# Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control

Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Sethu Vijayakumar, Nicolas Mansard

# Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control

Carlos Mastalli [1] Rohan Budhiraja [1] Wolfgang Merkt [2] Guilhem Saurel [1]
Bilal Hammoud [3] Maximilien Naveau [3] Justin Carpentier [4] Sethu Vijayakumar [2] Nicolas Mansard [1]

***Abstract*—We introduce Crocoddyl (Contact RObot COntrol by Differential DYnamic Library), an open-source framework tailored for efficient multi-contact optimal control. Crocoddyl efficiently computes the state trajectory and the control policy for a given predefined sequence of contacts. Its efficiency is due to the use of sparse analytical derivatives, exploitation of the problem structure, and data sharing. It employs differential geometry to properly describe the state of any geometrical system, e.g. floating-base systems. We have unified dynamics, costs, and constraints into a single concept — action — for greater efficiency and easy prototyping. Additionally, we propose a novel multiple-shooting method called Feasibility-prone Differential Dynamic Programming (FDDP). Our novel method shows a greater globalization strategy compared to classical Differential Dynamic Programming (DDP) algorithms, and it has similar numerical behavior to state-of-the-art multiple-shooting methods. However, our method does not increase the computational complexity typically encountered by adding extra variables to describe the gaps in the dynamics. Concretely, we propose two modifications to the classical DDP algorithm. First, the backward pass accepts infeasible state-control trajectories. Second, the rollout keeps the gaps open during the early "exploratory" iterations (as expected in multiple-shooting methods). We showcase the performance of our framework using different tasks. With our method, we can compute highly-dynamic maneuvers for legged robots (e.g. jumping, front-flip) in the order of milliseconds.**

## I. Introduction

Multi-contact optimal control promises to generate whole-body motions and control policies that allow legged robots to robustly react to unexpected events in real-time. It has several advantages compared with state-of-the-art approaches in which a whole-body controller compliantly tracks an optimized centroidal trajectory [1], [2], [3], [4], [5], [6]. For instance, they cannot properly handle the angular momentum produced by the extremities of the limbs. In other words, the angular momentum conservation in a multibody system creates nonholonomic constraints on the dynamics [7], and it is well-known that instantaneous time-invariant control cannot properly track these systems. Indeed, in our previous work [8], we have shown these advantages by producing more efficient motions, with lower forces and impacts.

Recent work on optimal control have shown that nonlinear Model Predictive Control (MPC) is plausible for controlling legged robots in real-time [9], [10], [11]. All these methods have in common that they solve the nonlinear Optimal Control (OC) problem by iteratively building and solving a Linear-Quadratic Regulator (LQR) problem. These frameworks use numerical or automatic differentiation which is inefficient compared to sparse and analytical derivatives [12]. Furthermore, they do not handle geometrical systems, typically found in legged systems as the floating-base is a $\mathbb{SE}(3)$ element. DDP has proven to efficiently solve nonlinear MPC problems due to its intrinsic sparse structure. However, it has poor globalization strategy and struggles to handle infeasible warm-start. In this vein, Giftthaler et al. [13] proposed a variant of the DDP algorithm for multiple-shooting OC, which has a better convergence rate than DDP. Nonetheless, it does not match exactly the numerical behavior of classical multiple-shooting methods [14]. In this work, we address these drawbacks found in the literature with our framework called Crocoddyl (Fig. 1).

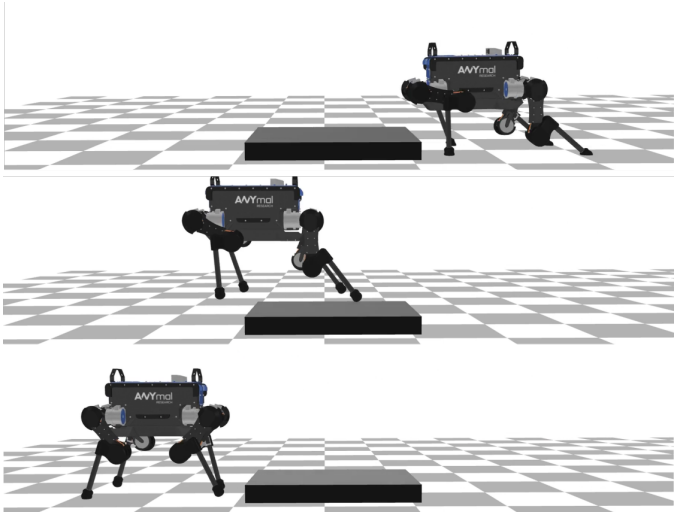

Fig. 1. Crocoddyl: an efficient and versatile framework for multi-contact optimal control. Highly-dynamic maneuvers needed to traverse an obstacle with the ANYmal robot.

### A. Contribution

We propose a novel and efficient framework for multi-contact OC called Crocoddyl. Our framework efficiently solves this problem by employing sparse and analytical derivatives of the contact and impulse dynamics. It also handles any kind of geometrical system by employing a dedicated algebra and its derivatives in our optimal control solver. Indeed, we model the floating-base as a $\mathbb{SE}(3)$ element, needed for example for the generation of front-flip motions. Additionally, we propose a

[1] Gepetto Team, LAAS-CNRS, Toulouse, France.
[2] The Institute of Perception, Action, and Behaviour, School of Informatics, University of Edinburgh, Edinburgh, UK
[3] Max Plank Institute for Intelligent Systems, Tuebingen, Germany.
[4] INRIA, France.
*email*: carlos.mastalli@laas.fr.

variant of the DDP algorithm that exactly matches the behavior of direct multiple-shooting methods. Our algorithm is called Feasibility-prone Differential Dynamic Programming (FDDP) as it handles infeasible guesses that occur whenever there is a gap between subsequent nodes in the trajectory. FDDP has a greater globalization strategy compared to classical DDP, allowing us to solve very complex maneuvers in few iterations and milliseconds. Our framework exceeds the computational performance and the convergence rate of state-of-the-art methods [10], [11], [13].

## II. Multi-contact Optimal Control

In this section, we first introduce the multi-contact optimal control problem for multibody systems under any physical constraints (Section II-A). This problem can be seen as a bilevel optimization problem, where the lower-level formulates the system dynamics based on the Gauss principle of least constraint. However, this problem is hard to solve in real-time, and there is no clear way of doing it. For that, we simplify the problem by modeling the contacts as holonomic constraints (Section II-B). With this method, we derive tailored analytical and sparse derivatives for fast computation. The calculation of derivatives typically represents the main computation carried out by optimal control solvers.

