2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Squash-Box Feasibility Driven Differential Dynamic Programming

Josep Marti-Saumell Joan Solà Carlos Mastalli Angel Santamaria-Navarro

***Abstract*— Recently, Differential Dynamic Programming (DDP) and other similar algorithms have become the solvers of choice when performing non-linear Model Predictive Control (nMPC) with modern robotic devices. The reason is that they have a lower computational cost per iteration when compared with off-the-shelf Non-Linear Programming (NLP) solvers, which enables its online operation. However, they cannot handle constraints, and are known to have poor convergence capabilities. In this paper, we propose a method to solve the optimal control problem with control bounds through a squashing function (*i.e.*, a sigmoid, which is bounded by construction). It has been shown that a naive use of squashing functions damage the convergence rate. To tackle this, we first propose to add a quadratic barrier that avoids the difficulty of the plateau produced by the sigmoid. Second, we add an outer loop that adapts both the sigmoid and the barrier; it makes the optimal control problem with the squashing function converge to the original control-bounded problem. To validate our method, we present simulation results for different types of platforms including a multi-rotor, a biped, a quadruped and a humanoid robot.**

I. Introduction

A. Motivation and related work

Nonlinear Model Predictive Control (nMPC) is a powerful technique that is used in robotics to generate trajectories. It finds the control commands (over a period of time) by optimizing a user-defined task, *i.e.*, a cost function. Most of the approaches solve, numerically, a static optimization for each MPC step, *i.e.*, by following the so-called *first discretize, then optimize* approach. The static optimization is typically formulated using a *direct* approach [1], in which a general-purpose nonlinear program is used to retrieve the solution, *e.g.* SNOPT [2], KNITRO [3], and IPOPT [4].

Applying nMPC to robotic systems with high dynamics, *e.g.* aerial vehicles [5], quadrupeds [6] and humanoids [7], requires fast solvers to obtain optimal control commands at sensor sampling rates. With current computer capabilities, this is hard to achieve using the general-purpose solvers cited above because they require big matrix factorizations. Thus, the nMPC is usually addressed by resorting to Differential Dynamic Programming (DDP) techniques [8] —or any variant of it such as the iterative Linear Quadratic Regulator (iLQR) [9]. The key idea behind DDP is to break the whole problem into a sequence of smaller sub-problems, thanks to the Bellman's principle of optimality. However, these methods, in their original form, have a poor globalization capability and lack the ability to handle constraints beyond the ones imposed by the dynamics.

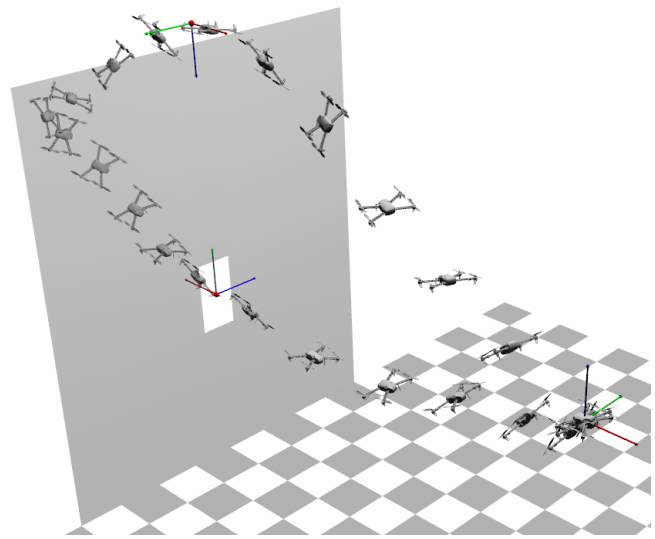

Fig. 1: A complex maneuver computed with the proposed method SquashBox-FDDP. The goal of the UAV is to pass through two defined configuration points while generating limited thrust commands. We draw both configuration points as frames with the blue axis indicating the vertical direction of the UAV.

Recent works have tackled some of the mentioned limitations. In [10], Giffhaler et al. present a lifted version of the Riccati equations that enables to warm-start the solver and to avoid an initial, and commonly unstable, rollout. Later, Mastalli et al. [11] propose modifications of both backward/forward passes to emulate the numerical behavior of a direct multiple-shooting formulation with equality constraints. This latter method is called Feasibility-driven Differential Dynamic Programming (FDDP), and it has shown greater globalization strategy needed for highly-dynamic maneuvers in legged robots.

In parallel to the research for improving the globalization capability, other works have focused on including arbitrary constraints. In [12], Tassa et al. propose a method to add bounds to the control actions (Box-DDP). To do so, they propose to handle the box constraints inside the Quadratic

J. Marti-Saumell and Joan Solà are with the Institut de Robòtica i Informàtica Industrial, CSIC-UPC, Llorens Artigas 4-6, Barcelona 08028, Spain (e-mail: jmarti, jsola@iri.upc.edu).

C. Mastalli is with the Informatics School, University of Edinburgh and the Alan Turing Institute, Edinburgh, UK (e-mail: carlos.mastalli@ed.ac.uk).

A. Santamaria-Navarro is with NASA-JPL, California Institute of Technology, Pasadena, CA 91109 USA (e-mail: angel.santamaria.navarro@jpl.nasa.gov).

This work was partially supported by the EU H2020 project GAUSS (H2020-Galileo-2017-1-776293), project EB-SLAM (DPI2017-89564-P), by the Spanish State Research Agency through the María de Maeztu Seal of Excellence to IRI (MDM-2016-0656) and by the European Commission under the Horizon 2020 project Memory of Motion (MEMMO, project ID: 780684). Part of this research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (NASA, USA).

Programming (QP) problem that minimizes the Hamiltonian (*i.e.*, the $\mathbf{Q}$ function). Following a similar approach, Xie et al. [13] extend the method for arbitrary nonlinear constraints on the states and the controls. Independently, Lantoine and Russell [14] take a similar approach to handle hard constraints. Besides, they propose to use an Augmented Lagrangian method. Another Augmented Lagrangian extension called ALTRO [15] has been proposed recently.

B. Contribution and outline

In this paper we propose a method to solve the optimal control problem with control bounds. We do so by means of a squashing function (SF), *i.e.*, a sigmoid function that is bounded by construction. It maps the unbounded variable s to the bounded output u, *i.e.*, $u = g(s)$. When using a SF, we substitute the control variable u by the SF input s. With this modification, the optimal control problem constrained on the control u becomes an unconstrained problem on s, and the control bounds are guaranteed by the SF.

