Accelerating Hybrid Systems Differential Dynamic Programming

John Nganga[1] and Patrick M. Wensing[1]

***Abstract*— This abstract presents methods that reduce the computational demand of including second-order dynamics sensitivity information into optimization algorithms for robots in contact with the environment. The work extends a previous study on unconstrained whole-body motion planning. A full second-order Differential Dynamic Programming (DDP) algorithm is presented where all the necessary dynamics partial derivatives are computed with lower computational complexity than conventional DDP methods. Compared to using a first-order approximation, the second-order partials more accurately represent the dynamics locally, but since they are tensorial and expensive to compute, they are often ignored. This work illustrates how to avoid the need for computing the derivative tensor by instead leveraging reverse-mode accumulation of derivatives. We also exploit the structure of the contact-constrained dynamics to further accelerate these computations. The relative performance of the proposed approach is benchmarked on a simulated model of the MIT Mini Cheetah executing a bounding gait.**

***Paper Type* – Original Work**

I. Introduction

In comparison to their biological counterparts, legged robots are yet to be as mobile and dexterous. Animals devise and execute fast and fluid movements on the fly despite their numerous degrees of freedom (DoF). This capability is most beneficial in new, unknown, and uneven terrain – where the fast and fluid movements are tailored to the novel context. The scientific gap for legged robots to achieve this goal is, in part, due to challenges from the robot's highly non-linear dynamics and the curse of dimensionality from many DoFs. A common approach in the literature is to cast the robotic locomotion problem as a trajectory optimization problem, solved in a receding horizon fashion. The computation of an optimal control tape then, ideally, allows for the robot to achieve a desired motion while maintaining energetic efficiency and respecting constraints.

Whole-body motion planning considers a finite-horizon trajectory optimization problem posed over the full dynamics of the robot, enabling the synthesis of complex desired motions. While the computational burden of considering the full model is significant, continuous gains in computational power have enabled a steady transition to whole-body trajectory optimization. For example, the authors in [1] use a whole-body trajectory optimizer based on Differential Dynamic programming (DDP) to control a 22-DoF humanoid robot. Rather than optimizing all control variables at once, DDP exploits the sparsity of the OCP and solves a sequence of smaller optimization problem at each point along the horizon. DDP also outputs a locally optimal feedback policy, which can be used to handle disturbances [2]. As originally described [3], DDP does not handle constraints. However, several recent modifications to DDP that address control limits [4] and general state/control constraints [5], [6] have been proposed. Further, the work of [5], [7]–[9] have extended the algorithm from smooth dynamical systems to hybrid systems that have sequences of smooth modes with reset maps determining transitions between them. This work considers the extension of DDP to systems undergoing rigid contacts with the environment in the same fashion as [7].

In its originally proposed form, DDP considers a second-order approximation of the dynamics model. However, in practice (e.g., [1], [8], [10]) many researchers have opted to use a first-order dynamics approximation due to its faster evaluation time. 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. Our previous work [11] avoids the evaluation of the dynamics derivative tensor for the DDP algorithm for smooth dynamical systems. In this work, we extend that study [11] to systems undergoing rigid contacts with the environment.

II. Background: Trajectory Optimization

Consider a rigid-body model in contact with the environment at some known set of points. Using generalized coordinates $\mathbf{q}$, the dynamics of the model are given by:

$$\underbrace{\begin{bmatrix} \mathbf{H} & -\mathbf{J}_c^\top \\ -\mathbf{J}_c & 0 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{bmatrix}}_{\boldsymbol{\nu}} = \underbrace{\begin{bmatrix} \mathbf{S}^\top \boldsymbol{\tau} + \mathbf{h} \\ \dot{\mathbf{J}}_c \dot{\mathbf{q}} \end{bmatrix}}_{\boldsymbol{\Psi}} \tag{1}$$