### A. Formulation of the optimal control problem

We focus on an efficient formulation of the multi-contact optimal control problem. One can formulate this problem as follows:

$$\begin{Bmatrix} \mathbf{x}_0^*, \cdots, \mathbf{x}_N^* \\ \mathbf{u}_0^*, \cdots, \mathbf{u}_{N-1}^* \end{Bmatrix} = \arg\min_{\mathbf{X},\mathbf{U}} l_N(\mathbf{x}_N) + \sum_{k=0}^{N-1} \int_{t_k}^{t_k+\Delta t_k} l(\mathbf{x},\mathbf{u}) dt$$
$$\text{s.t.} \quad \dot{\mathbf{v}}, \boldsymbol{\lambda} = \arg\min_{\dot{\mathbf{v}},\boldsymbol{\lambda}} \|\dot{\mathbf{v}} - \dot{\mathbf{v}}_{free}\|_{\mathbf{M}},$$
$$\mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U} \tag{1}$$

where the state $\mathbf{x} = (\mathbf{q}, \mathbf{v}) \in X$ lies in a differential manifold formed by the configuration point $\mathbf{q}$ and its tangent vector $\mathbf{v}$ and is described by a $n_x$-tuple, the control $\mathbf{u} = (\boldsymbol{\tau}, \boldsymbol{\lambda}) \in \mathbb{R}^{n_u}$ composed by the input torque commands $\boldsymbol{\tau}$ and contact forces $\boldsymbol{\lambda}$, $\dot{\mathbf{x}} \in T_{\mathbf{x}}X$ lies in the tangent space of the state manifold and it is described by a $n_{dx}$-tuple, and $\mathcal{X}$, $\mathcal{U}$ represent the state and control admissible sets, respectively, $\dot{\mathbf{v}}_{free}$ is the unconstrained acceleration in generalized coordinates, and $\mathbf{M}$ is the joint-space inertia matrix.

This problem can be seen as a bilevel optimization where the lower-level optimization uses the Gauss principle of least constraint to describe the physical constraints as described in [15]. State and control admissible sets can belong to the lower-level optimization (e.g., joint limits and force friction constraints) as well as to the upper-level one (e.g., task-related constraints and collision with the environment).

### B. Contacts as holonomic constraints

To solve this optimization problem in real-time, we need to efficiently handle (a) the high-dimensionality of the search-space and (b) the instabilities, discontinuities, and non-convexity of the system dynamics (lower-level optimization), among others. One way of reducing the complexity of the problem is by solving the lower-level optimization analytically. In our previous work [16], we have derived the contact model using holonomic rheonomic constraints on the frame placement, (i.e. $\boldsymbol{\phi}(\mathbf{q}) = \mathbf{0}$ where $\mathbf{J}_c = \frac{\partial \boldsymbol{\phi}}{\partial \mathbf{q}}$ is the contact Jacobian) as:

$$\begin{bmatrix} \dot{\mathbf{v}} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{M} & \mathbf{J}_c^\top \\ \mathbf{J}_c & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\tau}_b \\ -\mathbf{a}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{y}(\mathbf{x},\boldsymbol{\tau}) \\ -\mathbf{g}(\mathbf{x},\boldsymbol{\tau}) \end{bmatrix}, \tag{2}$$

where $\mathbf{J}_c$ is expressed in the local frame, and $\mathbf{a}_0 \in \mathbb{R}^{n_f}$ is the desired acceleration in the constraint space. Eq. (2) allows us to express the contact forces in terms of the state and torques, and this problem has a unique solution if $\mathbf{J}_c$ is full-rank. To improve stability in the numerical integration, we define PD gains that are similar in spirit to Baumgarte stabilization [17]:

$$\mathbf{a}_0 = \mathbf{a}_{\lambda(c)} - \alpha\, {}^oM^{ref}_{\lambda(c)} \ominus {}^oM_{\lambda(c)} - \beta \mathbf{v}_{\lambda(c)}, \tag{3}$$

where $\mathbf{v}_{\lambda(c)}$, $\mathbf{a}_{\lambda(c)}$ are the spatial velocity and acceleration at the parent body of the contact $\lambda(c)$, respectively, $\alpha$ and $\beta$ are the stabilization gains, and ${}^oM^{ref}_{\lambda(c)} \ominus {}^oM_{\lambda(c)}$ is the $\mathbb{SE}(3)$ inverse composition between the reference contact placement and the current one [18]. We use the logarithmic map to transform elements of $\mathbb{SE}(3)$ to their Lie algebra.

If we neglect the friction-cone constraints or the joint limit inequality constraints, then the problem only has the equality constraint for the forward dynamics under rigid contacts [16]. This kind of problem can easily be solved through DDP, i.e., by solving a set of smaller and simpler unconstrained Quadratic Programming (QP) subproblems [19].

*1) Efficient rollout and derivative computation:* We do not need to invert the entire Karush-Kuhn-Tucker (KKT) matrix, in Eq. (2), during the numerical integration of the dynamics. Indeed, the evolution of the system acceleration and contact can be described as:

$$\begin{aligned} \mathbf{y}(\mathbf{x},\boldsymbol{\tau}) &= \mathbf{M}^{-1}\left(\boldsymbol{\tau}_b + \mathbf{J}_c^\top \mathbf{g}(\mathbf{x},\boldsymbol{\tau})\right), \\ \mathbf{g}(\mathbf{x},\boldsymbol{\tau}) &= \widehat{\mathbf{M}}^{-1}(\mathbf{a}_0 - \mathbf{J}_c\mathbf{M}^{-1}\boldsymbol{\tau}_b), \end{aligned} \tag{4}$$

and, for instance, we can use the Cholesky decomposition for efficiently computing $\mathbf{M}^{-1}$ and $\widehat{\mathbf{M}}^{-1} = \left(\mathbf{J}_c\mathbf{M}^{-1}\mathbf{J}_c^\top\right)^{-1}$. We can interpret $\widehat{\mathbf{M}}$ as the apparent inertial matrix which lies in contact space, and it is usually called operational space inertia matrix [20]. For these operations, we exploit the structure of the inertial matrix and contact Jacobian using Pinocchio, for further detail see [12].

If we analytically derive Eq. (2) by applying the chain rule, then we can describe the Jacobians of $\mathbf{y}(\cdot)$ and $\mathbf{g}(\cdot)$ with respect to the derivatives of the Recursive Newton-Euler Algorithm (RNEA) algorithm and kinematics, i.e.:

$$\begin{aligned} \begin{bmatrix} \delta\dot{\mathbf{v}} \\ -\delta\boldsymbol{\lambda} \end{bmatrix} &= -\begin{bmatrix} \mathbf{M} & \mathbf{J}_c^\top \\ \mathbf{J}_c & \mathbf{0} \end{bmatrix}^{-1} \left( \begin{bmatrix} \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{a}_0}{\partial \mathbf{x}} \end{bmatrix} \delta\mathbf{x} + \begin{bmatrix} \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{a}_0}{\partial \mathbf{u}} \end{bmatrix} \delta\mathbf{u} \right) \\ &= \begin{bmatrix} \mathbf{y}_\mathbf{x} \\ -\mathbf{g}_\mathbf{x} \end{bmatrix} \delta\mathbf{x} + \begin{bmatrix} \mathbf{y}_\mathbf{u} \\ -\mathbf{g}_\mathbf{u} \end{bmatrix} \delta\mathbf{u}, \end{aligned} \tag{5}$$

where $\frac{\partial \boldsymbol{\tau}}{\partial \mathbf{x}}$, $\frac{\partial \boldsymbol{\tau}}{\partial \mathbf{u}}$ are the RNEA derivatives, and $\frac{\partial \mathbf{a}_0}{\partial \mathbf{x}}$, $\frac{\partial \mathbf{a}_0}{\partial \mathbf{u}}$ are the kinematics derivatives of the frame acceleration [21]. We employ an efficient method to invert this blockwise matrix.

