Second-Order Differential Dynamic Programming for Whole-Body MPC of Legged Robots

John Nganga[1], He Li[1], and Patrick M. Wensing[1]

***Abstract*— This work presents the first use-case of full second-order Differential Dynamic Programming (DDP) with a whole-body model for model predictive control on legged robot hardware. Recent advances in the literature show that DDP can be simplified by exploiting the dynamics structure in the algorithm, allowing the use of reverse-mode derivative accumulation to efficiently compute sensitives. These advances allow DDP to be run at similar compute times as the iterative LQR (iLQR) algorithm, its first-order counterpart. Beyond a hardware implementation of these past theoretical developments for a robot making regular contact with the environment, this paper provides a characterization of full DDP vs. iLQR following push disturbances with a quadruped robot. The resulting DDP controller is shown to achieve lower costs and withstand bigger disturbances as compared to iLQR.**

***Paper Type* – Original Work**

I. Introduction

Most existing robot locomotion controllers based on model predictive control (MPC) consider a reduced model of the robot's dynamics when solving for the optimal control. The choice of a reduced model (template), at its core, aims to capture as much important information as possible about the robot's dynamics. While the inclusion of more dynamic information may lead to a better controller overall, care must be taken to ensure that the resulting dynamic information does not lead to dimension explosion and increased computation time. However, with increasing computational capabilities, the trend toward more complex template models and whole-body trajectory optimization is steadily gaining traction. For example, the authors in [1] used a whole body algorithm known as Differential Dynamic Programming (DDP) to control a simulated humanoid at seven times slower than real-time. [2] used DDP to control a 22-DOF humanoid robot walking and making contact with the environment and [3] a quadruped performing dynamic such as cantering.

DDP is a shooting method that provides dynamically feasible trajectories at any iteration of its optimization process, encouraging its use in an MPC fashion. The algorithm provides both a feed-forward control tape and a locally optimal feedback policy, which can be used to handle disturbances [4]. Originally described by [5] as a single shooting method for smooth unconstrained nonlinear systems, DDP has been extended to handle control limits [1], state-control constraints [6], [7] and hybrid dynamics [7]–[10] where the robot has to make and break contact with its environment. In the literature, DDP presentations often consider the second-order approximation of the robot's dynamics, but in practice, only the first-order dynamics approximation [2], [9], [11] is used due to its faster evaluation time. This configuration gives rise to the iterative linear quadratic regulator (iLQR). While the second-order dynamics sensitivity information in DDP retains higher fidelity to the full model locally, these sensitivities are tensorial and represent the most intensive computation in DDP. The work in [12], [13] presented an algorithmic method to efficiently include the second-order information. It resulted in a reduction from cubic to quadratic computational complexity (in the number of bodies of the robot) compared to conventional tensor methods. The advancement therein showed that full second-order DDP could be performed in similar compute time as first-order iLQR, potentially allowing the use of DDP on hardware. It should be noted that even for strictly convex running cost functions, the inclusion of second-order dynamics information may lead to a non-convex approximation of the local cost landscape. As such, a reconditioning of the cost landscape known as regularization (i.e., via an artificial Hessian shift) is often needed and we note that this incurs additional compute effort for DDP as compared to iLQR.

It is hypothesized that since DDP includes more information about the full dynamics of the robot, it is likely to be able to reject larger disturbances compared to iLQR. We evaluate this hypothesis via Monte Carlo simulations, where we vary the disturbance applied to the robot and characterize the resultant properties of the robot's motion.

The results presented herein are enabled by efficiently including second-order information in the DDP algorithm [12], [13]. These advances allow (I) the presentation of full second-order DDP on hardware for a robot making regular contact with the ground. Further, in simulation, we use this advancement to perform a Monte Carlo characterization of the disturbance rejection capabilities of DDP/iLQR when used for closed-loop MPC. The results presented here show that (II) DDP can withstand larger disturbances as compared to iLQR and without sacrificing significant compute time.