As stated in [12], only considering this variable substitution constitutes a naive approach that leads to poor practical performance. Mainly, this happens because the shape of the sigmoid cannot be captured by the quadratic approximation considered in DDP algorithms. In order to tackle this, we modify the naive solution and propose a penalty method that adds a quadratic barrier on the squashing variable s, together with the consideration of an outer loop.

The outer loop is responsible to jointly control the sharpness of the squashing function and the steepness of the barrier, while the inner loop calls the FDDP solver, presented in [11], to solve a penalized problem. Our approach starts with a smooth penalization and makes it sharper as outer iterations pass. Eventually, this modified problem converges to the initial bounded problem. The performance of our algorithm is shown in simulations of several challenging tasks involving one aerial, one humanoid, and one quadruped robot.

The rest of this paper is organized as follows. In Section II we present an overview of the original DDP algorithm as well as its feasibility-driven version (FDDP). In Section III we describe our proposed algorithm called squash-box FDDP (sb-FDDP). In Section IV we compare our method with the naive SF as well as with the Box-DDP of [12]. The paper finishes with conclusions and discussion in Section V.

II. Background

The control inputs are commonly limited by the real system characteristics. Therefore, when solving the optimal control problem, we should account for the following bounds on the control inputs $\mathbf{u}$,

$$\begin{aligned} \min_{\mathbf{X},\mathbf{U}} \; & l_N(\mathbf{x}_N) + \sum_{k=0}^{N-1} l_k(\mathbf{x}_k, \mathbf{u}_k) \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), && \text{(dynamics)} \\ & \mathbf{x}_0 = \mathbf{x}(0), && \text{(initial condition)} \\ & \underline{\mathbf{u}} \leq \mathbf{u}_k \leq \bar{\mathbf{u}}, && \text{(control bounds)} \end{aligned} \tag{1}$$

where N defines the number of nodes along the discretized trajectory $(\mathbf{X}, \mathbf{U})$, defined as $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_N]$ and $\mathbf{U} = [\mathbf{u}_0, \ldots, \mathbf{u}_{N-1}]$, $\mathbf{x}_k \in \mathbb{R}^{n_x}$ and $\mathbf{u}_k \in \mathbb{R}^{n_u}$ are the state and control at the k-th node, respectively, $l_k(\mathbf{x}_k, \mathbf{u}_k) : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}$ is its associated cost, $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}^{n_x}$ describes its system evolution, and $[\underline{\mathbf{u}}, \overline{\mathbf{u}}]$ is the control interval (lower and upper bounds, respectively).

In the following section we introduce the algorithms of DDP and FDDP. The classical DDP does not include the state trajectory $\mathbf{X}$ as a decision variable. This is the reason why it is considered an *indirect* method (Section II-A). On the contrary, FDDP does consider $\mathbf{X}$ as decision variables, fact that improves the globalization capability of the algorithm (Section II-B). FDDP is the solver used in the inner loops of our proposed method, presented in Section III.

A. Differential dynamic programming (DDP)

DDP splits the whole problem into a sequence of N smaller problems. Each sub-problem only considers the controls $\mathbf{u}_k$ as decision variables. DDP is known to have a low memory footprint and a high solving speed [8]. It comprises the following three main stages: (1) derivatives computation at each node, (2) backward pass and (3) forward pass. Below we briefly describe the backward and forward passes.

1) Backward pass: This pass owes its name to the backward recursion that gives, as a result, an optimal policy at every node. It starts from the final node and ends at the first one. The cost associated to the tail of the trajectory from the node i onward can be expressed as the *cost-to-go*, *i.e.*,

$$J_i(\mathbf{x}_i, \mathbf{U}_i) = \sum_{k=i}^{N-1} l_k(\mathbf{x}_k, \mathbf{u}_k) + l_N(\mathbf{x}_N) \tag{2}$$

where the tail of trajectory $\mathbf{U}_i$ is considered to be from the node i until the end. Thanks to the Bellman principle, we can recursively solve this problem as

$$\begin{aligned} V_i(\mathbf{x}_i) &\triangleq \min_{\mathbf{U}_i} J_i(\mathbf{x}, \mathbf{U}_i) = \\ &= \min_{u_i} [l_i(\mathbf{x}_i, \mathbf{u}_i) + V_{i+1}(\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i))] \end{aligned} \tag{3}$$

where $V_i(\mathbf{x}_i) \in \mathbb{R}$ is the *optimal cost-to-go*. Then, DDP finds the local optimum of the following expression

$$\begin{aligned} Q_i(\delta\mathbf{x}_i, \delta\mathbf{u}_i) = l_i(&\mathbf{x}_i + \delta\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_i) \\ &+ V_{i+1}(\mathbf{f}(\mathbf{x}_i + \delta\mathbf{x}_i, \mathbf{u}_i + \delta\mathbf{u}_i)) \,. \end{aligned} \tag{4}$$

From now on subindices i will be omitted and the optimal cost to go at the next time step will be indicated with a prime, *i.e.*, $V'(\mathbf{f}(\mathbf{x}, \mathbf{u})) \triangleq V_{i+1}(\mathbf{f}(\mathbf{x}_i, \mathbf{u}_i))$. The optimum of (4) is found iteratively by writing the quadratic approximation,

$$Q(\delta\mathbf{x}, \delta\mathbf{u}) \approx \frac{1}{2} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}^\top \begin{bmatrix} 0 & \mathbf{Q}_\mathbf{x}^\top & \mathbf{Q}_\mathbf{u}^\top \\ \mathbf{Q}_\mathbf{x} & \mathbf{Q}_{\mathbf{xx}} & \mathbf{Q}_{\mathbf{xu}} \\ \mathbf{Q}_\mathbf{u} & \mathbf{Q}_{\mathbf{xu}}^\top & \mathbf{Q}_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} 1 \\ \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix} , \tag{5}$$

and then optimizing with respect to $\delta\mathbf{u}$. The partial derivatives of (4) are the following:

$$\mathbf{Q_x} = \mathbf{l_x} + \mathbf{f_x}^\top \mathbf{V}'_\mathbf{x}\,, \tag{6a}$$
$$\mathbf{Q_u} = \mathbf{l_u} + \mathbf{f_u}^\top \mathbf{V}'_\mathbf{x}\,, \tag{6b}$$
$$\mathbf{Q_{xx}} = \mathbf{l_{xx}} + \mathbf{f_x}^\top \mathbf{V}'_\mathbf{xx}\mathbf{f_x} + \mathbf{V}'_\mathbf{x}\cdot\mathbf{f_{xx}}\,, \tag{6c}$$
$$\mathbf{Q_{ux}} = \mathbf{l_{ux}} + \mathbf{f_u}^\top \mathbf{V}'_\mathbf{xx}\mathbf{f_x} + \mathbf{V}'_\mathbf{x}\cdot\mathbf{f_{ux}}\,, \tag{6d}$$
$$\mathbf{Q_{uu}} = \mathbf{l_{uu}} + \mathbf{f_u}^\top \mathbf{V}'_\mathbf{xx}\mathbf{f_u} + \mathbf{V}'_\mathbf{x}\cdot\mathbf{f_{uu}}\,, \tag{6e}$$

where $\mathbf{V}'_\mathbf{x}$, $(\mathbf{l_x},\ \mathbf{l_u})$, $(\mathbf{f_x},\ \mathbf{f_u})$ and $\mathbf{V}'_\mathbf{xx}$, $(\mathbf{l_{xx}},\ \mathbf{l_{ux}},\ \mathbf{l_{uu}})$, $(\mathbf{f_{xx}},\ \mathbf{f_{ux}},\ \mathbf{f_{uu}})$ describe the Jacobians and Hessians of the Value, cost and dynamics functions, respectively. Note that the last terms of (6c)-(6e) denote the product of a vector by a tensor.

The minimization of (5) with respect to $\delta\mathbf{u}$ leads to the following optimal policy:

$$\delta\mathbf{u}^*(\delta\mathbf{x}) = \mathbf{k} + \mathbf{K}\delta\mathbf{x}\,, \tag{7}$$

with $\mathbf{k} \triangleq -\mathbf{Q_{uu}^{-1}}\mathbf{Q_u}$ and $\mathbf{K} \triangleq -\mathbf{Q_{uu}^{-1}}\mathbf{Q_{ux}}$. If we plug this optimal policy into the quadratic expansion in (5), we obtain a quadratic approximation of the optimal cost to go as a function of $\delta\mathbf{x}$, *i.e.*,

$$V(\delta\mathbf{x}) = \Delta V + \mathbf{V_x}^\top\delta\mathbf{x} + \delta\mathbf{x}^\top\mathbf{V_{xx}}\delta\mathbf{x}\,, \tag{8}$$

where

$$\Delta V = -\frac{1}{2}\mathbf{k}^\top\mathbf{Q_{uu}}\mathbf{k}\,, \tag{9a}$$
$$\mathbf{V_x} = \mathbf{Q_x} - \mathbf{K}^\top\mathbf{Q_{uu}}\mathbf{k}\,, \tag{9b}$$
$$\mathbf{V_{xx}} = \mathbf{Q_{xx}} - \mathbf{K}^\top\mathbf{Q_{uu}}\mathbf{K}\,. \tag{9c}$$

The set of equations (6) and (9) constitute the backward pass. These equations are used to update recursively the optimal policy for each node.

2) Forward pass: Considering a current guess $(\mathbf{X}, \mathbf{U})$, the forward pass applies appropriate modifications to the current guess trajectories as follows:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0\,, \tag{10a}$$
$$\hat{\mathbf{u}}_k = \mathbf{u}_0 + \alpha\mathbf{k}_k + \mathbf{K}_k(\hat{\mathbf{x}}_k - \mathbf{x}_k)\,, \tag{10b}$$
$$\hat{\mathbf{x}}_{k+1} \triangleq \mathbf{f}(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k)\,, \tag{10c}$$

where the $\hat{\mathbf{x}}$ and $\hat{\mathbf{u}}$ are the updated values for the states and controls, respectively. The assignment in (10c) implies the integration of the dynamics to update the next state. This procedure is also known as *nonlinear rollout*. The parameter $\alpha \in (0, 1]$ indicates the length of the step taken by the current iteration. A value of $\alpha = 1$ results in the application of a full step.

B. Feasibility-driven DDP (FDDP)

For the sake of completeness, we briefly explain the FDDP algorithm. To gain insight into this method, we refer the reader to [11].

The major modification introduced in FDDP is the addition of the state trajectory $\mathbf{X}$ as an artificial decision variables. As a multiple shooting technique, FDDP allows for infeasible trajectories during the solving process, *i.e.*,

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \overline{\mathbf{f}}_{k+1}\,, \tag{11}$$

where $\overline{\mathbf{f}}_{k+1} \in \mathbb{R}^{n_x}$ indicates the gap between the nonlinear rollout and the next state. Keeping these gaps opened during the first iterations of the solver is what gives the solver a better globalization capability [11], [10]. Once these gaps are closed for all the nodes, *i.e.*, $\overline{\mathbf{f}}_{k+1} = 0$ for $k = 0 \dots N-1$, the FDDP algorithm becomes the DDP explained in the previous section.

1) Derivatives and backward pass: Because we allow the existence of gaps during the solving process, we have to modify the three steps of the DDP algorithm accordingly. First, we will compute the derivatives of the optimal cost-to-go at $\mathbf{x}_{k+1}$, *i.e.*, $\mathbf{V}_{\mathbf{x}_{k+1}}$. However, in the backward pass we need them at $\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$. Making use of the Hessian and the gap at step $k+1$, we can modify the derivative as

$$\mathbf{V}^+_{\mathbf{x}_{k+1}} = \mathbf{V}_{\mathbf{x}_{k+1}} + \mathbf{V}_{\mathbf{x}\mathbf{x}_{k+1}}\overline{\mathbf{f}}_{k+1}\,, \tag{12}$$

where the Hessian remains constant since V is assumed to be a quadratic function. Having this in mind, the backward pass in (6) should be modified accordingly.

2) Forward pass: In DDP, $\mathbf{U}$ and $\mathbf{X}$ are updated sequentially. That is, α determines how $\mathbf{U}$ is updated (10b) and then, by performing a nonlinear rollout we compute the trajectory $\mathbf{X}$ (10c). Due to the introduction of the gaps, in the FDDP technique both trajectories are updated simultaneously. Thus, (10c) has to be modified so that it captures also the evolution of the gaps, *i.e.*,

$$\hat{\mathbf{x}}_{k+1} = \mathbf{f}(\hat{\mathbf{x}}_k, \hat{\mathbf{u}}_k) - (1-\alpha)\overline{\mathbf{f}}_{k+1}\,, \tag{13}$$

where the new value of the gap is given by $\hat{\overline{\mathbf{f}}}_{k+1} = (1-\alpha)\overline{\mathbf{f}}_{k+1}$. Again, the α parameter indicates the length of the step taken at every iteration. It also affects to the manner in which the gaps are closed throughout the solving process (see [11]). A full step ($\alpha = 1$) closes the gaps.