Using the LDU decomposition, we can write this matrix inversion as:

$$\mathbf{L}^{-1} = \begin{bmatrix} \mathbf{M}^{-1} - \mathbf{M}^{-1}\mathbf{J}_c^\top \widehat{\mathbf{M}}^{-1}\mathbf{J}_c\mathbf{M}^{-1} & \mathbf{M}^{-1}\mathbf{J}_c^\top \widehat{\mathbf{M}}^{-1} \\ \widehat{\mathbf{M}}^{-1}\mathbf{J}_c\mathbf{M}^{-1} & -\widehat{\mathbf{M}}^{-1} \end{bmatrix} \quad (6)$$

where we reuse the computation of the inertia matrix and apparent inertia matrix inversions computed during the rollout of the dynamics. In case that $\mathbf{J}_c$ is not full-rank, then we damp the Cholesky decomposition of $\widehat{\mathbf{M}}$. The damping value can be carefully hand-tuned, depending on the contact configuration, for each node of the OC problem. For instance, we use $10^{-12}$ as a damping factor in many of our results.

*2) Impulse dynamics:* Transitions from non-contact to contact condition are called contact gain in the literature [22]. In these instantaneous phases, the system encounters very high forces acting instantaneously (i.e., impulses). We can similarly describe the impulse dynamics of multibody system as:

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}_c^\top \\ \mathbf{J}_c & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v}^+ \\ -\mathbf{\Lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{M}\mathbf{v}^- \\ -e\mathbf{J}_c\mathbf{v}^- \end{bmatrix}, \quad (7)$$

where $e \in [0, 1]$ is the restitution coefficient that considers compression / expansion, $\mathbf{\Lambda}$ is the contact impulse and, $\mathbf{v}^-$ and $\mathbf{v}^+$ are the discontinuous changes in the generalized velocity (i.e., velocity before and after impact, respectively). Perfect inelastic collision produces a contact velocity equal to zero, i.e., $e = 0$. Similarly, we use the Cholesky decomposition to efficiently compute the impulse dynamics and its derivatives.

## III. Feasibility-prone Differential Dynamic Programming

In this section, we describe our novel solver for multiple-shooting OC called Feasibility-prone Differential Dynamic Programming (FDDP). First, we introduce a brief description of the DDP algorithm (Section III-A). Then, we analyze the numerical behavior of classical multiple-shooting methods (Section III-B). With this in mind, we propose a modification of the forward and the backward passes in Section III-C and III-D, respectively. Our method entirely matches the numerical behavior of multiple-shooting algorithm in contrast to [13]. Finally, we propose a new model for the expected reduction cost and line-search procedure based on the Goldstein condition (Section III-E).

### A. Differential dynamic programming

DDP belongs to the family of OC and indirect trajectory optimization methods [19]. It locally approximates the optimal flow (i.e., the Value function) as

$$V_k(\delta\mathbf{x}_k) = \min_{\delta\mathbf{u}_k} l_k(\delta\mathbf{x}_k, \delta\mathbf{u}_k) + V_{k+1}(\mathbf{f}_k(\delta\mathbf{x}_k, \delta\mathbf{u}_k)), \quad (8)$$

which breaks the OC problem into a sequence of simpler subproblems by using "Bellman's principle of optimality", i.e.:

$$\delta\mathbf{u}_k^*(\delta\mathbf{x}_k) = \arg\min_{\delta\mathbf{u}_k} \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}^T \begin{bmatrix} 0 & \mathbf{Q}_{\mathbf{x}_k}^T & \mathbf{Q}_{\mathbf{u}_k}^T \\ \mathbf{Q}_{\mathbf{x}_k} & \mathbf{Q}_{\mathbf{x}\mathbf{x}_k} & \mathbf{Q}_{\mathbf{x}\mathbf{u}_k} \\ \mathbf{Q}_{\mathbf{u}_k} & \mathbf{Q}_{\mathbf{x}\mathbf{u}_k}^T & \mathbf{Q}_{\mathbf{u}\mathbf{u}_k} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}. \quad (9)$$

Note that $l_k(\cdot)$, $\mathbf{f}_k(\cdot)$ are the Linear Quadratic (LQ) approximation of the cost and dynamics functions, respectively; $\delta\mathbf{x}_k$, $\delta\mathbf{u}_k$ reflects the fact that we linearize the problem around a guess $(\mathbf{x}_k^i, \mathbf{u}_k^i)$. This remark is particularly important (1) to understand our FDDP algorithm and (2) to deal with geometrical systems,[1] for instance, formed by the $\mathbb{SE}(3)$ element of the floating-base.

The $\mathbf{Q}$ terms represent the LQ approximation of the Hamiltonian function. The solution of the entire OC problem is computed through the Riccati recursion formed by sequentially solving Eq. (9). This procedure provides the feed-forward term $\mathbf{k}_k$ and feedback gains $\mathbf{K}_k$ at each discretization point $k$.

### B. The role of gaps in multiple-shooting

The multiple-shooting OC formulation introduces intermediate states $\mathbf{x}_k$ (i.e., shooting nodes) as additional decision variables to the numerical optimization problem [14]. The difference between the rollout state and the shooting state forms a gap in the dynamics:

$$\bar{\mathbf{f}}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1}, \quad (10)$$

where $\bar{\mathbf{f}}_{k+1}$ represents the gap in the dynamics, $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$ is the rollout state at interval $k+1$, and $\mathbf{x}_{k+1}$ is the next shooting state (decision variable). For the remainder of this letter, we assume that, along the trajectory, there is a shooting node for each integration step.

By approaching the direct multiple-shooting formulation as a Sequential Quadratic Programming (SQP) problem, one can describe a single QP iteration as

$$\begin{aligned} \min_{\delta\mathbf{X},\delta\mathbf{U}} \quad & l_N(\delta\mathbf{x}_k) + \sum_{k=0}^{N-1} l_k(\delta\mathbf{x}_k, \delta\mathbf{u}_k) \\ \text{s.t.} \quad & \delta\mathbf{x}_0 = \tilde{\mathbf{x}}_0, \\ & \delta\mathbf{x}_{k+1} = \mathbf{f}_{\mathbf{x}k}\delta\mathbf{x}_k + \mathbf{f}_{\mathbf{u}k}\delta\mathbf{u}_k + \bar{\mathbf{f}}_{k+1}, \end{aligned} \quad (11)$$

where the SQP sequentially builds and solves a single QP problem until it reaches the convergence criteria. The solution of Eq. (11) provides us with a search direction. Then, we can find a step length $\alpha$ for updating the next guess $(\mathbf{X}_{i+1}, \mathbf{U}_{i+1})$ as

$$\begin{bmatrix} \mathbf{X}_{i+1} \\ \mathbf{U}_{i+1} \end{bmatrix} = \begin{bmatrix} \mathbf{X}_i \\ \mathbf{U}_i \end{bmatrix} + \alpha \begin{bmatrix} \delta\mathbf{X}_i \\ \delta\mathbf{U}_i \end{bmatrix} \quad (12)$$

where the new guess trajectory $(\mathbf{X}_{i+1}, \mathbf{U}_{i+1})$ does not necessary close the gaps as we explain below.