II. Background: Trajectory Optimization

Consider a model with discrete-time dynamics

$$\begin{bmatrix}\mathbf{x}_{k+1}\\ \boldsymbol{\lambda}_k\end{bmatrix} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) = \begin{bmatrix}\mathfrak{F}(\mathbf{x}_k, \mathbf{u}_k)\\ \mathfrak{g}(\mathbf{x}_k, \mathbf{u}_k)\end{bmatrix} \tag{1}$$

where $\mathbf{x} = [\mathbf{q}^\top \ \dot{\mathbf{q}}^\top]^\top$ is the state vector and $\mathbf{q}$ is the generalized coordinates. Further, $\mathbf{u}_k$ is the control input at time k with $\boldsymbol{\lambda}_k$ being the ground reaction force at that time instant. The term $\mathfrak{F}$ represents the forward dynamics, whereas $\mathfrak{g}$ represents the realized contact forces function. For a model (or a contact mode, e.g., flight mode) that does not exert

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,hli25,pwensing}@nd.edu

contact with the environment, $\boldsymbol{\lambda}_k = \mathfrak{g} = \emptyset$. Therefore in (1), $\mathbf{f}$ captures both the contact-free and constrained dynamics as appropriate.

DDP [5] is used to solve an optimal control problem (OCP) of the form

$$\mathbf{J}(\mathbf{U}) = \Phi_k(\mathbf{x}_N) + \sum_{k=0}^{N} \ell_k(\mathbf{x}_k, \mathbf{u}_k, \boldsymbol{\lambda}_k) \tag{2}$$

where $\mathbf{J}(\mathbf{U})$ represents the total cost incurred when starting at some initial state, $\mathbf{x}_0$ and applying a control sequence $\mathbf{U} = [\mathbf{u}_1, , ..., \mathbf{u}_N]$. The term, ℓ_k is the running cost that measures the cost of taking some control $\mathbf{u}_k$ at the intermediate state $\mathbf{x}_k$. The term Φ_k is the terminal cost. To obtain an optimal trajectory, (2) is minimized such that

$$\min_{\mathbf{U}} \ \mathbf{J}(\mathbf{U}) \tag{3}$$

$$\text{subject to } \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \tag{4}$$

$$\text{Other Constraints .} \tag{5}$$

In solving (3), DDP/iLQR seeks the optimal control tape $\mathbf{U}_0^\star(\mathbf{x}_0) = \underset{\mathbf{U}_0}{\text{argmin}} \ \mathbf{J}(\mathbf{U}_0)$, where the superscript $\star$ implies an optimal quantity. Rather than optimizing over the entire control tape, DDP/iLQR solves (3) by optimizing a control policy at each time point. This process recursively provides an approximation of the value function

$$V_k(\mathbf{x}_k) = \min_{\mathbf{u}_k} \Big[\underbrace{\ell_k(\mathbf{x}_k, \mathbf{u}_k, \mathfrak{g}(\mathbf{x}_k, \mathbf{u}_k)) + V_{k+1}(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k))}_{Q_k(\mathbf{x}_k, \mathbf{u}_k)} \Big]$$

$$\text{where} \quad V_N(\mathbf{x}_N) = \Phi(\mathbf{x}_N)\,, \tag{6}$$

where the function, $Q_k(\mathbf{x}_k, \mathbf{u}_k)$, 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. Consider the differential change to Q_k in (6) around a nominal state-control pair $\bar{\mathbf{x}}_k$ and $\bar{\mathbf{u}}_k$ with

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

The second-order approximation of the Q function gives

$$Q_{\mathbf{x}} = \ell_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^\top V'_{\mathbf{x}} \tag{7a}$$

$$Q_{\mathbf{u}} = \ell_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^\top V'_{\mathbf{x}} \tag{7b}$$

$$Q_{\mathbf{xx}} = \ell_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^\top V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}} + V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{xx}} \tag{7c}$$

$$Q_{\mathbf{uu}} = \ell_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^\top V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{u}} + V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{uu}} \tag{7d}$$

$$Q_{\mathbf{ux}} = \ell_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^\top V'_{\mathbf{xx}} \mathbf{f}_{\mathbf{x}} + V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{ux}}\ . \tag{7e}$$

where the prime in (7) denotes the next time step, i.e., $V'_{\mathbf{xx}} = V_{\mathbf{xx}}(k{+}1)$ whereas the subscripts indicate partial derivatives. The operation $\cdot$ denotes a contraction with a tensor, e.g., $V'_{\mathbf{x}} \cdot \mathbf{f}_{\mathbf{xx}}$ denotes the contraction of the tensor $\mathbf{f}_{\mathbf{xx}}$ with the vector $V'_{\mathbf{x}}$ to attain a matrix. When the $\cdot$ operation is ignored, i.e., when second-order partials of the dynamics are ignored, the resulting algorithm is known as iLQR [2].