III. SQUASH-BOX FDDP

In this section we describe our novel method called squash-box FDDP (sb-FDDP). We start by presenting the SF used in this work (Section III-A). In Section III-B, we briefly introduce a naive approach based on squashing functions. This method introduces nonlinearities that produce a poor sublinear convergence when compared to [12]. To tackle this, we propose a penalty method that consists of two loops (Section III-C).

A. Squashing function

A squashing function is basically a sigmoid with the following properties

$$g(s) : \mathbb{R} \to (\underline{g}, \overline{g})\,, \tag{14}$$
$$g'(s) \geq 0 \tag{15}$$

with

$$\underline{g} = \lim_{s\to-\infty} g(s) \text{ and } \overline{g} = \lim_{s\to\infty} g(s)\,. \tag{16}$$

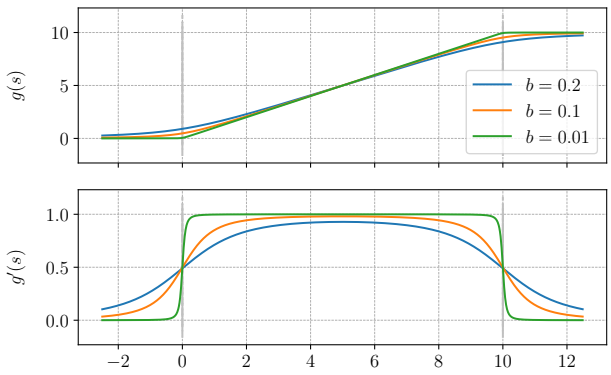

Fig. 2: Squashing functions (top) and their first derivatives (bottom) for different smoothness values $b \in \{0.2, 0.1, 0.01\}$. The areas beyond the bounds where the SF becomes flat are known as the *plateaus*. Notice that the control interval is $\overline{u} - \underline{u} = 10$, giving widths of the smoothed corners $a \in \{2.0, 1.0, 0.1\}$.

By constructing a proper SF so that $\underline{g} = \underline{u}$ and $\overline{g} = \overline{u}$, the control bounds are now guaranteed by construction. Notice that, without loss of generality, we expressed the function g for a single control input. In order to consider the whole $\mathbf{u}$, we just need to apply it component-wise to $\mathbf{u}$. We write $\mathbf{u} = \mathbf{g}(\mathbf{s})$ for simplicity. Likewise, we will note $\mathbf{S} = [\mathbf{s}_0, \cdots, \mathbf{s}_{N-1}]$ the trajectory of the squashing variables, having then $\mathbf{U} = \mathbf{g}(\mathbf{S})$.

Now, the aim is to find an expression for g that is bounded (14), monotonic (15) and, importantly for the convergence of FDDP, continuously differentiable. For the rest of this paper, our sigmoid function has the form (see Fig. 2),

$$g(s) = \frac{1}{2}(\underline{u} + \sqrt{a^2 + (s - \underline{u})^2}) + \frac{1}{2}(\overline{u} - \sqrt{a^2 + (s - \overline{u})^2}) , \tag{17}$$

where the parameter a, which has the units of u, defines the sharpness or smoothness of the function around the saturation points, and corresponds roughly to the width of the smoothed corners connecting the central linear segment with the saturated plateau. Note that when a approaches 0, $g(s)$ becomes the saturation function, *i.e.*,

$$\lim_{a \to 0} g(s) = \begin{cases} \underline{u} & s < \underline{u} , \\ s & \underline{u} \leq s \leq \overline{u} , \\ \overline{u} & \overline{u} < s . \end{cases} \tag{18}$$

To easily cope with different control saturation intervals $[\overline{u}, \underline{u}]$, we rather define a through a ratio $b \in (0, 1]$ of the interval, i.e.

$$a = b(\overline{u} - \underline{u}) \ . \tag{19}$$

This way, b corresponds (see Fig. 2) to the normalized size of the smoothed corner relative to the control interval.

B. Squashed FDDP problem

By including the SF (17) $\mathbf{U} = \mathbf{g}(\mathbf{S})$ in the problem (1), the optimal control problem becomes unbounded[1]:

$$\begin{aligned} \min_{\mathbf{S}, \mathbf{X}} \quad & \textstyle\sum_{k=0}^{N-1} l_k(\mathbf{x}_k, \mathbf{g}(\mathbf{s}_k)) + l_N(\mathbf{x}_N) , \\ \text{s.t.} \quad & \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{g}(\mathbf{s}_k)) , \\ & \mathbf{x}_0 = \mathbf{x}(\mathbf{0}) , \end{aligned} \tag{20}$$

where we now focus on finding the input sequence $\mathbf{S}$, whose entries can take any value in $\mathbb{R}$. Using the chain rule, the Riccati equations are expressed in terms of $\mathbf{X}$ and $\mathbf{S}$ as

$$\mathbf{Q_s} = \mathbf{l_s} + \mathbf{g_s}^\top \mathbf{Q_u} = \mathbf{l_s} + \mathbf{g_s}^\top (\mathbf{l_u} + \mathbf{f_u}^\top \mathbf{V_x^+}) , \tag{21a}$$

$$\mathbf{Q_{sx}} = \mathbf{g_s}^\top \mathbf{Q_{ux}} \tag{21b}$$

$$= \mathbf{g_s}^\top (\mathbf{l_{ux}} + \mathbf{f_u}^\top \mathbf{V'_{xx}} \mathbf{f_x}) , \tag{21c}$$

$$\begin{aligned} \mathbf{Q_{ss}} &= \mathbf{l_{ss}} + \mathbf{Q_u} \cdot \mathbf{g_{ss}} + \mathbf{g_s}^\top \mathbf{Q_{uu}} \mathbf{g_s} \\ &= \mathbf{l_{ss}} + (\mathbf{l_u} + \mathbf{f_u}^\top \mathbf{V_x^+}) \cdot \mathbf{g_{ss}} + \\ &+ \mathbf{g_s}^\top (\mathbf{l_{uu}} + \mathbf{f_u}^\top \mathbf{V'_{xx}} \mathbf{f_u}) \mathbf{g_s} , \end{aligned} \tag{21d}$$

where $\mathbf{g_s}$ is the diagonal Jacobian matrix of the SF, and $\mathbf{g_{ss}}$ is a sparse cubic tensor with its non-zero values placed at the diagonal of the cube[2]. The barrier gradient and Hessian matrix with respect to the squashing variables $\mathbf{S}$ are represented by $\mathbf{l_s}$ and $\mathbf{l_{ss}}$, respectively. Note that we use the Gauss-Newton approximation in (21), and therefore the second partial derivatives of the dynamics have been omitted.