*1) KKT problem of the multiple-shooting formulation:* To understand the behavior of the gaps, we formulate the KKT problem in Eq. (12) for a single shooting interval $k$ as:

$$\overbrace{\begin{bmatrix} \mathbf{l}_{\mathbf{x}\mathbf{x}_k} & \mathbf{l}_{\mathbf{x}\mathbf{u}_k} \\ \mathbf{l}_{\mathbf{x}\mathbf{u}_k}^T & \mathbf{l}_{\mathbf{u}\mathbf{u}_k} \end{bmatrix}}^{\mathbf{H}_k} \overbrace{\begin{bmatrix} \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix}}^{\delta\mathbf{w}_k} + \overbrace{\begin{bmatrix} \mathbf{I} & -\mathbf{f}_{\mathbf{x}_k}^T \\ & -\mathbf{f}_{\mathbf{u}_k}^T \end{bmatrix}}^{[\nabla\mathbf{g}_k^- \ \nabla\mathbf{g}_k^+]} \begin{bmatrix} \boldsymbol{\lambda}_k \\ \boldsymbol{\lambda}_{k+1} \end{bmatrix} = -\overbrace{\begin{bmatrix} \mathbf{l}_{\mathbf{x}_k} \\ \mathbf{l}_{\mathbf{u}_k} \end{bmatrix}}^{\nabla\mathbf{\Phi}_k}, \quad (13)$$

$$\begin{bmatrix} \mathbf{I} & \\ -\mathbf{f}_{\mathbf{x}_k} & -\mathbf{f}_{\mathbf{u}_k} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}_k \\ \delta\mathbf{u}_k \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{f}}_k \\ \bar{\mathbf{f}}_{k+1} \end{bmatrix}, \quad (14)$$

[1] A geometrical system has a configuration point that lies on a manifold $Q$, e.g., a Lie group.

where Eq. (13), (14) are the dual and primal feasibility of the First-order Necessary Condition (FONC) of optimality, respectively. The Jacobians and Hessians of the cost function (LQ approximation) are $\mathbf{l_x}$, $\mathbf{l_u}$, and $\mathbf{l_{xx}}$, $\mathbf{l_{xu}}$, $\mathbf{l_{uu}}$, respectively. The Lagrangian of the problem is $(\boldsymbol{\lambda}_k, \boldsymbol{\lambda}_{k+1})$.

We obtain the search direction $(\delta\mathbf{X}_i, \delta\mathbf{U}_i)$ by solving the FONC as follows:

$$\begin{bmatrix} \delta\mathbf{w}_k \\ \delta\boldsymbol{\lambda}_k \\ \delta\boldsymbol{\lambda}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_k & \nabla\mathbf{g}_k^- & \nabla\mathbf{g}_k^+ \\ \nabla\mathbf{g}_k^{-^T} & & \\ \nabla\mathbf{g}_k^{+^T} & & \end{bmatrix}^{-1} \begin{bmatrix} \nabla\boldsymbol{\Phi}_k \\ \bar{\mathbf{f}}_k \\ \bar{\mathbf{f}}_{k+1} \end{bmatrix}, \quad (15)$$

in which we note that a $\alpha$-step closes the gap at $k$ by a factor of $(1-\alpha)\bar{\mathbf{f}}_k$, while only a full-step ($\alpha = 1$) can close the gap completely. Below, we explain how to ensure this multiple-shooting behavior in the forward-pass.

### C. Nonlinear rollout avoids merit function

SQP often requires a *merit* function to compensate the errors that arise from the local approximation of the classical line-search. Defining a suitable merit function is often challenging, which is why we do not follow this approach. Instead, we avoid (a) the linear-prediction error of the dynamics with a nonlinear rollout and (b) the requirement of a merit function. Furthermore, the nonlinear rollout is often faster to compute[2] than the linear-predicted dynamics; $\mathbf{f_x}$, $\mathbf{f_u}$ are large matrices with few, albeit known, non-zero coefficients — a predictable, sparse structure.

For a nonlinear rollout, the prediction of the gaps after applying an $\alpha$-step is:

$$\begin{aligned} \bar{\mathbf{f}}_k^{i+1} &= \bar{\mathbf{f}}_k^i + \alpha(\delta\mathbf{x}_{k+1} - \mathbf{f}_{\mathbf{x}k}\delta\mathbf{x}_k - \mathbf{f}_{\mathbf{u}k}\delta\mathbf{u}_k) \\ &= (1-\alpha)(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_{k+1}), \end{aligned} \quad (16)$$

and we suggest to maintain the same gaps of the linear line-search. Therefore, we have the following rollout:

$$\begin{aligned} \hat{\mathbf{x}}_0 &= \tilde{\mathbf{x}}_0 - (1-\alpha)\bar{\mathbf{f}}_0, \\ \hat{\mathbf{u}}_k &= \mathbf{u}_k + \alpha\mathbf{j}_k + \mathbf{K}_k(\hat{\mathbf{x}}_k - \mathbf{x}_k), \\ \hat{\mathbf{x}}_{k+1} &= \mathbf{f}_k(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k) - (1-\alpha)\bar{\mathbf{f}}_k. \end{aligned} \quad (17)$$

Note that the forward pass of the classical DDP always closes the gaps. In contrast to [13], we fully match the numerical behavior of the gaps such as in the standard multiple-shooting formulation.

### D. Backward pass under an infeasible guess trajectory

Gaps in the dynamics and infeasible warm-start generate derivatives at different points. The Riccati recursion updates the Value and Hamiltonian functions based on these derivatives. The classical DDP algorithm overcomes this problem by first performing an initial forward pass. However, from a theoretical point, it corresponds to only being able to warm-start the solver with the control trajectory $\mathbf{U}_0$, which is not convenient in practice.[3]

[2]In robotics, we often use efficient and recursive rigid-body algorithms based on spatial algebra.

[3]It is straight-forward to obtain a state trajectory $\mathbf{X}_0$ to provide as an initial guess to an OC solver, however, establishing a corresponding control trajectory $\mathbf{U}_0$ beyond quasi-static maneuvers is a limiting factor.