Minimizing (6) over $\delta\mathbf{u}_k$ attains the incremental control

$$\delta\mathbf{u}_k^\star = -Q_{\mathbf{uu}}^{-1} Q_{\mathbf{u}} - Q_{\mathbf{uu}}^{-1} Q_{\mathbf{ux}} \delta\mathbf{x}_k\,. \tag{8}$$

The resulting control from (8) is used to form the quadratic approximation of the value function and 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 under the incremental control policy (8) resulting in a new state-control trajectory. We refer the interested reader to [5] or [2] for a detailed derivation of the algorithm, and [10] for detailed derivation when the hybrid dynamics are considered.

A. DDP vs iLQR

The main difference between iLQR and DDP is that the vector-tensor contraction in (7) is ignored in iLQR. This contraction contains second-order information about the dynamics, and its inclusion in DDP confers quadratic convergence for trajectories that are sufficiently close to local optimality [2]. However, even for a strictly convex running cost function, the inclusion of those terms may cause the ensuing control cost landscape in (6) to have negative curvature in some directions, which corresponds with $Q_{\mathbf{uu}}$ being indefinite. This non-convexity necessitates a Hessian shift (regularization) to render $Q_{\mathbf{uu}}$ positive definite (PD) [2], [14], and this process incurs additional computational cost. Following the regularization, the entire backward sweep is repeated.Therefore, this regularization procedure, may at times, increase the computation cost of the algorithm.

Despite this challenge for DDP, it is theorized that since DDP reasons about the second-order partials of the dynamics, policies based upon it are likely to have better disturbance rejection capabilities compared to iLQR. Conventionally, the inclusion of second-order tensorial dynamics terms in DDP is the most arduous operation. The authors' previous work [12], [13] showed that this computation cost could indeed be relaxed by considering the structure of dynamics formulation in (7) such that the tensorial terms are avoided. We briefly review the advancement needed to allow the efficient inclusion of second-order partials in DDP.

III. Toward Hardware Implementation

A. Second-order Dynamics

Consider that if the dynamics function $\mathbf{f}(\mathbf{x}, \mathbf{u}) \in \mathbb{R}^n$, its first-order partial $\nabla_{\mathbf{x}}\mathbf{f} \in \mathbb{R}^{n\times n}$, and its second-order partials $\nabla_{\mathbf{x}}^2\mathbf{f} \in \mathbb{R}^{n\times n\times n}$. In (7), this second-order partial will be left multiplied with a fixed vector $\boldsymbol{\gamma} \triangleq V'_{\mathbf{x}}$. Both the calculation of $\nabla_{\mathbf{x}}^2\mathbf{f}$ and subsequent tensor-contraction are computationally expensive and represent a bottleneck in DDP.

The work [12] developed an efficient method to include this second-order information without resorting to any tensor operations. The key derivation therein exploited the structure of the dynamics partials in (7), recognizing the second-order partials are left multiplied with a fixed vector $V'_{\mathbf{x}}$. Reverse mode derivative accumulation allowed efficiently calculating the tensor contraction without ever computing the tensor. That work demonstrated an order of magnitude reduction in computational complexity when computing the second-order dynamics partials as compared to conventional tensor

operations. In this letter, we rely on a similar methodology to enable the inclusion of second-order information in DDP.