where $\mathbf{H}$ and $\mathbf{S}$ denotes the inertia matrix and selector matrix respectively. The term $\mathbf{h}$ is the joint-space bias vector and it groups the Coriolis, centrifugal, and gravitational components. $\mathbf{J}_c$ and $\boldsymbol{\lambda}_c$ represent the contact Jacobian and the corresponding contact forces. Overall, (1) is known as the contact-constrained dynamics with the first row of (1) representing inverse dynamics in the conventional sense and the second row representing contact acceleration constraint. The solution for (1) can be given as

$$\begin{bmatrix} \mathfrak{F} \\ \mathfrak{g} \end{bmatrix} \triangleq \boldsymbol{\nu} = \mathbf{K}^{-1} \boldsymbol{\Psi} \, . \tag{2}$$

where $\mathfrak{F}$ represents the realized continuous joint acceleration and $\mathfrak{g}$ is the realized contact force functions.

A. Hybrid Systems DDP Algorithm

Consider a model with state $\mathbf{x} = [\mathbf{q}^\top \ \dot{\mathbf{q}}^\top]^\top$, control input $\boldsymbol{\tau}$, and ground reaction forces $\boldsymbol{\lambda}$. The continuous state and

This work was supported in part by NASA Award 80NSSC21K1281.

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

control trajectories can be discretized such that

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \triangleq \mathbf{x}_k + h \begin{bmatrix} \dot{\mathbf{q}} \\ \mathfrak{F}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau}) \end{bmatrix} \tag{3}$$

$$\boldsymbol{\lambda}_k = \mathfrak{g}(\mathbf{x}_k, \mathbf{u}_k)\,, \tag{4}$$

where h is the integration stepsize.

DDP is used to solve an OCP with m modes such that the cost function is a summation across each mode given as:

$$\mathbf{J}(\mathbf{U}) = \sum_{i=1}^{m} \left[\Phi_i(\mathbf{x}_N) + \sum_{k=0}^{N_{i-1}} \ell_i(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_k) \right]. \tag{5}$$

where ℓ_i is the running cost and Φ_i is the terminal cost incurred at the end of a horizon N. To obtain an optimal trajectory, (5) is minimized across modes as

$$\min_{\mathbf{U}} \ \mathbf{J}(\mathbf{U}) \tag{6a}$$

$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{f}_i(\mathbf{x}_k, \mathbf{u}_k) \tag{6b}$$

$$\boldsymbol{\lambda}_i = \mathfrak{g}_i(\mathbf{x}_k, \mathbf{u}_k) \tag{6c}$$

$$\text{Constraints} \tag{6d}$$

where $\mathbf{f}_i$ and $\mathfrak{g}_i$ denote the dynamics and ground reaction force functions for each mode. Equation (6d) groups several constraints such as switching constraints, torque limits, and contact force constraints.

DDP solves (6) by optimizing a control policy at each time point, and working backwards in time using Bellman's principle [12]. This process recursively provides a value function approximation as:

$$V_k^\star(\mathbf{x}_k) = \min_{\mathbf{u}_k} \Bigg[\underbrace{\ell_k(\mathbf{x}_k, \mathbf{u}_k, \mathfrak{g}(\mathbf{x}_k, \mathbf{u}_k)) + V_{k+1}^\star(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k))}_{Q_k(\mathbf{u}_k, \mathbf{x}_k)} \Bigg]$$
$$\text{where} \quad V_N^\star(\mathbf{x}_N) = \Phi(\mathbf{x}_N)\,, \tag{7}$$

and where the function, $Q_k(\mathbf{x}, \mathbf{u})$, captures the cost to go when starting in state $\mathbf{x}_k$ at time k, taking action $\mathbf{u}_k$, and then acting optimally thereafter. In solving (7), we consider the differential change to Q_k around a nominal state-control pair $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{u}}_k$ such that $\delta Q_k(\delta\mathbf{x}_k, \delta\mathbf{u}_k) = Q_k(\bar{\mathbf{x}}_k + \delta\mathbf{x}_k, \bar{\mathbf{u}}_k + \delta\mathbf{u}_k) - Q_k(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k)$. The second-order approximation of the Q-function leads to its first- and second-order partials as

