Trajectory Optimization of Contact-rich Motions using Implicit Differential Dynamic Programming

Trajectory Optimization of Contact-rich Motions using Implicit Differential Dynamic Programming

Iordanis Chatzinikolaidis and Zhibin Li

***Abstract*—This paper presents a Differential Dynamic Programming (DDP) approach for systems characterized by implicit dynamics using sensitivity analysis, such as those modelled via inverse dynamics, variational, and implicit integrators. It leads to a more general formulation of DDP, enabling the use of the faster recursive Newton-Euler inverse dynamics. We leverage the implicit formulation for precise and exact contact modelling in DDP, where we focus on two contributions: (1) contact dynamics at the acceleration level; (2) formulation using an invertible contact model in the forward pass and a closed-form solution in the backward pass to improve the numerical resolution of contacts. The performance of the proposed framework is validated by comparing implicit versus explicit DDP for the swing-up of a double pendulum, and by planning motions for two tasks using a single leg model making multi-body contacts with the environment: standing up from ground, where a priori contact enumeration is challenging, and maintaining balance under an external perturbation.**

I. Introduction

Trajectory optimization (TO) has attracted increasing research interest for motion planning and control of highly dynamical, underactuated robots [1]. This is due to the potential of generating complex motions in a high-level manner: A user can design and specify a desired task using physical terms with associated weights via a cost function, which can also be automatically tuned [2], and a motion planner is able to automatically generate a sequence of feasible motions [3].

This is particularly interesting for robotic systems that require through-contact motion plans, *i.e.* plans that involve multiple unspecified contact interactions. Physical contacts are traditionally difficult to model and incorporate in motion planning frameworks. Most approaches are multiphase, in the sense that contact schedule patterns [4] or corresponding timings are provided a priori, while contacts are desirable with the end-effectors only. This leads to difficulties in practical implementations because selection of locations and timings is in general non-trivial, while restricting contacts to end-effectors only limits the motion repertoire.

DDP—a prominent shooting TO methodology—is among the most promising approaches for its efficiency in through-contact motion planning. This is demonstrated by a multitude of previous works that used DDP as backbone: From the simulated results [5], to real-time applications for high-dimensional legged robots [6], [7]. However, properly modelling contacts is a considerable challenge; most DDP implementations resort to approximations and simplifications requiring well-tuned contact parameters. A fundamental reason is that contact phenomena are canonically described implicitly.

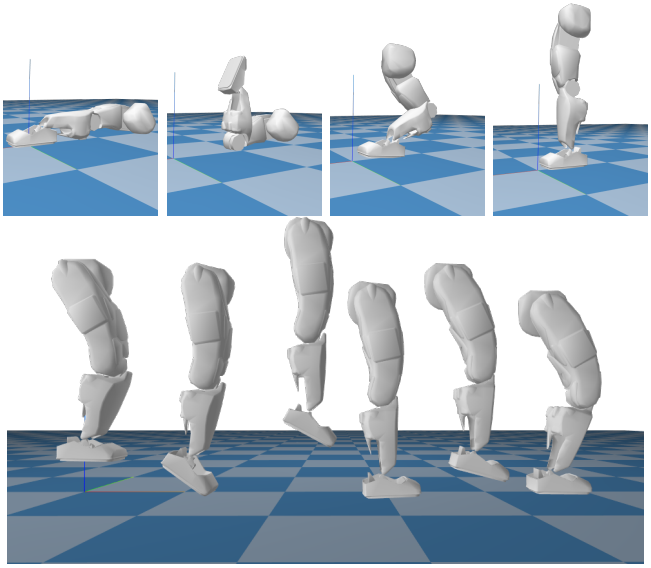

Fig. 1. Complex multi-contact motions of a single leg robot computed by the proposed framework in time-lapsed snapshots: dynamic standing up from the ground (top), and balancing against an external perturbation (bottom).

The original DDP algorithm and its subsequent studies often assume that the discrete-time systems considered are explicitly defined. Thus, it relies only on the forward sensitivity of the state's evolution. Yet DDP can be readily applied to implicitly defined dynamical systems [8] when combined with appropriate sensitivity analysis. These are typically more challenging because they require the solution of nonlinear equations. However, they offer computational advantages, *e.g.* providing stability even for stiff differential equations. Further, handling implicitly defined systems allows more principled contact modelling in DDP.

A. Contributions

In this work, we provide *theoretical* and *algorithmic* contributions as:

- A sensitivity analysis approach for applying DDP to explicitly and implicitly defined systems in a unified manner.
- Based on this, we propose an approach leveraging an invertible model [9] for exact contact resolution in DDP.
- Results demonstrating the possibility of exploiting properties of implicit integrators in DDP settings.

We benchmark our approach by applying it on implicitly and explicitly defined models, and on two cases of multi-

contact whole-body motion planning for a planar single-leg robot that makes multi-body contacts: standing-up from ground and balancing from an initial perturbation in a receding horizon fashion (Fig. 1). Our approach is equally applicable to models with large degrees of freedom and arbitrary contact configurations, such as using multiple legs.

The remaining sections are organized as follows. Sec. II discusses prior work on the DDP algorithm, and applications of DDP for through-contact motion planning. Sec. III summarizes DDP and how contacts are typically resolved in simulation. In Sec. IV, we present our application of DDP and, in Sec. V, how to utilize it for through-contact planning. Sec. VI provides comparisons between explicit and implicit systems in the context of DDP, and two motion planning studies for a single leg standing up and balancing in multi-contact settings. We summarize and conclude in Sec. VII.