C. Quadratic penalization method

We could solve the unbounded optimal control in (20) by using the FDDP algorithm. The smoothness b should be high to ensure convergence, and some kind of regularization should be imposed on $\mathbf{s}$ to prevent it from departing to infinity upon saturation. In such cases, the vanishing derivatives of the SF on the plateaus (see Fig. 2) negatively impact the convergence rate of the solver. We propose a penalty method with an outer loop and a quadratic barrier to greatly improve this naive solution.

1) Quadratic barrier: To improve the convergence speed, we add a quadratic barrier that attracts the squashing variable towards the smoothed corner of the SF, *i.e.*, away from the plateau. Since the size of this smoothed region is a, given by (19) and controlled by the parameter b, we use the same a to control the width of the quadratic barrier, *i.e.*,

$$l_{barr}(\mathbf{s}) = \sum_{i=1}^{n_u} \begin{cases} w_i (\frac{s_i - \underline{u}_i}{a})^2 & s_i < \underline{u}_i , \\ 0 & \underline{u}_i \leq s_i \leq \overline{u}_i , \\ w_i (\frac{s_i - \overline{u}_i}{a})^2 & \overline{u}_i < s_i , \end{cases} \tag{22}$$

where the subindex i represents the element-wise components of $\mathbf{s}$, $\underline{\mathbf{u}}$ and $\overline{\mathbf{u}}$, and $\mathbf{w} = [w_i]$ are the weights given to the respective quadratic functions. Fig. 3 shows how the width of the barrier follows the degree of smoothness of the SF corner.

[1]Without loss of generality, our discourse assumes a FDDP solver, although the same is applicable to a DDP solver; however and as said, FDDP presents better globalization capabilities than DDP [11].

[2]Note that this tensor product is equivalent to a matrix product due to its diagonal structue

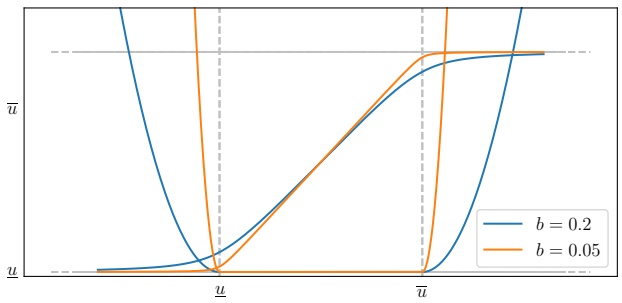

Fig. 3: SF and its associated quadratic barrier for $b = 0.2$ and $b = 0.05$. The barrier attraction towards the smoothed corner of the SF is adapted to the width of the smoothed area.

2) Penalization method: We have the barrier that keeps s away from the plateau, *i.e.*, inside the smoothed corner, improving convergence rates. However, for very smooth SFs (high b value), all $u = g(s)$ with s in the smooth corner will not saturate the controls and, therefore, the solver will not exploit the full range of control commands. Since we require both good solver convergence and good control saturation, we propose a method inspired by the penalty methods [16, Ch. 17]. Our approach involves two nested loops (Algorithm 1). The inner loop runs an FDDP solver with a SF and a quadratic barrier. The outer loop is in charge of modifying b, hence the smoothness of the SF and the width of the barrier. The basic idea is to start with a large b and progressively decrease it. That is, the solver starts with a smooth SF and a wide quadratic barrier (high b value), and each outer loop imposes a sharper SF with a steeper barrier. Eventually, as b gets smaller the problem converges to a true saturated problem (see (16)). Contrary to common penalty methods, where this convergence is required to reach very tight values to ensure the constraints, our method can exit much earlier, *i.e.*, with a last b not extremely small. This is due to the fact that it satisfies the control bounds by construction of the SF. Moreover, during the initial outer loops it does not require very precise solutions, so our algorithm starts with a loose exit criterion τ (Section III-C.4) and reduces it progressively as outer iterations advance.

3) Trajectory warm start and gap contraction: We warm start the FDDP solver at each outer iteration with the solution of the previous iteration (lines 5-6). At the end of each outer iteration, the SF becomes sharper due to the decrease of b. This might make some variables $\mathbf{s}_k$ fall onto the plateau area of the new SF. In such cases the new steeper quadratic barrier will rapidly take these variables back to the smoothed corner. A-priori better alternatives such as $\mathbf{S}_{ini} \leftarrow g(\mathbf{S})$, which brings the variables on the plateau back to the smoothed area, does not show a significant improvement.

In practice, when the resulting trajectory from the inner loop $[\mathbf{S}_j, \mathbf{X}_j]$ is feasible (Section II-B), this assignment often results in an infeasible trajectory. Thus, the gaps are usually opened at the beginning of every outer iteration. The next inner loop will close them again.

Algorithm 1: squash-box FDDP

Data:
Initial guess: $(\mathbf{X}_{ini,0}, \mathbf{S}_{ini,0})$
Initial barrier weight: ω
Squashing values: $\{b_0, \dots, b_M\}$, with $b_j > b_{j+1}$
Converging thresholds: $\{\tau_0, \dots, \tau_M\}$ with $\tau_j > \tau_{j+1}$
Gap threshold: $\overline{f}_{th}$

1 **for** $j = 0$ **to** M **do**
2 $\mu_j \leftarrow$ update barrier weight from (b_j, w)
3 warm start FDDP with $(\mathbf{X}_{ini,j}, \mathbf{S}_{ini,j}, \mu_j)$
4 run FDDP with stopping criteria $(\tau_j, \overline{f}_{th})$
5 $\mathbf{X}_{ini,j+1} \leftarrow \mathbf{X}_j$
6 $\mathbf{S}_{ini,j+1} \leftarrow \mathbf{S}_j$
7 **if** $[\mathbf{S}_M, \mathbf{X}_M]$ *not feasible* **then**
8 warm start DDP: $(\mathbf{X}_M, \mathbf{S}_M, \mu_M)$
9 run DDP with tolerance convergence τ_M