$$\begin{aligned}
Q_{\mathbf{x}} &= \ell_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^\top V_{\mathbf{x}}' + \mathfrak{g}_{\mathbf{x}}^\top \ell_{\boldsymbol{\lambda}} \\
Q_{\mathbf{u}} &= \ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^\top V_{\mathbf{x}}' + \mathfrak{g}_{\mathbf{u}}^\top \ell_{\boldsymbol{\lambda}} \\
Q_{\mathbf{xx}} &= \ell_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^\top V_{\mathbf{xx}}' \mathbf{f}_{\mathbf{x}} + \mathfrak{g}_{\mathbf{x}}^\top \ell_{\boldsymbol{\lambda}\boldsymbol{\lambda}} \mathfrak{g}_{\mathbf{x}} + \begin{bmatrix} V_{\mathbf{x}}' \\ \ell_{\boldsymbol{\lambda}}^\top \end{bmatrix}^\top \begin{bmatrix}\!\!\begin{bmatrix} \mathbf{f}_{\mathbf{xx}} \\ \mathfrak{g}_{\mathbf{xx}} \end{bmatrix}\!\!\end{bmatrix} \\
Q_{\mathbf{uu}} &= \ell_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^\top V_{\mathbf{xx}}' \mathbf{f}_{\mathbf{u}} + \mathfrak{g}_{\mathbf{u}}^\top \ell_{\boldsymbol{\lambda}\boldsymbol{\lambda}} \mathfrak{g}_{\mathbf{u}} + \begin{bmatrix} V_{\mathbf{x}}' \\ \ell_{\boldsymbol{\lambda}}^\top \end{bmatrix}^\top \begin{bmatrix}\!\!\begin{bmatrix} \mathbf{f}_{\mathbf{uu}} \\ \mathfrak{g}_{\mathbf{uu}} \end{bmatrix}\!\!\end{bmatrix} \\
Q_{\mathbf{ux}} &= \ell_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^\top V_{\mathbf{xx}}' \mathbf{f}_{\mathbf{x}} + \mathfrak{g}_{\mathbf{u}}^\top \ell_{\boldsymbol{\lambda}\boldsymbol{\lambda}} \mathfrak{g}_{\mathbf{x}} + \begin{bmatrix} V_{\mathbf{x}}' \\ \ell_{\boldsymbol{\lambda}}^\top \end{bmatrix}^\top \begin{bmatrix}\!\!\begin{bmatrix} \mathbf{f}_{\mathbf{ux}} \\ \mathfrak{g}_{\mathbf{ux}} \end{bmatrix}\!\!\end{bmatrix}
\end{aligned} \tag{8}$$

where $[\![\cdot]\!]$ denotes a tensor term, which is often ignored [1], [13], [14]. Further, $[\cdot]^\top [\![\cdot]\!]$ implies a tensor contraction with a vector. Minimizing (7) over $\delta\mathbf{u}_k$ attains the incremental control

$$\delta\mathbf{u}_k^\star = \underset{\delta\mathbf{u}_k}{\operatorname{argmin}} \ \delta Q_k(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \tag{9}$$

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 (9) resulting in a new state-control trajectory [3]. The sweeps are repeated to convergence.

B. Dynamics Function Realization

The inverse dynamics (ID) of a rigid-body system is given by the recursive Newton Euler Algorithm (RNEA) as:

$$\begin{aligned}
\boldsymbol{\tau} &= \text{RNEA}(\text{model}, \mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{a}_g) \\
&= \mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}})\,,
\end{aligned}$$

with $\mathbf{a}_g$ the gravitational vector. The RNEA [15], [16] can be evaluated with $\mathcal{O}(n)$ complexity where n is the number of DoFs in the model. The calculations in the RNEA include the velocities and accelerations of all bodies, including those in contact. These quantities will indeed prove important later on in this development.