II. Prior Work

A. Differential Dynamic Programming

DDP was originally introduced in [10]. Its main advantage with respect to the Dynamic Programming algorithm [11] is that it does not suffer from the curse of dimensionality by sacrificing global optimality. Subsequently, a number of improvements of DDP have been introduced. Recently, there was a resurgence of interest due to its potential for efficient planning for high-dimensional systems.

DDP is a second-order algorithm that exhibits quadratic convergence similar to Newton's methods [12]. Thus, it requires second-order information, which can be computationally challenging for high-dimensional models. To resolve this, the iLQR variant performs a Gauss-Newton approximation of the Hessian based on first-order information only, albeit with superlinear convergence [13].

The original DDP algorithm is concerned with unconstrained discrete dynamical systems only. Control bounds can be considered via a projected Newton quadratic programming solver [14]. More general nonlinear inequality constraints via an active-set method [15]. In robotics, it is common to consider multiple tasks in a hierarchical fashion, which is possible to do for DDP too [16]. In legged locomotion, the discontinuous nature of contact phenomena has led to the development of tailored approaches. For example, a pre-defined gait pattern and centroidal dynamics model was considered in [17], and more general hybrid systems in [18]. We underline that the DDP framework presented next can incorporate the previous approaches straightforwardly.

Finally, a brief discussion about the application of DDP for implicitly defined systems from a Lie theoretic viewpoint is given in [19]. Here, we present a more complete and deep treatment, with extensive comparisons. Furthermore, our vector-based formulation is much more familiar and common for robotic systems applications.

B. Through-contact Motion Planning

Applications of DDP for motion planning and control of legged robots have been very impressive. From simple, approximate models up to whole-body models, DDP provides a means for fast and even real-time solutions.

In [5], DDP is used to control a humanoid model. A diverse set of behaviours is generated by simply changing weights in the cost function through a graphical user interface. An approximate solution for the contact dynamics is used, with a contact model similar to the one that is used here. The implicit formulation that we present next allows the consideration of contacts in DDP without requiring approximations to the contact model itself.

For quadruped robots, a diverse set of motions both in simulation and in hardware is shown in [6]. To take into account contacts, a nonlinear spring-damper model was used. Even though tuning for each contact is done independently, spring-damper models can be difficult to tune in practice and require very small time steps. It is common for the optimizer to explore states where the current model parameters are not valid, while the small time steps translate into a large problem. Here, in the forward pass, the model takes into account all possible contacts in a centralized manner (through the coupling with the contact-space inertia matrix), while independently solve for each contact in the backward pass (by leveraging our implicit DDP formulation and the model's invertibility). Thus, performance is similar to complementarity formulations with large time steps, while we are capable to compute straightforwardly gradients in the backward pass.

To eliminate the unrealistic effects of spring-damper models, a hard contact model is used in [20]. Unfortunately, contact impulses require the numerical solution of a quadratically constrained quadratic program (QCQP), typical in time-stepping approaches with unilateral and friction cone constraints, and formulates the problem in a bilevel fashion. This complicates the derivative computation due to the numerical nature of the solution. We resolve this issue by leveraging the invertibility of the contact model: in the forward pass, the QCQP is solved with the associated constraints; in the backward pass, a closed-form computation is used that avoids the bilevel formulation. As a result, this does not pose issues with differentiation and leads to a faster and simpler implementation, without the need for backpropagation.

A multiple shooting variant is presented in [21], extending the work in [22]. It allows easier initialization since both state and control sequences can be used. Unfortunately, the intermediate iterates of the algorithm are infeasible, meaning that early stopping with a feasible trajectory, as in DDP, is not possible. This is a necessary property in our case since the through-contact motion planning approach that we present is running in a receding horizon fashion. Furthermore, the contact schedule is pre-defined in [21], while here contacts are activated according to the natural dynamics of the system [23]. Finally, friction cone constraints are neglected or can be taken into account through penalization in the cost function, which can be in practice difficult to tune and can lead to unrealistic solutions. Due to the imposition of contacts as equality constraints, attractive forces can arise at the solution, violating the unilateral constraint. Our framework here utilizes full unilateral and

friction cone contact constraints without any approximation or penalization.

III. Preliminaries

A. Summary of Differential Dynamic Programming

DDP is concerned with the optimization of a performance criterion for an unconstrained discrete-time dynamical system [10], [5]. This can be expressed as

$$\min_{u_i} \quad l_f(x_N) + \sum_{i=0}^{N-1} l_i(x_i, u_i) \tag{1a}$$
$$\text{s.t.} \quad x' = f(x, u). \tag{1b}$$

Here, l_i is an additive cost at time step i and l_f is the final cost, x_i and u_i are the state and control, N is the length of the horizon, while $\cdot'$ denotes the quantity at the next time step, *e.g.* the next state in our context.

According to the principle of optimality, (1) can be expressed via the value function, which is the total cost at a given state once we apply the optimal control sequence. The principle of optimality makes the computation of the value function iterative, and at a state x is given by

$$V(x) = \min_u l(x, u) + V'(x') = \min_u l(x, u) + V'(f(x, u)).$$

Since finding the global minimum is challenging, DDP performs a quadratic approximation of the value function and subsequently improves the control sequence $\{u_i\}$ locally. If we define the Q-function as

$$Q(x, u) = l(x, u) + V'(x'), \tag{2}$$

a quadratic approximation about the current point (x_i, u_i) is

$$\begin{aligned} Q(x, u) \approx\ & Q(x_i, u_i) + Q_x(x_i, u_i)\delta x + Q_u(x_i, u_i)\delta u \\ & + \tfrac{1}{2}\begin{bmatrix}\delta x \\ \delta u\end{bmatrix}^T \begin{bmatrix} Q_{xx}(x_i, u_i) & Q_{xu}(x_i, u_i) \\ Q_{ux}(x_i, u_i) & Q_{uu}(x_i, u_i)\end{bmatrix}\begin{bmatrix}\delta x \\ \delta u\end{bmatrix} \end{aligned} \tag{3}$$