4) Stopping criteria: One common stopping criteria in DDP is to check whether a change in the controls can still produce a considerable change in the cost, i.e.:

$$\sum_{k=0}^{N-1} ||\mathbf{Q}_{\mathbf{u},k}||^2 < \tau_j \,, \tag{23}$$

and, when using the SF, the criterion in (23) considers $\mathbf{Q}_\mathbf{s}$ instead of $\mathbf{Q}_\mathbf{u}$. Note that, using (21a), this criterion becomes

$$\sum_{k=0}^{N-1} ||\mathbf{l}_{\mathbf{s},k} + \mathbf{g}_{\mathbf{s},k}^\top \mathbf{Q}_{\mathbf{u},k}||^2 < \tau_j \,. \tag{24}$$

However, this is problematic at the nodes where the $\mathbf{s}_k$ is outside the bounds, i.e. $\mathbf{l}_{\mathbf{s},k} \neq \mathbf{0}$. The reason is that the two gradients $\mathbf{l}_{\mathbf{s},k}$ and $\mathbf{g}_{\mathbf{s},k}^\top \mathbf{Q}_{\mathbf{u},k}$ are pointing at opposite directions, i.e. the first term is trying to keep the $\mathbf{s}_k$ variable inside the quasi-linear region while the second tries to decrease the original problem cost by moving $\mathbf{s}_k$ away. Intuitively, with (24), we are asking the solver to find an input $\mathbf{s}_k$ that makes these two large terms equal up to a tiny threshold τ_j. This results in very small steps with no actual improvement in the cost, producing a high number of iterations without significant performance gain.

To overcome this situation, we follow the stopping criterion proposed in [10]. This criterion considers the relative cost improvement in every iteration as well as the sum of the gaps over the entire trajectory. Thus, the new stopping criterion for the FDDP solver is,

$$\frac{|J_j(\mathbf{x}_0, \mathbf{S}_j) - J_{j-1}(\mathbf{x}_0, \mathbf{S}_{j-1})|}{J_j(\mathbf{x}_0, \mathbf{S}_j)} < \tau_j \tag{25}$$

$$\text{and} \quad \sum_{k=0}^{N-1} ||\overline{\mathbf{f}}_{k,j}|| < \overline{f}_{th} \,. \tag{26}$$

It is worth noticing that, with this criterion and since $\overline{f}_{th}$ is not exactly zero, it is possible that the algorithm exits the outer loop with an infeasible trajectory $[\mathbf{S}_j, \mathbf{X}_j]$. In such

case, we consider an extra run of a DDP solver, which closes the gaps at the first iteration (lines 7-9).

IV. Results

We first compare our proposed sb-FDDP algorithm with two existing methods. Concretely, we consider 1) a naive DDP approach based on only a squashing function, and 2) the Box-DDP algorithm proposed in [12]. Our results show the effect of using the SF and the quadratic barrier, and the importance of the outer loop. We test all solvers with a wide range of examples developed in the optimal control library Crocoddyl [11]. We report the generated motions in the accompanying video. For more details about the examples check the Github repository[3].

A. Case studies

1) Quadcopter: We use the Iris quadcopter to perform two tasks: 1) cross a wall through a vertical narrow passage, 2) cross the wall and come back passing on top of the wall. These motions are achieved by specifying different way-points, which are reached at different instants throughout the trajectory (see the way-points represented as frames in Fig. 1). Each of them considers a pose in $\mathbb{SE}(3)$ and an associated target velocity in the tangent space. In order to impose the pose and the motion (expressed by the way-point) at a given node, we add a cost that penalizes the error of the quadcopter state with respect to the way-point. The control inputs of the system are the thrusts produced by each motor-propeller set. We configured the controls to move within a range of 0.1 to $10.3\,\mathrm{N}$. For all the cases, we do not warm-start the solvers.

2) Biped walk: We use the Talos legs with its dynamics properties (see Fig. 4). We formulate the optimal control problem for one single walking cycle, *i.e.*, we consider two steps, with a double support phase. To better benchmark our approach, we reduce the joint torque limits by half and use the default posture and quasi-static torques to warm-start the solvers. For details about the used contact and impulse dynamics, we refer the reader to the existing literature, *e.g.* [17], [11].

3) Quadruped: We use the ANYmal robot for which we generate a jumping motion (see Fig. 5). Again, we reduce the torque and the joint velocity limits to $32\,\mathrm{N\,m}$ and $7.5\,\mathrm{rad/s}$, respectively. The latter is imposed through soft constraints as quadratic penalizations in the cost function. Like in the previous example we have warm-started the solver with the quasi-static solution.

4) Humanoid: We perform a whole body experiment with the Talos robot (see Fig. 6). It consists of first reaching a goal with the hand, and then standing on one foot while balancing the whole body. As in the previous case, we limit the maximum torque to 40% of the full torque capacity, and warm-start the solver with the quasi-static solution.

[3]https://github.com/loco-3d/crocoddyl.

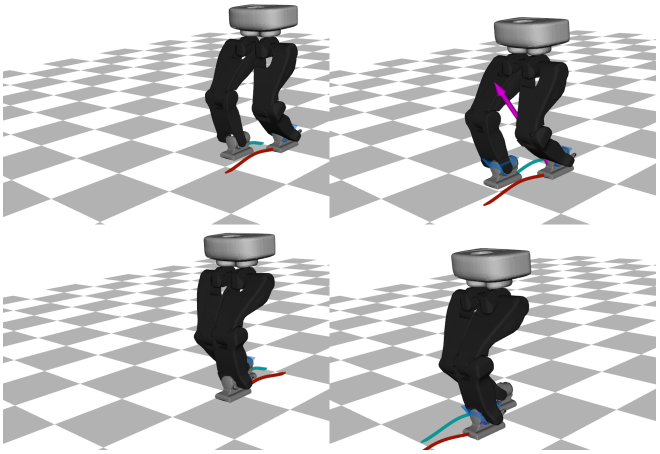

Fig. 4: Talos legs performing several steps. Image order: column-wise, from top to bottom and left to right.