We adapt the backward pass to accept infeasible guesses. For that, we observe that for a LQ model the Hessian is constant and the Jacobian varies linearly. We use this fact to map the Jacobians and Hessian of the Value function from the next shooting-node to the current one. Therefore, the Riccati recursions are modified as follows:

$$\begin{aligned} \mathbf{Q}_{\mathbf{x}_k} &= \mathbf{l}_{\mathbf{x}_k} + \mathbf{f}_{\mathbf{x}_k}^T V_{\mathbf{x}_{k+1}}^+, \\ \mathbf{Q}_{\mathbf{u}_k} &= \mathbf{l}_{\mathbf{u}_k} + \mathbf{f}_{\mathbf{u}_k}^T V_{\mathbf{x}_{k+1}}^+, \\ \mathbf{Q}_{\mathbf{xx}_k} &= \mathbf{l}_{\mathbf{xx}_k} + \mathbf{f}_{\mathbf{x}_k}^T V_{\mathbf{xx}_{k+1}} \mathbf{f}_{\mathbf{x}_k}, \\ \mathbf{Q}_{\mathbf{xu}_k} &= \mathbf{l}_{\mathbf{xu}_k} + \mathbf{f}_{\mathbf{x}_k}^T V_{\mathbf{xx}_{k+1}} \mathbf{f}_{\mathbf{u}_k}, \\ \mathbf{Q}_{\mathbf{uu}_k} &= \mathbf{l}_{\mathbf{uu}_k} + \mathbf{f}_{\mathbf{u}_k}^T V_{\mathbf{xx}_{k+1}} \mathbf{f}_{\mathbf{u}_k}. \end{aligned} \quad (18)$$

where $V_{\mathbf{x}_{k+1}}^+ = V_{\mathbf{x}_{k+1}} + V_{\mathbf{xx}_{k+1}}\bar{\mathbf{f}}_{k+1}$ is the Jacobian of the Value function after the deflection produced by the gap $\bar{\mathbf{f}}_{k+1}$, and the Hessian of the Value function remains unchanged. With this approach, we can also include adaptive integration schemes.

### E. Accepting a step

The expectation of the total cost reduction proposed by [9] does not consider the deflection introduced by the gaps. This is a critical point to evaluate the success of a trial step during the numerical optimization. From our line-search procedure, we know that the expected reduction on the cost has the form:

$$\Delta J(\alpha) = \Delta_1\alpha + \frac{1}{2}\Delta_2\alpha^2, \quad (19)$$

where, by closing the gaps as predicted in Eq. (15) in the linear rollout, we obtain:

$$\begin{aligned} \Delta_1 &= \sum_{k=0}^{N} \mathbf{k}_k^\top \mathbf{Q}_{\mathbf{u}_k} + \bar{\mathbf{f}}_k^\top (V_{\mathbf{x}_k} - V_{\mathbf{xx}_k}\mathbf{x}_k), \\ \Delta_2 &= \sum_{k=0}^{N} \mathbf{k}_k^\top \mathbf{Q}_{\mathbf{uu}_k}\mathbf{k}_k + \bar{\mathbf{f}}_k^\top (2V_{\mathbf{xx}_k}\mathbf{x}_k - V_{\mathbf{xx}_k}\bar{\mathbf{f}}_k). \end{aligned} \quad (20)$$

Note that if all gaps are closed, then this expectation model matches the one reported in [9].

We use the Goldstein condition to check for the trial step, instead of the Armijo condition typically used in classical DDP algorithms, e.g., [9]. The reason is due to the fact that $\Delta$ might be an ascent direction, for instance, during the infeasible iterations. Therefore, FDDP accepts the step if the cost reduction is:

$$l' - l \leq \begin{cases} b_1\Delta(\alpha) & \text{if } \Delta(\alpha) \leq 0 \\ b_2\Delta(\alpha) & \text{otherwise} \end{cases} \quad (21)$$

where $b_1$, $b_2$ are adjustable parameters. This critical mathematical aspect has not been considered neither in [13].

### F. Regularization and search direction

We employ two regularization schemes: the Tikhonov regularization over $\mathbf{Q_{uu}}$, and the Tassa regularization over $V_\mathbf{x}$ [9]. The Tikhonov regularization changes the search-direction from Gauss-Newton to steepest descent. We exploit this fact to ensure a good progress towards the (local) optimal solution.

Roughly speaking, we increase both regularizations whenever the backward pass fails or the applied step length is too short; otherwise we reduce them.

## IV. Crocoddyl Framework

Crocoddyl [23] has been written in C++ for efficiency and uses the Eigen library [24] for linear algebra routines. It comes with Python bindings for easy prototyping. Crocoddyl is currently supported for most Linux distributions, with plans to release on Mac OS X and Windows. The project is fully open-source under the permissive BSD-3-Clause license and is hosted on GitHub: https://github.com/loco-3d/crocoddyl. In the rest of this section, we introduce the main features of Crocoddyl.

### A. Analytical and sparse derivatives

Crocoddyl uses Pinocchio to efficiently compute analytical and sparse derivatives [12]. Pinocchio has dedicated algorithms to compute analytical derivatives of rigid-body algorithms (e.g. RNEA and Articulated Body Algorithm (ABA)) using spatial algebra [21]. Crocoddyl employs these routines to derive the analytical derivative of contact dynamics as described in Section II-B, as well as the ones of cost functions, e.g., Center of Mass (CoM) tracking, frame placement tracking, etc. Crocoddyl shares the computation load required for the dynamics, costs, and constraints by storing the data in a common container.

### B. Model and data

One of the main design concepts of Crocoddyl is the strict separation between *model* and *data*. A *model* describes a system or procedure, e.g. *action*, *cost*, and *activation* models. Any *model* has been abstracted to easily derive new systems in both C++ and Python. By *data*, we refer to a container that stores computed and intermediate values used during the calculation routine performed by a *model*. Each model creates its own data, and with this, we avoid any temporary memory allocation produced by algebraic expressions in Eigen. To improve efficiency, especially in systems with dimensions lower than 16, the Eigen members inside a data object can be defined with fixed dimensions. It allows for efficient use of modern CPU features by using vectorization and Same Instruction Multiple Data (SIMD) operations.

*1) Action models:* The most important model is the *action* one. An action model combines dynamics, cost, and constraint models. Each node, in our OC problem, is described through an action model. With this plain description, we can easily model hybrid optimal control problems (or logic-based problem formulations) that typically arise in robots with legs and arms.

To numerically solve the nonlinear OC problem, we have to compute a search direction and a step length. The former procedure is typically computed from the derivatives of the action models — Jacobians and Hessians. The latter requires to rollout the dynamics, cost, and constraints along different step lengths. We have divided both procedures into two main functions: `calcDiff` and `calc`, respectively. The results of these functions (i.e., Jacobians, Hessians, next state, cost value, etc.) are stored inside their corresponding *action data*. The fact that the memory required for a data object is only allocated once, at the start-up/initialization phase, enhances the time-predictability and parallelizability of the code. Finally, some of our action models use Pinocchio [12] to efficiently compute rigid-body algorithms and their analytical derivatives. We can easily use numerical and automatic differentiation routines for debugging and prototyping of action models.

*2) Differential and integrated action models:* It is often convenient to implement action models in continuous time, which we call it *differential action model*. Together with a predefined *integrated action models*, it is possible to retrieve the time-discrete action model. Our integrated action models (i.e. integration schemes) conserve the geometrical properties of the system, e.g., the properties of the $\mathbb{SE}(3)$ manifold in a floating-base system.