In [11], a refactoring of the RNEA algorithm was introduced to present a modified RNEA (mRNEA) that outputs $\boldsymbol{\mu}^\top\boldsymbol{\tau}$ where $\boldsymbol{\mu}$ is any fixed vector. The resulting mRNEA is an $\mathcal{O}(n)$ one-pass algorithm. Toward extending that algorithm for use with contact-constrained dynamics, we first recognize that contact-constrained dynamics involve contact forces. As such, we extend [11, Algorithm 1] to include the effects of the contact forces. Second, we extend the method to add an output from the acceleration of contact bodies.

$$\begin{aligned}
\boldsymbol{\mu}^\top\boldsymbol{\tau} + \boldsymbol{\pi}^\top\mathbf{a}_c &= \text{mRNEAc}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{a}_g, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\pi}) \\
&= \boldsymbol{\mu}^\top \left[\mathbf{H}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{J}_c^\top\boldsymbol{\lambda}\right] \\
&\quad + \boldsymbol{\pi}^\top \left[\mathbf{J}_c\ddot{\mathbf{q}} + \dot{\mathbf{J}}_c\dot{\mathbf{q}}\right].
\end{aligned}$$

where, $\boldsymbol{\mu}$ and $\boldsymbol{\pi}$ are fixed vectors. The utility of the contact acceleration term will become clear in the subsequent section, however, its presence can be intuitively motivated by the prominence of the contact acceleration in (1). The resulting algorithm is called the modified RNEA for contacts (mRNEAc), and is presented in Algo. 1 using rigid-body dynamics notation from [16]. The mRNEAc is still an $\mathcal{O}(n)$ one-pass algorithm. We proceed to demonstrate the importance of this new algorithm for accelerating DDP.

III. Reducing Complexity in Hybrid Systems DDP

Within DDP, the computations of the dynamics partials represent the most computationally intensive operations. We present an approach for reducing the computational complexity of evaluating the tensorial portions of the Q-function partials in (8).

A. Conventional Partials

Within DDP, we need the first- and second-order partials of (2). Let $\mathbf{z}$ and $\mathbf{y}$ represent $\mathbf{q}, \dot{\mathbf{q}}$, or $\boldsymbol{\tau}$. As such, we can use

Algorithm 1 Modified RNEA Algorithm for Contacts

Require: model, $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{a}_g, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\pi}$

$\mathbf{v}_0 = 0, \mathbf{a}_0 = -\mathbf{a}_g, \mathbf{w}_0 = 0, s = 0$
for $i = 1$ to N **do**
  $\mathbf{v}_i = {}^i\mathbf{X}_{p(i)}\, \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i\, \dot{\mathbf{q}}_i$
  $\mathbf{w}_i = {}^i\mathbf{X}_{p(i)}\, \mathbf{w}_{p(i)} + \boldsymbol{\Phi}_i\, \boldsymbol{\mu}_i$
  $\mathbf{a}_i = {}^i\mathbf{X}_{p(i)}\, \mathbf{a}_{p(i)} + (\mathbf{v}_i \times) \boldsymbol{\Phi}_i \dot{\mathbf{q}}_i + \boldsymbol{\Phi}_i \ddot{\mathbf{q}}_i$
  $\mathbf{F}_i = \mathbf{I}_i \mathbf{a}_i + (\mathbf{v}_i \times^*) \mathbf{I}_i \mathbf{v}_i - \boldsymbol{\lambda}_i$
  $s \mathrel{+}= \mathbf{w}_i^\top \mathbf{F}_i + \boldsymbol{\pi}_i^\top \mathbf{a}_i$
end for
return $[s = \boldsymbol{\mu}^T \boldsymbol{\tau} + \boldsymbol{\pi}^\top \mathbf{a}_c]$

Automatic Differentiation (AD) tools once on $\boldsymbol{\nu} \triangleq \mathbf{K}^{-1}\boldsymbol{\Psi}$ to obtain $\frac{\partial \boldsymbol{\nu}}{\partial \mathbf{z}}$. For a fixed number of contacts, this operation can be implemented to scale with $\mathcal{O}(n^2)$ complexity. This complexity is the lowest possible, since the operation provides the partials of the n entries of $\boldsymbol{\nu}$ w.r.t. the n entries of the $\mathbf{z}$ vector.