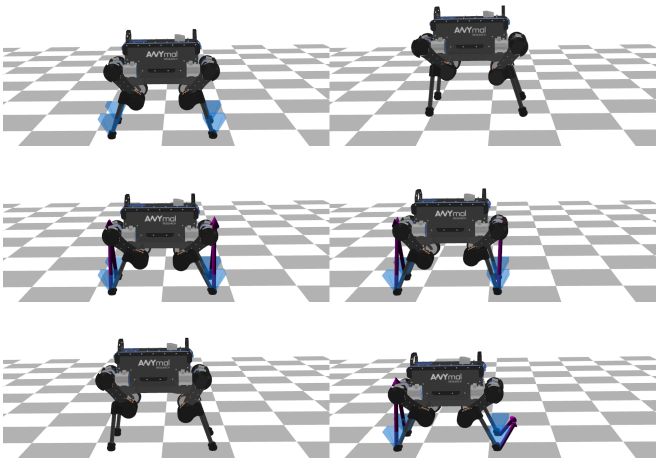

Fig. 5: ANYmal quadruped performing a jump. Image order: column-wise, from top to bottom and left to right.

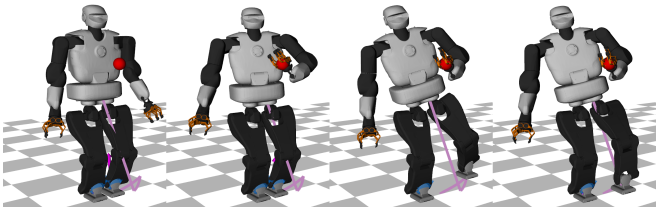

Fig. 6: Talos humanoid performing the balancing operation. From left to right: place the left hand at the goal and, without moving the hand, lift the left leg and finally perform a balancing maneuver.

B. Performance of the squash-box FDDP

We report the solvers behavior in Fig. 7. For the sb-FDDP, we consider two outer loops with $b = \{0.1, 0.05\}$ and $\tau = \{1 \times 10^{-2}, 1 \times 10^{-3}\}$ across all the different problems. We use different values of w and $\overline{f}_{th}$ because they are problem dependent. For the naive squash approach, we set $b = 0.05$ and $\tau = 1 \times 10^{-3}$. We use the same initial guess for all the solvers. In all the cases, sb-FDDP outperforms the other two solvers, not only in how fast they converge but also when we look at the cost of the solution. As in other penalty methods, the solving speed is determined by the inner solver and how

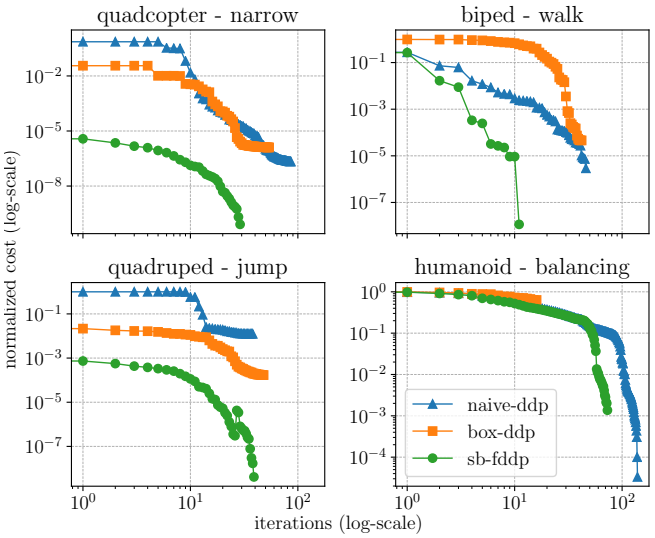

Fig. 7: Cost progression for the different studied problems. Our sb-FDDP algorithm converges to a better solution with fewer iterations than the other two methods. The jump problem cannot be solved by the others solvers. The Box-DDP stops prematurely in the humanoid balancing problem

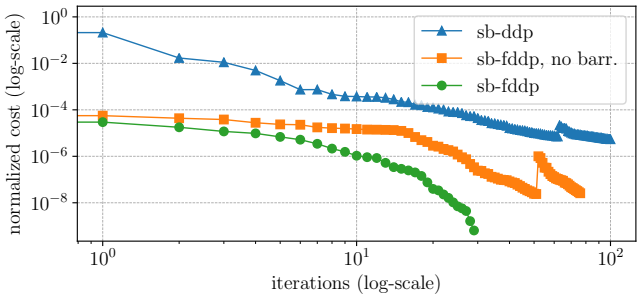

Fig. 8: Cost progression for the quadcopter narrow passage solved with the sb-DDP (blue) and the sb-FDDP solvers. The use of the FDDP as the inner solver considerably improves the results (orange). The addition of the barrier further improves the method performance (green). The first cost reported for each solver is the value after the first iteration.

it is implemented. Thus, we refer the reader to [11] for more details about it.

There are some cases where the naive squash DDP and the Box-DDP do not converge to the solution (*e.g.* quadrupedal jump). This is due to the poor globalization capability of the DDP-like algorithms. The fact that they do not allow infeasible iterations makes them very sensitive to poor initial guesses. Instead, our algorithm converges faster to a better solution partly thanks to the proposed FDDP step in the inner loop.

1) FDDP choice and quadratic barrier: To understand the benefits of the FDDP step, we substitute it with a DDP step (named sb-DDP). In Fig. 8, we can see the difference between the sb-FDDP and the sb-DDP algorithms. We also include a comparison with a sb-FDDP solver without the inclusion of the proposed barrier method. Clearly, the sb-FDDP outperforms both approaches.

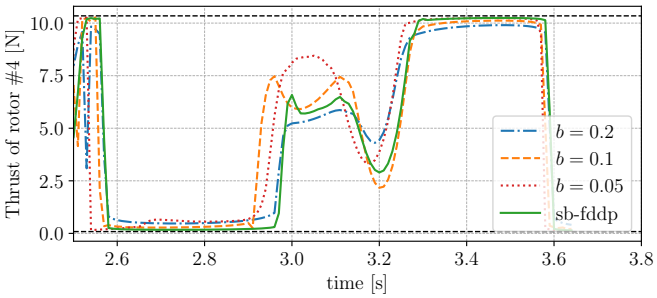

Fig. 9: Final section of the thrust trajectory obtained for a single motor of a quadcopter for the experiment in Fig. 1. We compare trajectories for one single outer iteration with a convergence threshold of $\tau = 1 \cdot 10^{-4}$. In green it is reached the same final values for b and τ but following the progression $b = \{0.2, 0.1, 0.05\}$ and $\tau = \{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$. Notice the tight saturation values of the sb-FDDP solution (green). Performance indicators are reported in Table I.