while $\delta x = x - x_i$ and $\delta u = u - u_i$ are state and input perturbations. The terms in (3) are computed by expanding and matching same terms in Eq. (2) as

$$\begin{aligned} Q_x &= l_x + V'_{x'} f_x \\ Q_u &= l_u + V'_{x'} f_u \end{aligned} \qquad \begin{aligned} Q_{xx} &= l_{xx} + f_x^T V'_{x'x'} f_x + V'_{x'} f_{xx} \\ Q_{xu} &= l_{xu} + f_x^T V'_{x'x'} f_u + V'_{x'} f_{xu} \\ Q_{uu} &= l_{uu} + f_u^T V'_{x'x'} f_u + V'_{x'} f_{uu}. \end{aligned} \tag{4}$$

Backward pass: The optimal control change δu^* is given by minimizing the unconstrained quadratic equation (3) as

$$\delta u^* = \operatorname*{argmin}_u Q(x, u) - u_i = \underbrace{-Q_{uu}^{-1} Q_u^T}_{k} \underbrace{-Q_{uu}^{-1} Q_{ux}}_{K} \delta x \tag{5}$$

The quadratic approximation of the value function at the current time step in (3) becomes

$$\delta V = V(x) - Q(x_i, u_i) = \tfrac{1}{2} Q_u k \tag{6a}$$
$$V_x = Q_x + Q_u K \tag{6b}$$
$$V_{xx} = Q_{xx} + Q_{xu} K, \tag{6c}$$

with boundary values $V_x^N = l_x^N$ and $V_{xx}^N = l_{xx}^N$.

Forward pass: Once the feedforward and feedback terms k_i and K_i for each time step are computed, we perform a forward pass to compute the updated control sequence as

$$\hat{x}_0 = x_0 \tag{7a}$$
$$\hat{u}_i = u_i + \delta u^* = u_i + k + K(\hat{x}_i - x_i) \tag{7b}$$
$$\hat{x}_{i+1} = f(\hat{x}_i, \hat{u}_i) \tag{7c}$$

for $i \in [0, N-1]$. In practice, regularization and line search are necessary, as explained in [5].

B. Simulation With Contacts

We summarize a typical simulation pipeline in the presence of contacts [24]. Contact resolution is usually done in the velocity–impulse level but our DDP is formulated at the acceleration–force level, which will be elaborated later.

The dynamics of a mechanical system are given by

$$M(q)\dot{v} + H(q, v) = S\tau + J^T(q) f, \tag{8}$$

where M the mass matrix, H the vector of nonlinear forces, S a selection matrix that maps actuated joint torques τ to generalized coordinates, while J denotes the concatenated Jacobian of the contacts, and f the corresponding forces' concatenation. We simplify notation by dropping explicit dependence on quantities.

In time-stepping approaches, *e.g.* [9], [24], Eq. (8) is discretized using a forward Euler approximation to obtain

$$M_i\,(v_{i+1} - v_i) = h(S\tau_i - H_i) + J_i^T \lambda_i,$$

where h is the time step size and λ_i corresponds to the concatenation of the contact impulses at time step i. These are projected in contact space

$$J_i\,(v_{i+1} - v_i) = J_i M_i^{-1}\left[h(S\tau_i - H_i) + J_i^T \lambda_i\right],$$

which can also be expressed as

$$c^+ = A\lambda + b + c^-, \tag{9}$$

with $c^+ = J_i v_{i+1}$, $c^- = J_i v_i$, $b = h J_i M_i^{-1}(S\tau_i - H_i)$, and $A = J_i M_i^{-1} J_i^T$.

Different contact models pose different conditions on what constraints accompany Eq. (9). In this work, the contact model defined in [9] is used because it is convex and analytically invertible. It penalizes movement in contact space by solving the following QCQP during the forward dynamics

$$\begin{aligned} \min_{\lambda} \quad & \tfrac{1}{2}\lambda^T(A + R)\lambda + \lambda^T(b + c^- + c^*) \\ \text{s.t.} \quad & \lambda \in \left\{\lambda \mid \lambda_{n(k)} \geq 0, \|\lambda_{t(k)}\| \leq \mu_k \lambda_{n(k)}\,,\ \forall k\right\}, \end{aligned} \tag{10}$$

where $\begin{bmatrix}\lambda_{t(k)} & \lambda_{n(k)}\end{bmatrix}^T$ are the tangential and normal components for the single contact impulse λ_k, R is a positive definite matrix that makes the solution unique and invertible, and c^* is a Baumgarte stabilization reference.

The inverse dynamics is well-defined and for a diagonal R we obtain an independent problem per contact

$$\begin{aligned} \min_{\lambda} \quad & \tfrac{1}{2}\lambda^T R\lambda + \lambda^T(c^+ + c^*). \\ \text{s.t.} \quad & \lambda \in \left\{\lambda \mid \lambda_{n(k)} \geq 0, \|\lambda_{t(k)}\| \leq \mu_k \lambda_{n(k)}\,,\ \forall k\right\}. \end{aligned} \tag{11}$$

IV. Implicit DDP

Our point of departure from the original DPP algorithm is the dynamics in (1b). Instead of the explicit dynamics, we assume dynamics of the form

$$g(x', x, u) = 0. \tag{12}$$

This will allow us to apply DDP for systems expressed via inverse dynamics, implicit or variational integrators, *etc.* Our focus will be contact dynamics, but we return to this later.

The goal is to compute the derivatives for the quadratic approximation of the Q-function (4). Terms related to the running cost l_i are trivial and will be omitted. Thus, we focus on the first and second-order sensitivity of the next step value function. A treatment of sensitivity analysis in the context of Newton methods can be found in [25].