For second-order partials in DDP, AD tools can be used twice on $\boldsymbol{\nu} \triangleq \mathbf{K}^{-1}\boldsymbol{\Psi}$ to attain $\left[\!\left[\frac{\partial^2 \boldsymbol{\nu}}{\partial \mathbf{z} \partial \mathbf{y}} \right]\!\right]$. In using AD tools twice, $\frac{\partial^2 \boldsymbol{\nu}}{\partial \mathbf{z} \partial \mathbf{y}}$ requires an $\mathcal{O}(n^3)$ operation that considers the partials of the n entries of $\boldsymbol{\nu}$ w.r.t. the n^2 entries of the $\mathbf{z}$ and $\mathbf{y}$ vectors. Within the DDP context, the second-order partials will be contracted with fixed vector $\boldsymbol{\gamma}^\top = [V_{\mathbf{x}}'^\top \ \ell_{\boldsymbol{\lambda}}^\top]^\top$ (see (8)) from the left. The resulting approach is denoted *Tensor DDP*. We aim to reduce the computation complexity for these second-order partials.

B. Proposed Approach: mRNEAc Partials

Consider the partials of $\mathbf{K}\boldsymbol{\nu} = \boldsymbol{\Psi}$. The first-order partials of $\mathbf{K}\boldsymbol{\nu} = \boldsymbol{\Psi}$ can be given as

$$\left[\!\left[\frac{\partial \mathbf{K}}{\partial \mathbf{z}} \right]\!\right] \boldsymbol{\nu} + \mathbf{K} \frac{\partial \boldsymbol{\nu}}{\partial \mathbf{z}} = \frac{\partial \boldsymbol{\Psi}}{\partial \mathbf{z}} \ ,$$

where we can then solve for $\frac{\partial \boldsymbol{\nu}}{\partial \mathbf{z}}$. The second-order partials can be given by

$$\begin{aligned} &\left[\!\left[\frac{\partial^2 \mathbf{K}}{\partial \mathbf{z} \partial \mathbf{y}} \right]\!\right] \boldsymbol{\nu} + \left[\!\left[\frac{\partial \mathbf{K}}{\partial \mathbf{z}} \right]\!\right] \frac{\partial \boldsymbol{\nu}}{\partial \mathbf{y}} + \\ &\left[\!\left[\frac{\partial \mathbf{K}}{\partial \mathbf{y}} \right]\!\right] \frac{\partial \boldsymbol{\nu}}{\partial \mathbf{z}} + \mathbf{K} \left[\!\left[\frac{\partial^2 \boldsymbol{\nu}}{\partial \mathbf{z} \partial \mathbf{y}} \right]\!\right] = \left[\!\left[\frac{\partial^2 \boldsymbol{\Psi}}{\partial \mathbf{z} \partial \mathbf{y}} \right]\!\right] . \end{aligned} \tag{10}$$

We consider the rearrangement of (10) and the contraction of the second-order partials with the fixed vector $\boldsymbol{\gamma}^\top$ as needed within DDP. We focus exclusively on the second-order partials w.r.t. $\mathbf{q}$, and rewrite (10) as:

$$\boldsymbol{\gamma}^\top \left[\!\left[\frac{\partial^2 \boldsymbol{\nu}}{\partial \mathbf{q} \partial \mathbf{q}} \right]\!\right] = \underbrace{\boldsymbol{\gamma}^\top \mathbf{K}^{-1}}_{\boldsymbol{\xi}^\top} \left[\!\left[\frac{\partial^2 \boldsymbol{\Psi}}{\partial \mathbf{q} \partial \mathbf{q}} - \frac{\partial^2 \mathbf{K}}{\partial \mathbf{q} \partial \mathbf{q}} \boldsymbol{\nu} - (\mathcal{A}^\top + \mathcal{A}) \right]\!\right] ,$$