TABLE I: Comparison metrics for a single outer loop with $\tau = 1 \times 10^{-4}$ for various b values. The sb-FDDP considers $b = \{0.2, 0.1, 0.05\}$ and $\tau = \{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$. The number of iterations reported for the sb-FDDP case is the addition of all inner iterations.

	Quadcopter		Quadruped	
b	Iter.	Cost	Iter.	Cost
0.2	60	1.2131	32	6.28×10^3
0.1	55	1.2430	46	6.08×10^3
0.05	209	1.6748	44	5.97×10^3
sb-FDDP	98	1.1679	40	5.98×10^3

2) Outer loop: In Fig. 9 we see that the optimized controls resulting from smaller b are significantly closer to the saturation. As the last outer loop of the sb-FDDP considers the smallest b, we can also see that it almost saturates the controls.

In Table I, we report relevant metrics related to the experiments shown in Fig. 9, and an analogous experiment with the quadruped jump case. For the same final $b = 0.05$ value and same convergence $\tau = 1 \times 10^{-4}$, we observe that our algorithm achieves a faster solution by performing several outer iterations when it is approaching the final b value. The costs are very similar in the case of the quadruped and significantly better in the case of the quadcopter.

The early outer iterations are useful since they provide a good warm-start for the following ones. Additionally, reducing b opens the gaps. With the gaps opened, the inner FDDP often has better exploration capabilities. Fig. 10 shows the gap evolution for the different case studies. It is of special interest the case of the humanoid. Practically, the first outer iteration has the gaps fully closed. Therefore, the cost can barely be reduced (see Fig. 7). At iteration 32 the first outer loop finishes and b is updated accordingly. As a result, the gaps open and after approximately an exploration phase of 20 iterations, the gaps begin to close and the cost decreases rapidly.

Finally, Fig. 11 supports the decision of choosing only two outer loops —this is a trade-off decision as discussed

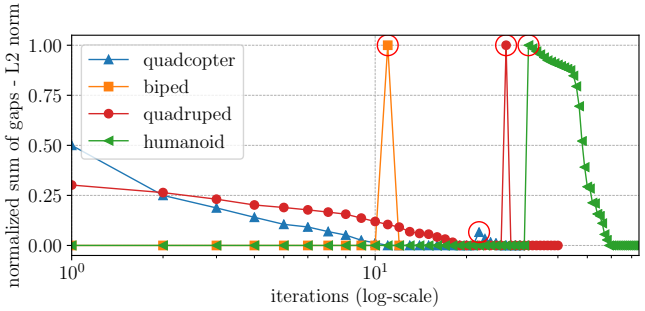

Fig. 10: L2 norm of the gaps for the whole trajectory. Due to the update of the b parameter, the gaps are opened at the beginning of every outer iteration. The iteration where the update occurs is marked with a red circle.

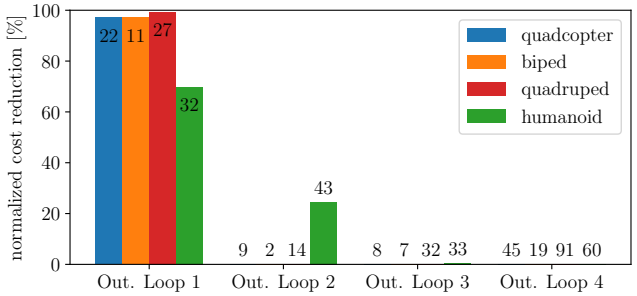

Fig. 11: Cost reduction produced at every outer-loop iteration. The numbers over the bars indicate the iterations that the inner solver has to perform in order to converge. Sequence of squashing values and convergence are $b = \{0.1, 0.05, 0.025, 0.0125\}$ and $\tau = \{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}, 1 \times 10^{-5}\}$. Note that the main cost reduction occurs in the first outer iteration. The other ones help to get the controls closer to the saturation values.

in Section V. We evaluate four outer loops and show that the actual cost reduction happens to be at the first outer loop (with the exception perhaps of the humanoid problem) but we still need the second outer loop to allow the controls to approach saturation. Further iterations have less impact on the final trajectory.

V. Conclusion

We proposed a method that modifies the control-limited optimal control problem in such a way that we could treat it using unconstrained solvers (*e.g.* FDDP). The solver modification uses, primarily, the computation of the controls through a squashing function (SF). In this manner, the solution to the optimization problem is found using the SF input as the decision variable. To improve the poor convergence speed as reported in [12], we proposed to include a quadratic barrier so that the decision variables do not fall far from the quasi-linear region. Additionally, our approach considers an outer loop that is in charge of modifying the problem that FDDP solves in the inner loop. As the outer loop progresses, the modified problem converges to the original one. This approach has proven to effectively solve a variety of optimal control-bounded problems, even those where other solvers failed.

The number of outer loops is a trade-off between mathematical perfection and pragmatism. The SF modifies the original problem and, therefore, the obtained solution is sub-optimal. With multiple outer iterations and as b gets smaller, both problems eventually converge and so does the solution to an optimum. We have shown that after only a very small number of outer iterations, and even if the controls did not reach the bounds accurately, the change in the final trajectory can seldom be noticed, concluding in practice that a couple of outer loops suffice to reach the optimal trajectory with sufficiently saturated controls.

References

[1] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. USA: Cambridge University Press, 2009.

[2] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization," *SIAM Rev.*, 2005.

[3] R. H. Byrd, J. Nocedal, and R. A. Waltz, "KNITRO: An integrated package for nonlinear optimization," in *Large Scale Nonlinear Optimization, 35–59, 2006*, 2006.

[4] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 2006.

[5] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2016.

[6] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback mpc for torque-controlled legged robots," in *IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, 2019.

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

[8] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *Int. J. of Cntr.*, 1966.

[9] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems," in *ICINCO*, 2004.

[10] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A Family of Iterative Gauss-Newton Shooting Methods for Nonlinear Optimal Control," in *IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, 2018.

[11] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2020.

[12] Y. Tassa, N. Mansard, and E. Todorov, "Control-Limited Differential Dynamic Programming," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2014.

[13] Z. Xie, C. K. Liu, and K. Hauser, "Differential dynamic programming with nonlinear constraints," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2017.

[14] G. Lantoine and R. Russell, "A Hybrid Differential Dynamic Programming Algorithm for Robust Low-Thrust Optimization," in *Astrodyn. Special. Conf. Exhibit*, 2008.

[15] T. A. Howell, B. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, 2019.

[16] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, 1999.

[17] 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.