A. First-Order Sensitivity Analysis

The first-order sensitivity of the value function in (2) is

$$V'_x = \frac{\partial V'}{\partial x} = \frac{\partial V'}{\partial x'}\frac{\partial x'}{\partial x} = V'_{x'}\frac{\partial x'}{\partial x}.$$

Here, V'_x is the sensitivity of the next step value function with respect to the current state, while $V'_{x'}$ is the sensitivity of the next step value function with respect to the next state; connected by the previous equation. Based on (12) we have

$$\frac{dg}{dx} = g_{x'}\frac{\partial x'}{\partial x} + g_x = 0 \Rightarrow \frac{\partial x'}{\partial x} = -g_{x'}^{-1}g_x, \tag{13}$$

where it is assumed that for any x and u, x' can be computed so that (12) is satisfied. Combining the previous two equations gives

$$V'_x = -V'_{x'}g_{x'}^{-1}g_x.$$

In practice, a faster computation can be achieved using the adjoint method [26] by computing first the quantity s by

$$s^T = V'_{x'}g_{x'}^{-1} \Rightarrow {V'_{x'}}^T = g_{x'}^T s$$

and then

$$V'_x = -s^T g_x. \tag{14a}$$

If we confine ourselves in a first-order analysis only this is computationally advantageous [26], but the computation of $\frac{\partial x'}{\partial x}$ in (13) is required for the second-order expansion. By a similar reasoning, V'_u is computed as

$$V'_u = -s^T g_u, \tag{14b}$$

which concludes our first-order analysis.

We now have all the ingredients for the first-order approximation of the Q-function. For example, the Q_x term in (4) is given by

$$Q_x = l_x - s^T g_x.$$

B. Second-Order Sensitivity Analysis

The second-order approximation of the value function is

$$V'_{xx} = {\frac{\partial x'}{\partial x}}^T V'_{x'x'}\frac{\partial x'}{\partial x} + V'_{x'}\frac{\partial^2 x'}{\partial x^2}.$$

The term $\frac{\partial^2 x'}{\partial x^2}$ constitutes a third-order tensor. We use matrix notation for the contractions but assume that their computation is clear from the context. It is computed as

$$\frac{d^2 g}{dx^2} = 0 \Rightarrow \frac{\partial^2 x'}{\partial x^2} = g_{x'}^{-1}\left({\frac{\partial x'}{\partial x}}^T g_{x'x'}\frac{\partial x'}{\partial x} + {\frac{\partial x'}{\partial x}}^T g_{x'x} + g_{xx'}\frac{\partial x'}{\partial x} + g_{xx}\right).$$

By combining the last two equations we have that

$$V'_{xx} = {\frac{\partial x'}{\partial x}}^T V'_{x'x'}\frac{\partial x'}{\partial x} - s^T\left({\frac{\partial x'}{\partial x}}^T g_{x'x'}\frac{\partial x'}{\partial x} + {\frac{\partial x'}{\partial x}}^T g_{x'x} + g_{xx'}\frac{\partial x'}{\partial x} + g_{xx}\right). \tag{15a}$$

For the remaining two terms in (4), a similar reasoning can be used to compute them as

$$V'_{xu} = {\frac{\partial x'}{\partial x}}^T V'_{x'x'}\frac{\partial x'}{\partial u} - s^T\left({\frac{\partial x'}{\partial x}}^T g_{x'x'}\frac{\partial x'}{\partial u} + {\frac{\partial x'}{\partial x}}^T g_{x'u} + g_{xx'}\frac{\partial x'}{\partial u} + g_{xu}\right), \tag{15b}$$

$$V'_{uu} = {\frac{\partial x'}{\partial u}}^T V'_{x'x'}\frac{\partial x'}{\partial u} - s^T\left({\frac{\partial x'}{\partial u}}^T g_{x'x'}\frac{\partial x'}{\partial u} + {\frac{\partial x'}{\partial u}}^T g_{x'u} + g_{ux'}\frac{\partial x'}{\partial u} + g_{uu}\right). \tag{15c}$$

This concludes the second-order sensitivity analysis. We can now compute all terms in (4). The rest of the DDP algorithm is implemented without changes.

It is worth pointing out that for the explicit dynamics (1b) we have that $g(x', x, u) = f(x, u) - x' = 0$, $g_{x'} = -I$, and $g_x = f_x$. Thus, $V'_x = V'_{x'}f_x$ as in (4). The same verification can be performed for the rest of the quantities.

C. Gauss-Newton Approximation

Especially for robot models with many degrees of freedom, computing the tensor terms (15) can be prohibitive expensive. Fortunately, it is possible to do a Gauss-Newton approximation of the Hessian—equivalent to iLQR—by ignoring them. Thus, the second-order sensitivity terms of the value function in an iLQR setting become

$$V'_{xx} = {\frac{\partial x'}{\partial x}}^T V'_{x'x'}\frac{\partial x'}{\partial x} \tag{16a}$$

$$V'_{xu} = {\frac{\partial x'}{\partial x}}^T V'_{x'x'}\frac{\partial x'}{\partial u} \tag{16b}$$

$$V'_{uu} = {\frac{\partial x'}{\partial u}}^T V'_{x'x'}\frac{\partial x'}{\partial u}. \tag{16c}$$

V. Acceleration-level Contact Dynamics

We describe here a contact resolution framework at the acceleration level rather than the commonly used velocity level. This way, we avoid the necessary first-order discretization of the dynamics. Other assumptions are not required about the robot's model (such as the assumption about a constant Jacobian in (9) that is inherent in the velocity-impulse formulations), without increasing the computation complexity. As such, we consider it

a superior choice. It is also the default choice in MuJoCo [27], which is a state-of-the-art robotics simulator.