$$\text{where } \mathcal{A} \triangleq \left[\!\left[\frac{\partial}{\partial \mathbf{q}} \left(\mathbf{K} \frac{\partial \boldsymbol{\nu}}{\partial \mathbf{q}} \right) \right]\!\right] \tag{11}$$

and where $\boldsymbol{\xi}^\top$ contracts the tensor terms to a matrix.

We consider (11), where rather than lumping several quantities into $\mathbf{K}$, $\boldsymbol{\nu}$, and $\boldsymbol{\Psi}$, we separate out the dynamics functions into their constituent terms and partition $\boldsymbol{\xi}^\top = [\boldsymbol{\xi}_{\boldsymbol{\tau}} \ \boldsymbol{\xi}_{\mathbf{a}}]^\top$. This enables us to rewrite (11) as:

$$\boldsymbol{\gamma}^\top \left[\!\left[\frac{\partial^2 \boldsymbol{\nu}}{\partial \mathbf{q} \partial \mathbf{q}} \right]\!\right] = \mathbf{T}_1 + \mathbf{T}_2 + \mathbf{T}_2^\top \tag{12}$$

$$\begin{aligned} \mathbf{T}_1 = &-\boldsymbol{\xi}_{\boldsymbol{\tau}}^\top \left[\!\left[\frac{\partial^2}{\partial \mathbf{q} \partial \mathbf{q}} \left[\mathbf{H}\ddot{\mathbf{q}} + \mathbf{h} - \mathbf{J}_c^\top \boldsymbol{\lambda} \right] \right]\!\right] \\ &- \boldsymbol{\xi}_{\mathbf{a}}^\top \left[\!\left[\frac{\partial^2}{\partial \mathbf{q} \partial \mathbf{q}} \left[\dot{\mathbf{J}}_c \dot{\mathbf{q}} + \mathbf{J}_c \ddot{\mathbf{q}} \right] \right]\!\right] \end{aligned} \tag{13}$$

$$\mathbf{T}_2 = -\boldsymbol{\xi}_{\boldsymbol{\tau}}^\top \left[\!\left[-\frac{\partial \mathbf{H}}{\partial \mathbf{q}} \frac{\partial \ddot{\mathbf{q}}}{\partial \mathbf{q}} - \frac{\partial \mathbf{J}_c^\top}{\partial \mathbf{q}} \frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{q}} \right]\!\right] - \boldsymbol{\xi}_{\mathbf{a}}^\top \left[\!\left[\frac{\partial \mathbf{J}_c}{\partial \mathbf{q}} \frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} \right]\!\right]$$

Herein, we can efficiently compute all the terms in (12) without the tensor operations and in lower computational complexity. Consider that, with appropriately chosen inputs to the $\mathrm{mRNEAc}(\cdot)$, we can attain all the necessary terms as

$$\mathbf{T}_1 = \frac{\partial}{\partial \mathbf{q}} \left[\frac{\partial}{\partial \mathbf{q}} \ \mathrm{mRNEAc}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{a}_g, \boldsymbol{\lambda}, \boldsymbol{\xi}_{\boldsymbol{\tau}}, \boldsymbol{\pi}) \right]$$

$$\mathbf{T}_2 = \frac{\partial}{\partial \mathbf{q}} \ \mathrm{mRNEAc}\left(\mathbf{q}, 0, \frac{\partial \ddot{\mathbf{q}}}{\partial \mathbf{q}}, 0, \frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{q}}, \boldsymbol{\xi}_{\boldsymbol{\tau}}, \boldsymbol{\pi} \right)$$