### C. Dealing with geometrical systems

Generally speaking, the state of the system lies in a manifold $X$ where its rate of change lies in the tangent space $T_{\mathbf{x}}X$ (see Section II-A). Each manifold obeys to its own algebra, i.e., there are specific rules for the following operations:

$$\begin{aligned} \mathbf{x}' &= \mathbf{x} \oplus \delta\mathbf{x}, \\ \delta\mathbf{x} &= \mathbf{x}_1 \ominus \mathbf{x}_0, \end{aligned} \tag{22}$$

where $\mathbf{x} \in X$ and $\delta\mathbf{x} \in T_{\mathbf{x}}X$ and can be described by a $n_x$- and $n_{dx}$-tuples, respectively. These operations are encoded by `integrate` and `difference`, respectively. We also need to define the Jacobians of these operators which are needed for our optimal control solver, i.e.

$$\begin{aligned} \frac{\partial \mathbf{x} \oplus \delta\mathbf{x}}{\partial \mathbf{x}}, \frac{\partial \mathbf{x} \oplus \delta\mathbf{x}}{\partial \delta\mathbf{x}} &= \texttt{Jintegrate}(\mathbf{x}, \delta\mathbf{x}), \\ \frac{\partial \mathbf{x}_1 \ominus \mathbf{x}_0}{\partial \mathbf{x}_0}, \frac{\partial \mathbf{x}_1 \ominus \mathbf{x}_0}{\partial \mathbf{x}_1} &= \texttt{Jdifference}(\mathbf{x}_0, \mathbf{x}_1), \end{aligned} \tag{23}$$

where they return the Jacobians for the first and second arguments of the method. In addition, we have developed efficient and dedicated routines for computing these operations and Jacobians given the URDF model of a multibody system. Note that for systems that lie in the Euclidean space, these operations are described using linear algebra and Jacobians are the identity matrix.

## V. Results

In this section, we show the capabilities of our multi-contact optimal control framework. We first compute various legged gaits for both quadruped and biped robots (Section V-A). The computation of the gait motion and commands is solved in few iterations and milliseconds. As our formulation is simple and does not depend on a good initial guess, it can be used easily with different legged robots. Next, we analyze the performance of the FDDP with the generation of highly-dynamic maneuvers such as jumps and front-flips. These motions are computed within a few iterations and milliseconds, and thanks to the dedicated Lie algebra developed in Crocoddyl, our framework

does not suffer from orientation singularity. The accompanying videos[4] highlight different motions generated by Crocoddyl.

### A. Various legged gaits

We computed different gaits — walking, trotting, pacing, and bounding — with our FDDP algorithm in the order of milliseconds (Fig. 2). All these gaits are a direct outcome of our algorithm, we only need to predefine the sequence of contacts and the step timings. These motions are computed in around 12 iterations with the exception of the bounding gait which takes at least 18 iterations. We used the same weight values and cost functions for all the quadrupedal gaits, and similar weight values for the bipedal walking. Indeed, we noticed that these weight values and cost functions might work out of the box for other legged robots, e.g., the HyQ and the ICub robots. We report the used values in Fig. 3.

The cost function is composed of the CoM and the foot placement tracking costs together with regularization terms for the state and control. We used piecewise-linear functions to describe the reference trajectory for the swing foot. Additionally, we strongly penalize footstep deviation from the reference placement. We warm-start our solver using a linear interpolation between the nominal body postures of a sequence of contact configurations. This provides us a set of body postures together with the nominal joint postures as state warm-start $\mathbf{X}_0$. Then, the control warm-start $\mathbf{U}_0$ is obtained by applying the quasi-static assumption[5] along $\mathbf{X}_0$.

In each switching phase[6], we use the impulse dynamics to ensure the contact velocity equals zero, see Eq. (7). We observed that the use of impulse models improves the algorithm convergence compared to penalizing the contact velocity. We used a weighted least-square function to regularize the state with respect to the nominal robot posture, and quadratic functions for the tracking costs and control regularization.

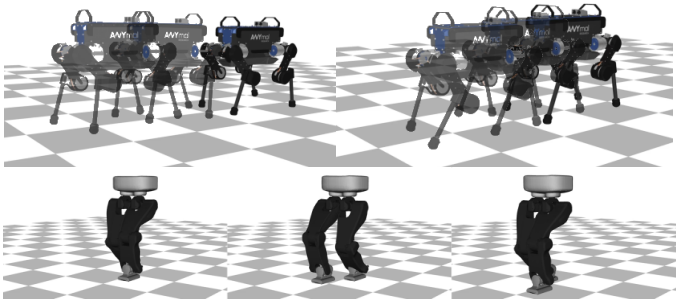

Fig. 2. Different legged gaits optimized by our FDDP algorithm. (top) from left to right, walking and trotting gaits on the ANYmal robot, respectively; (bottom) biped walking gait using the Talos' legs. You can run these results in our repository [23].

### B. Highly-dynamic maneuvers

Our FDDP algorithm is able to compute highly-dynamic maneuvers such as front-flip and jumping in the order of milliseconds (Fig. 4). These motions are often computed between 12–36 iterations with a naive and infeasible $\mathbf{X}_0, \mathbf{U}_0$ warm-start. We used the same initialization, weight values and cost functions reported in Section V-A, with a slightly incremented weight for the state regularization during the impact phases (i.e. $w_{xReg} = 10$). Additionally, and for simplicity, we included a cost that penalizes the body orientation in the ICub jumps. Similarly to other cost functions, we used a quadratic penalization with a weight value of $10^4$. Note that a more elaborate cost function could be incorporated: arm motions, angular momentum regulation, etc.

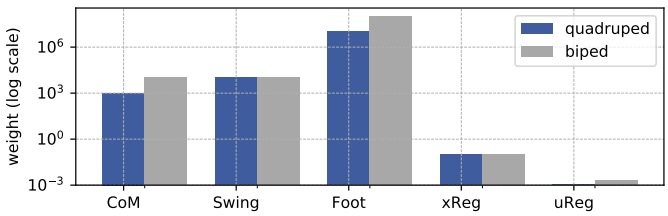


Fig. 3. Weight values for each cost function used in the generation of the legged gaits. The cost functions are (a) CoM: CoM tracking of the interpolated postures; (b) Swing: swing tracking of the reference foot trajectory; (c) footstep deviation from the predefined placement; (d) state regularization; (e) control regularization. Note that the values are in logarithmic scale.

The real advantage of our FDDP algorithm is clearly evident in the generation of highly-dynamic maneuvers, where feasible rollouts might produce trajectories that are unstable and far from the solution. The classical DDP has a poor globalization strategy that comes from inappropriate feasible rollouts in the first iterations; it struggles to solve these kind of problems. To overcome this limitation in the DDP, we need to properly warm-start the solver, in particular the state trajectory. This limits our solver to problems for which a suitable warm-start can be provided, e.g., from optimized centroidal trajectories [8].

### C. Runtime, contraction, and convergence

We analyzed the gaps contraction and convergence rates for all the presented motions: quadrupedal and bipedal gaits, and highly-dynamic maneuvers. To easily compare the results, we normalize the gaps and cost values per each iteration as shown in Fig. 5. For the case of the gaps plot Fig. 5 (top), we use the L2-norm of the total gaps. These results show that keeping the gaps open is particularly important for highly-dynamic maneuvers such as jumping. Indeed, the FDDP solver immediately closes the gaps for all the legged gaits. The reason is that the solver can apply a full-step in the first iteration with simply using a big initial regularization value[7] ($10^{-2}$). ANYmal and ICub jumps are computed with a sequence of OC problems, where each jump formulates a single problem. Additionally, we observed in practice super-linear convergence of FDDP algorithm after closing the gaps. This is expected since FDDP behaves as DDP when the gaps are closed.

