tools are convenient, they are general purpose, and thus may not be optimal. Alternative analytical methods for taking derivatives of rigid-body dynamics can accumulate

the derivatives recursively, as in [25], [26], [32], [33]. Recently, we extended the modified RNEA algorithm with an analytical accumulation of its first-order partials in a reverse-mode fashion, and further evaluation of this result is of immediate interest. Moreover, we aim to extend this work to address rigid-body dynamics with contacts [15], [35] by using similar approaches as in this paper. This generalization would then allow for use with hybrid dynamic systems that arise in legged locomotion problems.

There are many other opportunities that this work motivates as next steps. While we noted that the presented work is general for any explicit integration scheme, we also see opportunity to extend this work for implicit integration and implicit DDP [36]. Further, while the DDP used here was a singleshooting solver, our contributions could be used in multishooting DDP and other numerical optimal control solvers. Finally, our work may find applicability when working to control soft robots. Many multi-segment soft-body robots can be modeled assuming piece-wise constant curvature (PCC) [37], approximating PCC with a high-DoF augmented rigid model [37], or considering discrete Cosserat models [38]. The ideas herein may find applicability for these models due to their dynamics equations taking a similar structural form (e.g., (1)) as rigid-body systems. Augmented rigid models are most directly applicable in 2D, but generalizations of the RNEA and ABA [38] for the 3D continuum case also present interesting future avenues to broaden the application of this work.

REFERENCES

- [1] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer Handbook of Robotics*, 2016, pp. 1203–1234.
- [2] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," in *Nonlinear model* predictive control. Springer, 2009, pp. 391–417.
- [3] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.
- [4] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast motions* in biomechanics and robotics. Springer, 2006, pp. 65–93.
- [5] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," SIAM Review, vol. 59, no. 4, pp. 849–904, 2017.
- [6] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *IEEE-RAS Interna*tional Conference on Humanoid Robots, 2014, pp. 295–302.
- [7] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the HRP-2 humanoid," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2015, pp. 3346–3351.
- [8] M. Neunert, M. Stäuble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter, and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, 2018.
- [9] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback MPC for torque-controlled legged robots," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2019, pp. 4730–4737.
- [10] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2012, pp. 4906–4913.
- [11] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal* of Control, vol. 3, no. 1, pp. 85–95, 1966.
- [12] G. Lantoine and R. P. Russell, "A hybrid differential dynamic programming algorithm for constrained optimal control problems. Part 1: Theory," *Journal of Optimization Theory and Applications*, vol. 154, no. 2, pp. 382–417, 2012.

- [13] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A fast solver for constrained trajectory optimization," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2019, pp. 7674–7679.
- [14] A. Pavlov, I. Shames, and C. Manzie, "Interior point differential dynamic programming," *IEEE Transactions on Control Systems Technology*, 2021.
- [15] H. Li and P. M. Wensing, "Hybrid systems differential dynamic programming for whole-body motion planning of legged robots," *IEEE Robotics* and Automation Letters, vol. 5, no. 4, pp. 5448–5455, 2020.
- [16] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *IEEE Int. Conf. on Robotics and Automation*, 2014, pp. 1168–1175.
- [17] J. Marti-Saumell, J. Sola, C. Mastalli, and A. Santamaria-Navarro, "Squash-box feasibility driven differential dynamic programming," in IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2020.
- [18] E. Pellegrini and R. P. Russell, "A multiple-shooting differential dynamic programming algorithm," in AAS/AIAA Space Flight Mechanics Meeting, vol. 2, 2017.
- [19] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative Gauss-Newton shooting methods for nonlinear optimal control," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2018, pp. 1–9.
- [20] B. Plancher and S. Kuindersma, "A performance analysis of parallel differential dynamic programming on a GPU," in Workshop on the Algorithmic Foundations of Robotics, 2018, pp. 656–672.
- [21] F. Farshidian, E. Jelavic, A. Satapathy, M. Giftthaler, and J. Buchli, "Real-time motion planning of legged robots: A model predictive control approach," in *IEEE-RAS Int. Conf. on Humanoid Robotics*, 2017, pp. 577–584.
- [22] H. Li, R. J. Frei, and P. M. Wensing, "Model hierarchy predictive control of robotic systems," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3373–3380, 2021.
- [23] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO*, 2004, pp. 222– 229
- [24] A. Grievank, "Principles and techniques of algorithmic differentiation: Evaluating derivatives," SIAM, Philadelphia, 2000.
- [25] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Robotics: Science and Systems*, 2018.
- [26] A. Jain and G. Rodriguez, "Linearization of manipulator dynamics using spatial operators," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 1, pp. 239–248, 1993.
- [27] D. E. Orin, R. McGhee, M. Vukobratović, and G. Hartoch, "Kinematic and kinetic analysis of open-chain linkages utilizing Newton-Euler methods," *Math Biosciences*, vol. 43, no. 1-2, pp. 107–130, 1979.
- [28] R. Featherstone, Rigid body dynamics algorithms. Springer, 2014.
- [29] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," 1982.
- [30] L.-Z. Liao and C. A. Shoemaker, "Convergence in unconstrained discrete-time differential dynamic programming," *IEEE Transactions on Automatic Control*, vol. 36, no. 6, pp. 692–706, 1991.
- [31] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.
- [32] S.-H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, "Newton-type algorithms for dynamics-based robot movement optimization," *IEEE Transactions on robotics*, vol. 21, no. 4, pp. 657–667, 2005.
- [33] G. A. Sohl and J. E. Bobrow, "A recursive multibody dynamics and sensitivity algorithm for branched kinematic chains," J. Dyn. Sys., Meas., Control, vol. 123, no. 3, pp. 391–399, 2001.
- [34] J. Carpentier, "Analytical Inverse of the Joint Space Inertia Matrix," 2018. [Online]. Available: https://hal.laas.fr/hal-01790934
- [35] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, "Differential dynamic programming for multi-phase rigid contact dynamics," in *IEEE-RAS Int. Conf. on Humanoid Robots*, 2018, pp. 1–9.
- [36] I. Chatzinikolaidis and Z. Li, "Trajectory optimization of contactrich motions using implicit differential dynamic programming," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2626–2633, 2021.
- [37] R. K. Katzschmann, C. Della Santina, Y. Toshimitsu, A. Bicchi, and D. Rus, "Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model," in *IEEE Int. Conf. on Soft Robotics*, 2019, pp. 454–461.
- [38] F. Renda, F. Boyer, J. Dias, and L. Seneviratne, "Discrete cosserat approach for multisection soft manipulator dynamics," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1518–1533, 2018.