The term $\mathbf{T}_1$ can be calculated in $\mathcal{O}(n^2)$ complexity. First, reverse-mode AD tools can be used to compute $\frac{\partial}{\partial \mathbf{q}} \mathrm{mRNEAc}(\cdot)$ in $\mathcal{O}(n)$, and then AD tools can be used again for the partial of that result, setting the complexity at $\mathcal{O}(n^2)$. The term $\mathbf{T}_2$ can be calculated in $\mathcal{O}(n^2)$ complexity by first running mRNEAc n times with the n columns of $\frac{\partial \ddot{\mathbf{q}}}{\partial \mathbf{q}}$ as input. Taking the partials of that result using AD tools sets the complexity of this operation at $\mathcal{O}(n^2)$.

The effect is that all the second-order partials of the dynamics function are calculated efficiently in $\mathcal{O}(n^2)$ and without resorting to any tensor operations. Overall, this result reduces the computational complexity of taking the second-order partials needed for DDP compared to an approach based on tensor contraction.

IV. Results

Dynamics Partials

First, we evaluate the proposed methods on an underactuated $n-$link pendubot model whose last link is pinned to the ground through a pin joint. The $n-$link pendubot model allows us to test the scalablity of the proposed methods with an increasing number of DoFs. This work was implemented in MATLAB alongside CasADi [17], which allows for rapid and efficient testing of AD approaches. As shown in Fig. 2, the evaluation of the second-order partials using Tensor DDP took the longest time. The computation of the second-order partials via mRNEAc DDP showed a order reduction compared to Tensor DDP (as given by a reduction of the slope by more than one on the loglog plot). Computation time benefits were especially more apparent for systems with a higher number of DoFs, indicating the potential of the proposed methods to include second-order information into trajectory optimization algorithms for legged robots.

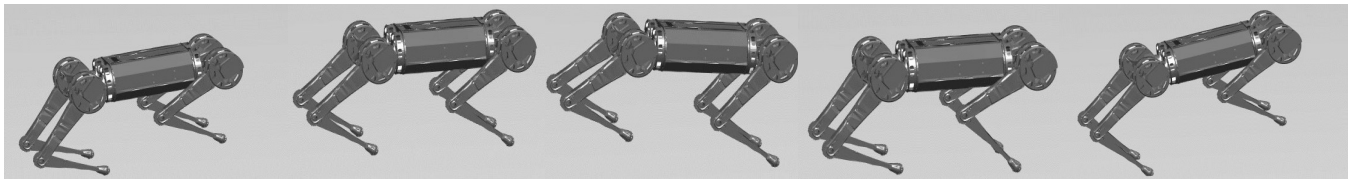

Fig. 1. Snapshots of the resulting optimized motion by DDP.

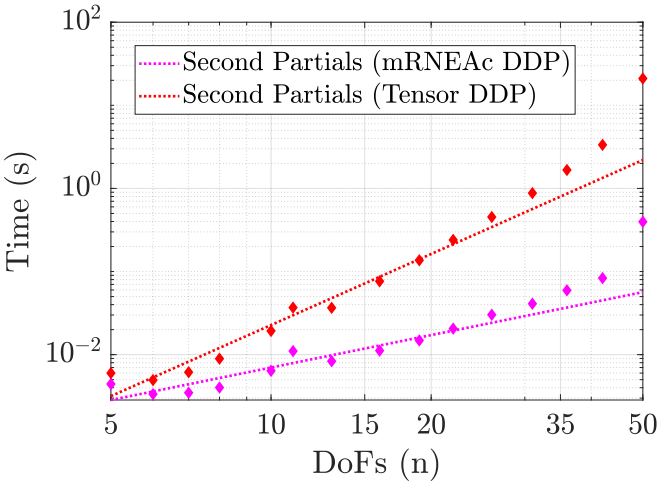

Fig. 2. The computation time of the dynamics partials of using the proposed methods of the pinned pendubot model.