Highly-dynamic maneuvers have a lower rate of improvement in the first iterations Fig. 5 (bottom). The same occurs

[4]https://youtu.be/rZ135Dw92No.

[5]The quasi-static torques are numerically computed through Newton steps using the reference posture as an equilibrium point.

[6]In this work, with "switching phases" we refer to contact gain.

[7]A big regularization value changes the search-direction from Newton to steepest-descent.

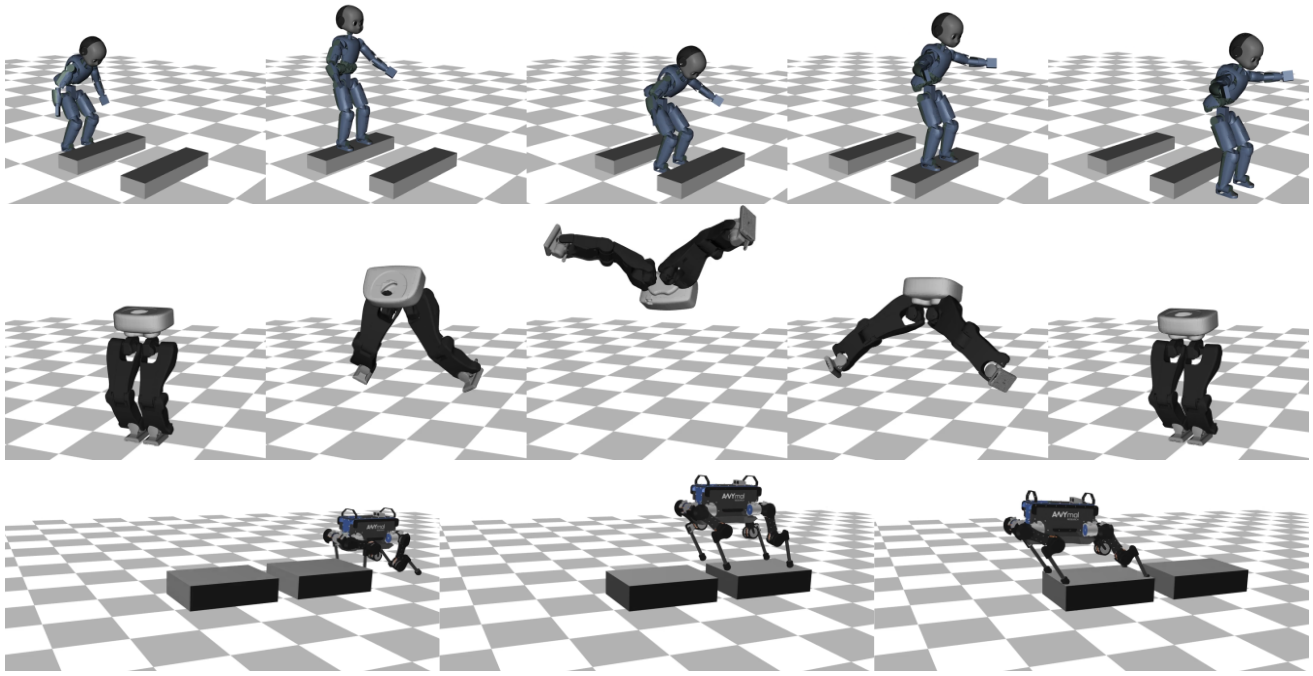

Fig. 4. Snapshots of generated highly-dynamic maneuvers in legged robots using the feasibility-prone differential dynamic algorithm. (top) jumping obstacles in a humanoid robot; (middle) front-flip maneuver in a biped robot; (bottom) jumping obstacles in a quadruped robot.

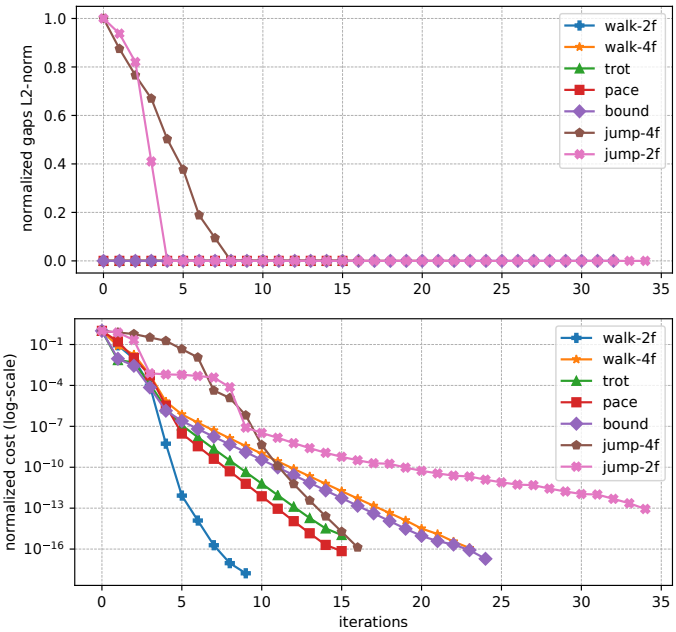


Fig. 5. Gaps contraction and convergence rates for different motions. (top) Gaps are closed in the first iteration for simpler motions such as biped walking and quadrupedal gaits. Instead, the FDDP solver chooses to keep the gaps open for the early iterations for highly-dynamic maneuvers. Note that we use the L2-norm of the total gaps, i.e., gaps for all the nodes of the trajectory. (bottom) The required iterations increases mainly with the dynamics of the gait and numbers of nodes. For instance, we can see lower rate of improvement in the first nine iterations in the ANYmal (jump-4f) and ICub jumps (jump-2f). In case of the quadrupedal walking, we have very short durations in the four-feet support phases, making it a *dynamic* walk.

in the quadrupedal walking case (walk-4f), in which the four-feet support phases have a very short duration ($\Delta t = 2$ ms). Our FDDP algorithm together with the impact models shows better and similar convergence rates compared to Sequential Linear Quadratic (SLQ) solvers for whole-body and centroidal OC control [11], [25], respectively.