Starting from the continuous time dynamics (8), we multiply both sides by JM^{-1} and add $\dot{J}v$, which gives

$$\underbrace{J\dot{v}+\dot{J}v}_{\alpha^+} = \underbrace{JM^{-1}J^T}_{A} f + \underbrace{JM^{-1}(S\tau - H)+\dot{J}v}_{\alpha^-}. \quad (17)$$

We can interpret this equation as follows: α^- is the unconstrained acceleration in contact space in the absence of any contacts, which is corrected by the term Af to result in the actual acceleration α^+ that satisfies the contact constraints.

As already explained, the contact model that we utilize was proposed in [9]. It computes the necessary contact forces by solving the following convex optimization problem that tries to minimize accelerations in contact space

$$\begin{aligned} \min_{f} \quad & \tfrac{1}{2} f^T (A+R) f + f^T(\alpha^- - \alpha^*) \\ \text{s.t.} \quad & f \in \left\{ f \mid f_{n(k)} \geq 0, \|f_{t(k)}\| \leq \mu_k f_{n(k)}, \forall k \right\}, \end{aligned} \quad (18)$$

which is the equivalent to (10) for accelerations.

While the bias accelerations α^* can be in a general Baumgarte stabilization form, a choice that works reasonably good across models is

$$\alpha^* = \dot{J}v - \tfrac{1}{h^2}\phi(q) - \tfrac{1}{h} Jv, \quad (19)$$

with $\phi(q)$ the gap distance, positive when bodies are separate. The first term is used to cancel the same term in α^+ and α^- and simplify computations. The second and third term are obtained by a Taylor expansion of the gap distance function and ignoring third and higher-order terms.

In the forward pass, the above optimization problem is solved for the contact forces using a standard Projected Gauss–Siedel solver [24]. Though in principle this can be implemented in the backward pass, the computation of the gradients becomes more complicated since we have to differentiate a numeric solution. Even with automatic differentiation, the quality of the gradients can suffer. Instead, a diagonal approximation of the system is assumed and an approximate solution to the contact forces is computed [5]. The implicit formulation avoids this issue and the exact solution for the contact forces is given in a closed form.

By utilizing the implicit framework and the invertibility of the model, problem (11) is expressed in acceleration space

$$\begin{aligned} \min_{f} \quad & \tfrac{1}{2} f^T R f + f^T(\alpha^+ - \alpha^*) \\ \text{s.t.} \quad & f \in \left\{ f \mid f_{n(k)} \geq 0, \|f_{t(k)}\| \leq \mu_k f_{n(k)}, \forall k \right\}. \end{aligned} \quad (20)$$

For the computation of α^+ as given by (17), the joint acceleration $\dot{v}$ is required. In the classical DDP algorithm this is not available, since we only have access to the current state q and v, and the acceleration is computed after the contact forces. In the implicit form, since we have additionally available the next state x', the computation of the acceleration is possible. Thus, we can compute each contact force in closed form as

$$f_i = P_\mu\{-R^{-1}(\alpha^+ - \alpha^*)\}. \quad (21)$$

Algorithm 1: Forward pass with contacts.

Input: x, k_i, K_i, R, and $\mu_k, \forall k$.
Output: x' and f.

1 Compute $A+R$ and $\alpha^- - \alpha^*$ based on Eqs. (17) and (19).
2 Solve Eq. (18) for the contact forces f.
3 Solve Eq. (12) together with (7b) for the next state x'.

Algorithm 2: Backward pass with contacts.

Input: x', x, u, R, and $\mu_k, \forall k$.
Output: k_i and K_i.

1 Compute α^+, α^*, and f from Eqs. (17), (19) and (21).
2 Differentiate Eq. (12) using the computed contact forces f to obtain the required expansion terms of g.
3 Compute the value function terms in (14) and (15).
4 Compute the Q-function terms in Eq. (4).
5 Compute the gains k_i and K_i in Eq. (5), and the current value function terms in (6) for the next step $i-1$.

P_μ projects contact forces to the cone with coefficient μ [24].

After the computation of the contact forces, we can enforce the implicit dynamics Eq. (12) either using a forward or inverse dynamics formulation. Given the available information, the computation of inverse dynamics is cheaper and numerically superior [28]. Furthermore, this decoupling between the forward and backward pass allows us to avoid the rootfinding problem during the forward, that would be necessary for a fully implicit implementation. Having to solve the rootfinding problem in the forward pass increases the computation time of the implicit formulation. We summarize the DDP computations subject to contacts in Algos. 1 and 2.

VI. Results

A. Implementation Details

For the computation of the rigid-body dynamics, the Julia library `RigidBodyDynamics.jl` is used [29]. Computation of first-, second- and third-order tensors is done using forward-mode automatic differentiation [30].

We begin by performing multiple comparisons between implicit and explicit DDP formulations for a double pendulum swing-up task. Next, we present two problems that require multi-contact motion planning: A single leg that is required a) to stand up from the ground, and b) to balance from an initial random state.

B. Aggregate Double Pendulum Swing-up

For the double pendulum swing-up task, we generate 100 random trials (that is, with random initial state) and we specify an objective that includes a desired upright posture at the end of a $T = 5$s horizon, with a time step of 10ms, while penalizing joint torques at intermediate states. Additionally, joint limits

The accompanying code is available at github.com/ichatzinikolaidis/iDDP and the video at youtu.be/w8oOPqo6oC0.