IV. Results

We benchmark this work using a similar formulation as in [15] where we optimize a trotting gait for the MIT Mini-Cheetah [16] robot using the hybrid Kinodynamic (HKD) model. The HKD model considers the dynamics of the floating base using the single rigid body dynamics and models leg kinematics during the swing phase. During stance, as the endpoint (i.e., foot) of each leg should not change, only the endpoint position is modeled. The running cost penalizes deviations from a reference trajectory and further includes a smoothing control term, as detailed above. The DDP/iLQR formulation is implemented in a similar fashion as [10] where we consider the hybrid nature of the trotting gait and dynamic constraints such as friction cone constraints. We use a custom DDP solver [10] running on a standard desktop computer. Following an initial solve of DDP/iLQR, the MPC solver plans at 50 Hz, with each solve using the previous solution to warm-start the current solution. In the MPC setting, the number of DDP/iLQR iterations is limited to 2 iterations. The main control loop/state estimator run at 500 Hz. In this work, Forward Euler integration is assumed with a time step of 0.011s and a planning horizon of 0.32s.

To characterize the contributions of our approaches, we simulate the robot trotting forward at forward at 0.5m/s and from a static initial position. In a Monte Carlo fashion, we apply an instantaneous velocity change to the robot at the $5-$second mark in the transverse plane. Essentially, this velocity change acts as a disturbance to the robot. The velocity change is sampled randomly from a pseudo-random disturbance generated uniformly from -2 to 2 m/s. We characterize the effects of this velocity change following the disturbance injection. This characterization is done for both DDP/iLQR.

A. Simulation

In Fig. 2 and Fig. 3, we show results for the trunk roll and yaw angle, respectively, across the experiments. The plot shows a band for the min/max angles at each time across the trials. For both Fig. 2 and Fig. 3, we limit our discussion to a sampling of lateral velocity changes within ± 0.5 m/s. The robot was also simulated without the instantaneous velocity change, and the average response of the roll/yaw angle over time is plotted as well to provide a baseline. Since the simulation includes non-deterministic inter-process communication between control components, the averaging removes small discrepancies that may exist between different runs. To eliminate the influence of outliers, we include a plot that shows the 75-th percentile value of the data. We also indicate on the plot, the maximum value over this subset of data. This value can be considered as an indicator of the robot's worst achieved angle in 75% of cases. We note that the jump from an initial static position to then accelerating toward the desired speed introduces a transient to the robot at the start and this behavior is also reflected in the extremal bands.

For the roll angle (Fig. 2), the iLQR approach performed seemingly better as compared to DDP approach. The iLQR algorithm with the maximal angle (over 75% of the data) of 0.1825 rad achieved a lower maximum as compared to the Traditional DDP angle of 0.2936 rad. However, this result is reversed for the yaw angle (Fig. 3) whereby DDP approach outperformed iLQR algorithm. For this metric, traditional DDP had the lower maximal angle of 0.0519 rad, with iLQR's maximal angle being at 0.0659 rad.

Since an individual algorithm may choose to prioritize one metric, such as yaw, over another, we consider the rollout cost (Fig. 4) of the MPC policy as another relevant metric. This metric reflects the sum of all minimized costs, including terminal costs, after each MPC update. In that sense, this figure captures the quality of each MPC solution across all disturbances and simulation instances. As shown, DDP attains a lower cost rollout as compared to iLQR. Further, the iLQR approach had a bigger spread in the rollout cost achieved as compared to DDP. This result indicates that DDP methods more quickly reattained the local optimality as compared to iLQR.

To characterize push recovery capabilities from another perspective, we consider the maximal velocity change that either method withstood at least once within the Monte-Carlo simulations. DDP was about to withstand a bigger velocity magnitude change of $|1.028|$m/s velocity change whereas iLQR was able to withstand a velocity magnitude of $|0.7|$ m/s. We take care to note that these results do not, in general, mean that DDP will, in all instances, withstand bigger disturbances as compared to iLQR. Rather, when the nominal trajectory is close to an optimal trajectory or when the injected disturbance does not substantially change the control landscape to warrant a complete re-solve, DDP will more quickly converge to the optimal trajectory. The results in this paper may only be seen in that context and not through the lens of global optimality guarantees.