Trajectory Optimization

Next, we consider trajectory optimization for a bounding gait with a planarized quadruped. The model of the 2D quadruped has 5 links (torso, and 2 links per leg) and a state dimension of $n = 14$. The bounding sequence is encoded via the design of the running and terminal costs in each bounding mode. The resulting motion following the trajectory optimization by DDP is illustrated in Fig. 1. The robot starts in the back-stance mode and runs forward at a speed of 0.5m/s.

Further, we consider several instances of adding random noise to study the timing advantages of the proposed DDP approach. Different realizations of Wiener process noise were added to the optimized state and control trajectories. Those modified trajectories then used as initial trajectories, and the algorithm was optimized for 50 iterations. As shown in Fig. 3, mRNEAc on average took less time (approx. 5.7111 times less) as compared to the Tensor DDP. The reduced effort to add second-order information and reduced optimization time of mRNEAc DDP indicates the potential of the proposed approach over conventional approaches.

V. Discussion

The proposed DDP approach shows an improvement in computational complexity and runtime of DDP as compared to the conventional DDP approach. The computational complexity of the proposed method has a lower polynomial degree ($\mathcal{O}(n^2)$) than that of Tensor DDP ($\mathcal{O}(n^3)$). This reduction was achieved by the use of the mRNEAc algorithm, which allows us to cheaply compute the contact-constrained dynamics sensitivities using reverse-mode AD. While second-order dynamics information is often ignored in literature, the outcomes of this work suggest its consideration for broader inclusion in DDP.

References

[1] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, 2012.

[2] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback MPC for torque-controlled legged robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4730–4737, 2019.

[3] 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.

[4] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *IEEE International Conference on Robotics and Automation*, pp. 1168–1175, 2014.

[5] 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.

[6] T. Howell, B. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 7674–7679, 2019.

[7] 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.

[8] N. J. Kong, G. Council, and A. M. Johnson, "iLQR for piecewise-smooth hybrid dynamical systems," in *IEEE Conference on Decision and Control (CDC)*, pp. 5374–5381, 2021.

[9] Y. Tang, X. Chu, and K. Au, "Hm-ddp: A hybrid multiple-shooting differential dynamic programming method for constrained trajectory optimization," *arXiv preprint arXiv:2109.07131*, 2021.

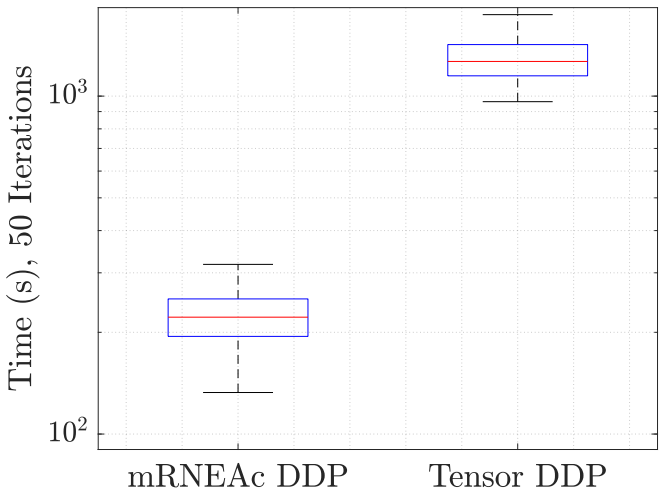

Fig. 3. Timing comparison of the proposed method as compared to the conventional method

[10] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," in *ICINCO (1)*, pp. 222–229, Citeseer, 2004.
[11] J. Nganga and P. M. Wensing, "Accelerating second-order differential dynamic programming for rigid-body systems," *IEEE Robotics and Automation Letters*, 2021.
[12] E. V. Denardo, *Dynamic programming: models and applications*. Courier Corporation, 2012.
[13] 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*, pp. 1–9, 2018.
[14] R. Budhiraja, *Multi-body Locomotion: Problem Structure and Efficient Resolution*. PhD thesis, Institut national des sciences appliquées de Toulouse, 2019.
[15] 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.
[16] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.
[17] 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.