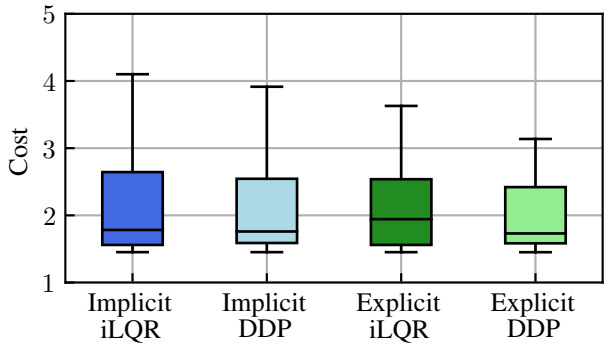

Fig. 2. Aggregate results for the total trajectory cost of each variant.

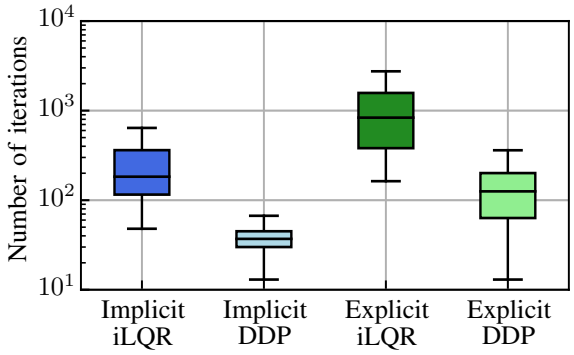

Fig. 3. Aggregate results for the total number of iterations of each variant.

are modelled using unilateral forces at the joints. Only the unilateral constraint is imposed (forces push the joint away from the limit), while friction is not required.

We compare four variants of the methods presented in this work:

- Implicit iLQR with backward Euler dynamics.
- Implicit DDP with backward Euler dynamics.
- Explicit iLQR with forward Euler dynamics.
- Explicit DDP with forward Euler dynamics.

For every random initialization, the four variants are executed until convergence (or until an upper iteration limit is reached) and the number of iterations and total cost of the trajectory is logged. Aggregate box plot results for the cost and the number of iterations are shown in Figs. 2 and 3, respectively.

From the comparison, the implicit formulations result in considerably fewer iterations than the explicit counterparts. Both median, minimum and maximum values, and the rest of the statistical properties in Fig. 3 are improved with an implicit formulation regarding the number of iterations. As expected, the trade-off for this is the larger in general cost of the resulting trajectory in Fig. 2. This can be partially explained by the fact that since the explicit formulations perform on average more iterations, they are capable to fine-tune the resulting trajectory more. But given the considerable fewer iterations for the implicit formulations, this aspect is more important in terms of the overall performance.

A possible reason behind this is the integrator's properties. Implicit Euler is an A-stable method suitable even for stiff systems. As such, it usually exhibits energy decrease—instead of the common increase in explicit methods—that makes the whole formulation more stable.

C. Single Double Pendulum Swing-up

1) Cost per iteration and timings: We evaluate the cost per iteration for one double pendulum swing-up and compare 6 different formulations (each with a DDP and iLQR variant):

(i) Forward Euler dynamics in the forward and backward passes.
(ii) Forward Euler dynamics in the forward pass, and forward Euler inverse dynamics in the backward pass.
(iii) Backward Euler dynamics in the forward and backward passes.

We use the same duration and time step as in the previous case but with a different cost function, without joint limits, and initialize at the stable equilibrium. The results are shown in Fig. 4. Formulation (i) corresponds to a classical iLQR/DDP with explicit dynamics. Formulation (ii) is enabled by the presented framework. The computation of the Jacobian and tensor terms is based on the automatic differentiation of the inverse dynamics. Since (ii) is equivalent to (i), the solutions by the two approaches are exactly the same and are plotted together in Fig. 4. Differences are found in the computation time, as reported next. Formulation (iii) is implicit in both passes, enabled by the presented framework.

In terms of computation, formulations (i) and (ii) with iLQR require 126 iterations, while with DDP require 55 iterations. In terms of timings, the mean time of each iteration for (i) with iLQR is 5.87ms and with DDP 29.91ms. For (ii) with iLQR is 5.03ms and with DDP 28.49ms. While the differences are not significant for such a low-dimensional model, these can become starker for robot models with larger degrees of freedom. For (iii), 75 iterations for iLQR and 40 iterations with DDP. The mean computation time of each iterations with iLQR is 7.19ms and with DDP 72.22ms. The increased computation is due to the solution of a nonlinear system of equations in the forward pass.

2) Effect of time step size: We focus now on the effect of the time step size to the solution of the problem. We solve the same problem as before for multiple time step selections and report the number of iterations required until convergence. Since formulations (i) and (ii) are equivalent, we focus the comparison on (i) and (iii). We solve them using iLQR but similar conclusions could be drawn if DDP was used.

The results are shown on Table I. For small time steps, the two formulations are essentially equivalent and, thus, require the same number of iterations. As the time step increases, the influence of the integrator's damping in (iii) becomes more apparent. This results in a desirable decrease in the number of iterations for convergence. The motions are included in the accompanying video. For larger time steps, the accuracy of both first-order integrators worsens significantly.

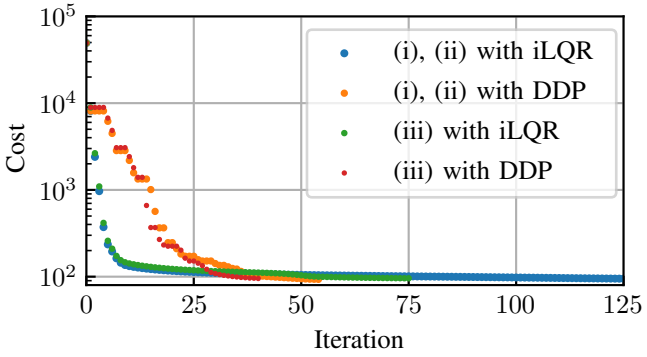

Fig. 4. Cost per iteration for the different formulations.