In Fig. 5, we show the average time to compute the forward pass, backward pass, and the dynamics partials using the different approaches. DDP, in general, took longer than iLQR. This is due to the fact that DDP includes the calculation of the second-order partials and, in certain instances, the regularization (in the backward pass) procedures incur additional timing costs. From Fig. 5, it is shown that the extra compute times in DDP is incurred mostly in the backward pass, i.e., regularization procedure, and minorly from the calculation of the dynamics partials.

B. Hardware

We perform similar tests as above on the MIT Mini-Cheetah robotic hardware. In Fig. 1, we show screenshots of the MIT Mini-Cheetah trotting on a treadmill. Further, in Fig. 6, we show the roll tracking of the robot using iLQR/DDP using both regularization methods. There is no discernible difference in the tracking control between the methods. Further, in Table I, we show the average policy lag for the robot wherein similar results as before are attained (see Fig. 5). Policy lag refers to the delay between the time a

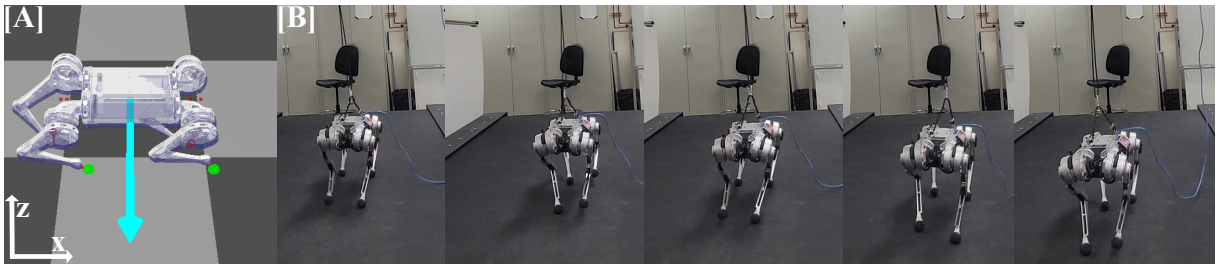

Fig. 1. (a) Simulation of the MIT Mini Cheetah in simulation. The blue arrow indicates the application of an instantaneous velocity change in the y direction (b) Video splice of the MIT Mini-Cheetah trotting on a treadmill.

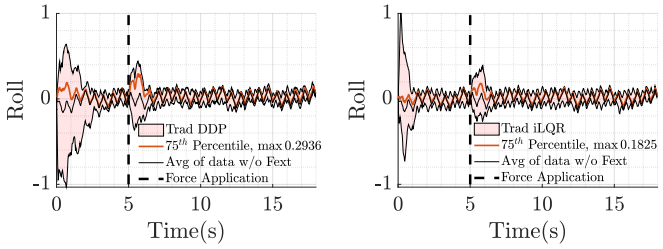

Fig. 2. Roll characterization in a Monte-Carlo simulation

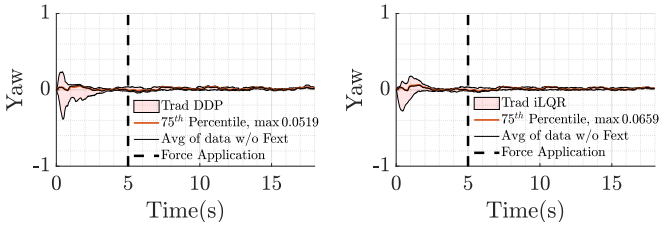

Fig. 3. Yaw Characterization in a Monte-Carlo simulation

policy decision is made and the time it is implemented, i.e., the time that the robot is using a stale policy awaiting a new policy. The DDP approach took slightly more time but the net difference in comparison with iLQR is no more than a few milliseconds (ms). Overall, the policy lags indicated in Table I was still fast enough to run on hardware platforms.

V. Discussion

We, firstly, note that this is the first time in literature that full second-order DDP has been shown to work in an MPC fashion on hardware owing to the recent technical contributions outlined herein. This advancement has previously not been possible since the inclusion of second-order information often requires expensive dynamic calculations. The advantages of full second-order DDP outlined here would allow for DDP to be deployed in environments that inadvertently introduce disturbances to the robot, such as allowing legged robots to walk in novel terrains. This work shows that DDP can be used to attain a lower cost at about the same compute time as iLQR even when running in MPC.