The motions converge within 10 to 34 iterations, with an overall computation time of less than $0.5\,\mathrm{s}$. The numerical integration step size is often $\delta t = 10^{-2}$ s, with the exception of the biped walking $\delta t = 3^{-2}$ s, and the number of nodes are typically between 60 to 115. Therefore, the optimized trajectories have a horizon of between $0.6\,\mathrm{s}$ to $3\,\mathrm{s}$. For more details please refer to the accompanying videos.[8]

We also benchmark the computation time for a single iteration using our solver. The number of contacts does not affect the computation time; it scales linearly with respect to the number of nodes. Without multi-threading, our efficient implementation of contact dynamics achieves computation rates up to $859.6\,\mathrm{Hz}$ for the quadrupedal gaits with 60 nodes (jump-4f on i9-9900K). We parallelize only the computation of the derivatives. For this case, roughly speaking, we reduce the computation time in half using four to eight threads (cf. Fig. 6). To understand the performance of Crocoddyl, we have run 5000 trials for each of the motions presented in this letter on four different Intel PCs with varying levels of parallelization: *PC1*: i7-6700K @ $4.00\mathrm{GHz}\times 8$ with 32 GB 2133MHz RAM, *PC2*: i7-7700K @ $4.20\mathrm{GHz}\times 8$ with 16 GB 2666MHz RAM, *PC3*: i9-9900K @ $3.60\mathrm{GHz}\times 16$ with 64 GB 3000MHz RAM, and *PC4*: i7-9900XE @ $3.00\mathrm{GHz}\times 36$ with 128 GB 2666MHz RAM. We used the optimal number of threads for each PC as identified in Fig. 6. The computation frequency per one iteration is reported in Fig. 7.

[8]See footnote 5.

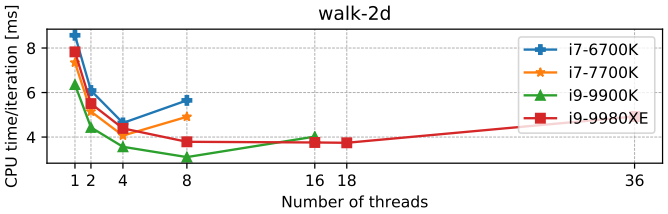


Fig. 6. Computation time per iteration for different CPUs and level of parallelism. Note that the use of hyper-threading decreases the computation frequency for all tested CPUs.

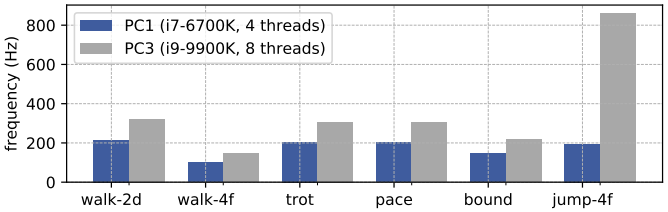


Fig. 7. Computation frequency per iteration for different motions for different PCs. PC1 has specifications typical for on-board computers found on robots, while PC3 uses high-performance CPU and RAM. The reported values use the optimal number of threads as identified in Fig. 6.

## VI. Conclusion

We presented a novel and efficient framework for multi-contact optimal control that outperforms state-of-the-art methods. We have shown that our novel FDDP algorithm is able to match the exact behavior of multiple-shooting methods. However, and in contrast to these methods, the FDDP does not add extra computation complexity. With the FDDP algorithm, we improve the poor globalization strategy of classical DDP methods. This allows us to solve highly-dynamic maneuvers such as jumping and front-flip in the order of milliseconds. Thanks to our efficient method for computing the contact dynamics and their derivatives, we can solve the optimal control problem at high frequencies. Finally, we demonstrated the benefits of using impact models for contact gain phases. Future work will focus on including efficient and computationally inexpensive procedures for including inequality constraints such as torque limits, friction cone, and self-collision.

## Acknowledgment

This research was supported by the European Commision under the Horizon 2020 project Memory of Motion (MEMMO, project ID: 780684). The authors would also like to thank all the developers of Pinocchio and Eigen which make this work possible by providing high-quality and open-source software. Last but not least, the authors especially thank Gabriele Buondonno, Joseph Mirabel, Andrea Del Prete, and other members of the MEMMO project for their precious feedback.

## References

[1] C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Motion planning for quadrupedal locomotion: coupled planning, terrain mapping and whole-body control," 2017, preprint.

[2] B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernandez-Lopez, and C. Semini, "Simultaneous Contact, Gait and Motion Planning for Robust Multi-Legged Locomotion via Mixed-Integer Convex Optimization," *IEEE Robot. Automat. Lett.*, 2017.

[3] J. Carpentier, S. Tonneau, M. Naveau, O. Stasse, and N. Mansard, "A versatile and efficient pattern generator for generalized legged locomotion," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2016.

[4] C. Dario Bellicoso, F. Jenelten, P. Fankhauser, C. Gehring, J. Hwangbo, and M. Hutter, "Dynamic locomotion and whole-body control for quadrupedal robots," in *IEEE Int. Conf. Intell. Rob. Sys. (IROS)*, 2017.

[5] A. W. Winkler, D. C. Bellicoso, M. Hutter, and J. Buchli, "Gait and Trajectory Optimization for Legged Systems through Phase-based End-Effector Parameterization," *IEEE Robot. Automat. Lett.*, 2018.

[6] S. Fahmi, C. Mastalli, M. Focchi, D. G. Caldwell, and C. Semini, "Passivity Based Whole-body Control for Quadruped Robots: Experimental Validation over Challenging Terrain," *IEEE Robot. Automat. Lett.*, 2019.

[7] P.-B. Wieber, "Holonomy and nonholonomy in the dynamics of articulated motion," in *Fast Motions in Biomechanics and Robotics*, 2005.

[8] R. Budhiraja, J. Carpentier, C. Mastalli, and N. Mansard, "Differential Dynamic Programming for Multi-Phase Rigid Contact Dynamics," in *IEEE Int. Conf. Hum. Rob. (ICHR)*, 2018.

[9] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IEEE Int. Conf. Intell. Rob. Sys. (IROS)*, 2012.

[10] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the hrp-2 humanoid," in *IEEE Int. Conf. Intell. Rob. Sys. (IROS)*, 2015.

[11] M. Neunert, F. Farshidian, A. W. Winkler, and J. Buchli, "Trajectory optimization through contacts and automatic gait discovery for quadrupeds," *IEEE Robot. Automat. Lett.*, 2017.

[12] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE Int. Sym. System Integration (SII)*, 2019.

[13] M. Giftthaler, M. Neunert, M. Stuble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *IEEE Int. Conf. Intell. Rob. Sys. (IROS)*, 2018.

[14] H. Bock and K. Plitt, "A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems," *IFAC Proceedings Volumes*, 1984.

[15] R. Kalaba and F. Udwadia, "Lagrangian mechanics, Gauss's principle, quadratic programming, and generalized inverses: new equations for nongolonomically constrained discrete mechanical systems," *Quarterly of Applied Mathematics*, 1994.

[16] R. Budhiraja, J. Carpentier, and N. Mansard, "Dynamics consensus between centroidal and whole-body models for locomotion of legged robots," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2019.

[17] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer Methods in Applied Mechanics and Engineering*, 1972.

[18] J.-L. Blanco, "A tutorial on se(3) transformation parameterizations and on-manifold optimization," University of Malaga, Tech. Rep., 2010.

[19] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, 1966.

[20] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robot. Automat.*, 1987.

[21] J. Carpentier and N. Mansard, "Analytical Derivatives of Rigid Body Dynamics Algorithms," in *Robotics: Science and Systems (RSS)*, 2018.

[22] R. Featherstone, *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007.

[23] C. Mastalli, R. Budhiraja, and N. Mansard, "Crocoddyl: a fast and flexible optimal control library for robot control under contact sequence," https://github.com/loco-3d/crocoddyl, 2019.

[24] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[25] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2017.