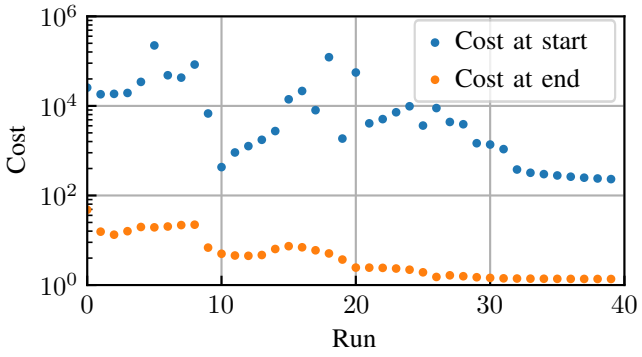

Fig. 5. Trajectory cost at each run of the receding horizon formulation.

TABLE I
EFFECT OF TIME STEP ON NO. OF ITERATIONS UNTIL CONVERGENCE

Time step	10^{-4}	10^{-3}	10^{-2}
(i) / (ii)	56	68	126
(iii)	56	66	75

D. Multi-contact Stand-up

Next, we consider a planar 3 degree-of-freedom single leg of a humanoid robot and the task now is to stand-up upright from the ground. The model can make multiple contacts with the terrain using all the bodies of its structure, but self-collisions are inactive. We pre-define a number of possible contact points but we do not prescribe the contact activation pattern. Adding a contact detection mechanism and avoiding the pre-specification of contacts is another possibility, as typically done in simulation engines.

The cost function of the problem is defined as

$$J = w_{q_f}||q_f - q_g||^2 + w_{v_f}||v_f||^2 + \sum_i (w_\tau||\tau_i||^2 + w_v||v_i||^2).$$

A penalization of the velocity and joint torque is applied throughout the trajectory, while a goal state is defined in the final cost term. The motion duration is $T = 4$s with a time step of 10ms; this is a relatively large time step for contacts, but our aim here is to output an approximate contact-rich motion plan. Given this plan as input, it is possible to post-process it to increase the quality.

The friction coefficient is selected as $\mu = 0.7$. Parameter R is initialized with a value of 1 for all components. While in principle it can take arbitrary values, we can test the validity from a numerical viewpoint as follows [27]: We run the forward and backward pass separately and compare the computed forces. The two solutions should match according to the desired numerical precision.

The main difficulty is that the problem exhibits a number of contact possibilities. Thus, mode enumeration can be very challenging. Notice also how delicate heel balance emerges while reaching the upright configuration. Our trajectory optimization framework is capable of outputting a locally optimal motion plan. Even though a zero torque initial solution is used here, its quality greatly affects the quality of the computed motion. Finally, by changing the terms in the cost function, it is possible to obtain different solutions, *e.g.* more conservative but with higher torque cost.

The resulting motion can be found in the accompanying video. There is an initial explosive and dynamic motion at about 1s. Such a motion would be in practice difficult to track. Yet being able to compute such a complex motion from high-level input only demonstrates the power of DDP-based approaches. There are a couple of ways to mitigate that: an obvious approach is to increase the torque, position, and velocity penalizations accordingly. Another option is to include terms that penalize the rate of the commanded torques. Finally, a more principled approach is to penalize high-frequency components of the signals involved [31].

E. Multi-contact Balancing

Using the same model as before, the state now is randomly initialized in the air. The task is to keep the initial posture with zero velocity, *i.e.* to balance. In contrast to the previous case, this problem is formulated in a receding horizon fashion. A fixed number of 15 iterations for DDP is pre-specified; this makes real-time iterations of the algorithm possible. The horizon length is $T = 0.5$s, with the simulation running at 200Hz, while our framework runs at 20Hz. The structure of the cost function remains the same as before, albeit the weight regarding the final velocity is increased to bias more towards a static final configuration.

The computed motion naturally performs a series of jumps to dissipate kinetic energy and come to a complete stop. The underactuated foot tilting emerged as the outcome of optimization without the need for programming explicit controllers as in [32]. Compared to the case in the previous section, the receding horizon formulation is capable of producing better motions in general. This is because the constant updates allow it to escape iterations with a very small cost decrease, which can be common in the fixed horizon optimization of the previous case. If a bad initialization is specified or the horizon and frequency are not chosen properly, the receding horizon formulation can be trapped too. The selection of these parameters depends on the desired task and the initial state.

Finally, a semi-log plot of the total trajectory cost at the beginning and at the end of each DDP step is shown in Fig. 5.

We notice that in about 20 runs a successful balancing motion is computed. Afterwards, each run rapidly converges to this motion. The reason why the cost is increased at the beginning of each run is because the horizon moves; the predicted trajectory for the new segment at the end of the previous horizon is that the robot will essentially fall, which incurs a large cost. Further, during the initial runs, the motion is highly unstable and a suitable balancing motion is not discovered yet. Thus, the total trajectory cost varies greatly between consecutive runs.

VII. Conclusion

This work presented an application of DDP suited for systems with implicitly defined dynamics that can handle dynamical interactions, with a particular focus on through-contact motion planning. This allowed extending the original DDP to a larger class of dynamics models, *e.g.* models based on inverse dynamics. We described how to use the implicit formulation for accurate contact resolution in the DDP framework without requiring approximations of contact dynamics. The proposed method is exact and straightforward to implement, utilizing a closed-form solution for quality gradient computations. We demonstrated its properties in a number of cases: comparisons of implicit and explicit dynamics for a double pendulum, and two case studies for a single leg model that required challenging multi-contact motion plans.