Finally, we note that in our analysis, we stated that DDP handled bigger velocity changes as compared to iLQR. We take care to note that these results do not, in general, mean that DDP will, in all instances, withstand bigger disturbances as compared to iLQR. The improved convergence properties of DDP over iLQR only hold locally around an optimal trajectory. As a result, any disturbance that substantially changes the optimal trajectory will require many iterations to find an optimal trajectory, wherein the comparative benefits of DDP over iLQR only hold in final iterations. As a result, it is difficult to conclusively say whether iLQR or DDP would perform better in these more extreme cases.

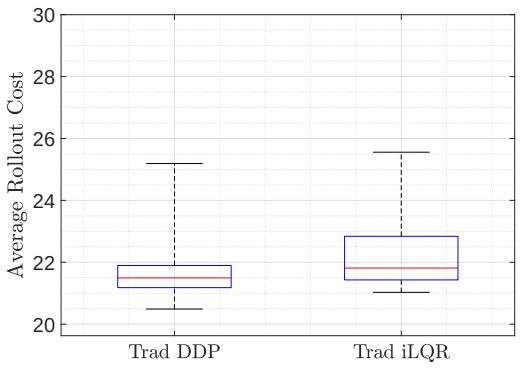

Fig. 4. Average Rollout cost over numerous instances

TABLE I

Policy Lag (ms)

DDP		iLQR	
Mean (ms)	Std. Dev. (ms)	Mean (ms)	Std. Dev. (ms)
11.39	5.84	9.7	5.83

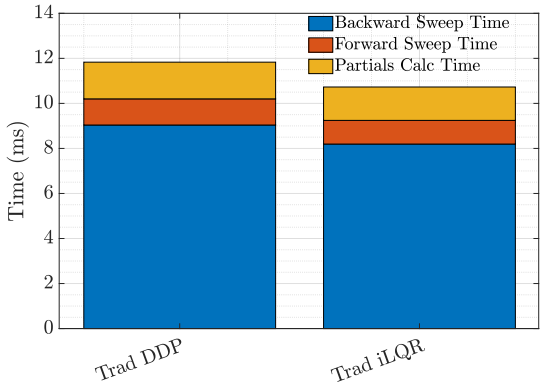

Fig. 5. Average time to compute forward, backward passes and time to calculate the partials for DDP/iLQR.

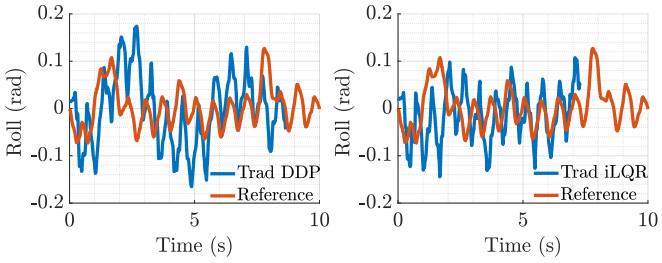

Fig. 6. Roll tracking results on the MIT Mini-Cheetah

REFERENCES

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

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

[3] Z. Zhou, B. Wingo, N. Boyd, S. Hutchinson, and Y. Zhao, "Momentum-aware trajectory optimization and control for agile quadrupedal locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7755–7762, 2022.

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

[5] 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.

[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, November 2019.

[7] 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.

[8] 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.

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

[10] 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.

[11] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.," in *ICINCO (1)*, pp. 222–229, 2004.

[12] J. N. Nganga and P. M. Wensing, "Accelerating second-order differential dynamic programming for rigid-body systems," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7659–7666, 2021.

[13] J. N. Nganga and P. M. Wensing, "Accelerating hybrid systems differential dynamic programming," *ASME Letters in Dynamic Systems and Control*, vol. 3, no. 1, p. 011002, 2023.

[14] 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.

[15] H. Li, W. Yu, T. Zhang, and P. M. Wensing, "Zero-shot retargeting of learned quadruped locomotion policies using hybrid kinodynamic model predictive control," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11971–11977, IEEE, 2022.

[16] B. Katz, J. Di Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 international conference on robotics and automation (ICRA)*, pp. 6295–6301, IEEE, 2019.