While the original DDP provides both feedforward and feedback gains that guarantee a level of robustness against small perturbations, we noticed that the computed motion plans can sometimes fail if the conditions of the problem change slightly. Though one can introduce robustness as part of the trajectory optimization modelling, we believe that running the framework in a receding horizon fashion is more appropriate and promising. Thus, the motion plans should be updated online to withstand unexpected perturbations.

It is worth noting that DDP simulates the dynamics of the system and activates a contact point if appropriate. Thus, contacts are taken into account according to the system's natural dynamics [23], which may lead to abrupt motions [5]. Being a shooting method for unconstrained systems, DDP is limited in terms of active search for potential contacts. Further improvements can be made by combinatorial planning and exploration, where transcription-based methods demonstrated better capabilities and flexibility [33], [34], although requiring additional and non-negligible computation cost in practice.

Acknowledgements

This work was supported by the Engineering and Physical Sciences Research Council as part of the Centre for Doctoral Training in Robotics and Autonomous Systems (EP/L016834/1). The authors are with the School of Informatics and the Edinburgh Centre for Robotics, University of Edinburgh, Edinburgh, UK. Emails: iordanis.cs@gmail.com; zhibin.li@ed.ac.uk

References

[1] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Rev.*, vol. 59, no. 4, pp. 849–904, 2017.

[2] K. Yuan *et al.*, "Bayesian optimization for whole-body control of high-degree-of-freedom robots through reduction of dimensionality," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2268–2275, 2019.

[3] E. Todorov, "Goal directed dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 2994–3000.

[4] I. Chatzinikolaidis *et al.*, "Nonlinear optimization using discrete variational mechanics for dynamic maneuvers of a 3D one-leg hopper," in *Proc. IEEE Int. Conf. Humanoid Robots*, 2018, pp. 932–937.

[5] Y. Tassa *et al.*, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.

[6] M. Neunert *et al.*, "Trajectory optimization through contacts and automatic gait discovery for quadrupeds," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1502–1509, 2017.

[7] J. Carius *et al.*, "Trajectory optimization for legged robots with slipping motions," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 3013–3020, 2019.

[8] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II*. Springer, 1996.

[9] E. Todorov, "Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 6054–6061.

[10] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *Int. J. Control*, vol. 3, no. 1, pp. 85–95, 1966.

[11] R. Bellman and S. Dreyfus, *Applied Dynamic Programming*. Princeton University Press, 1962.

[12] L. Liao and C. Shoemaker, "Advantages of differential dynamic programming over Newton's method for discrete-time optimal control problems," Cornell University, Tech. Rep., 1992. [Online]. Available: https://ecommons.cornell.edu/handle/1813/5474

[13] E. Todorov and W. Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proc. Am. Control Conf.*, 2005, pp. 300–306.

[14] Y. Tassa *et al.*, "Control-limited differential dynamic programming," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 1168–1175.

[15] Z. Xie *et al.*, "Differential dynamic programming with nonlinear constraints," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 695–702.

[16] M. Geisert *et al.*, "Regularized hierarchical differential dynamic programming," *IEEE Trans. Robot.*, vol. 33, no. 4, pp. 819–833, 2017.

[17] R. Budhiraja *et al.*, "Differential dynamic programming for multi-phase rigid contact dynamics," in *Proc. IEEE Int. Conf. Humanoid Robots*, 2018, pp. 1–9.

[18] H. Li and P. Wensing, "Hybrid systems differential dynamic programming for whole-body motion planning of legged robots," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5448–5455, 2020.

[19] G. Boutselis and E. Theodorou, "Discrete-time differential dynamic programming on Lie groups: Derivation, convergence analysis and numerical results," *IEEE Trans. Autom. Control*, pp. 1–1, 2020.

[20] J. Carius *et al.*, "Trajectory optimization with implicit hard contacts," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3316–3323, 2018.

[21] C. Mastalli *et al.*, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 2536–2542.

[22] M. Giftthaler *et al.*, "A family of iterative Gauss-Newton shooting methods for nonlinear optimal control," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.

[23] M. Posa *et al.*, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robotics Res.*, vol. 33, no. 1, pp. 69–81, 2014.

[24] P. Horak and J. Trinkle, "On the similarities and differences among contact models in robot simulation," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 493–499, 2019.

[25] S. Zimmermann *et al.*, "PuppetMaster: Robotic animation of marionettes," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 103:1–103:11, 2019.

[26] G. Strang, *Computational Science and Engineering*. Wellesley-Cambridge Press, 2007.

[27] Roboti LLC. (2020) MuJoCo computation. [Online]. Available: http://www.mujoco.org/book/computation.html

[28] J. M. Hollerbach, "A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity," *IEEE Trans. Syst., Man, Cybern.*, vol. 10, no. 11, pp. 730–736, 1980.
[29] T. Koolen and R. Deits, "Julia for robotics: Simulation and real-time control in a high-level programming language," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 604–611.
[30] J. Revels *et al.*, "Forward-mode automatic differentiation in Julia," *arXiv: 1607.07892 [cs.MS]*, 2016.
[31] R. Grandia *et al.*, "Frequency-aware model predictive control," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1517–1524, 2019.
[32] Z. Li *et al.*, "Humanoid balancing behavior featured by underactuated foot motion," *IEEE Trans. Rob.*, vol. 33, no. 2, pp. 298–312, 2017.
[33] I. Chatzinikolaidis *et al.*, "Contact-implicit trajectory optimization using an analytically solvable contact model for locomotion on variable ground," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6357–6364, 2020.
[34] A. Patel *et al.*, "Contact-implicit trajectory optimization using orthogonal collocation," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2242–2249, 2019.