PROXDDP: Proximal Constrained Trajectory Optimization

Wilson Jallet, Antoine Bambade, Etienne Arlaud, Sarah El-Kazdadi, Nicolas Mansard, Justin Carpentier

PROXDDP: Proximal Constrained Trajectory Optimization

Wilson Jallet[*,1,2], Antoine Bambade[1], Etienne Arlaud[1], Sarah El-Kazdadi[1], Nicolas Mansard[2] and Justin Carpentier[1]

***Abstract*—Trajectory optimization (TO) has proven, over the last decade, to be a versatile and effective framework for robot control. Several numerical solvers have been demonstrated to be fast enough to allow recomputing full-dynamics trajectories for various systems at control time, enabling model predictive control (MPC) of complex robots. These first implementations of MPC in robotics predominantly utilize some differential dynamic programming (DDP) variant for its computational speed and ease of use in constraint-free settings. Nevertheless, many scenarios in robotics call for adding hard constraints in TO problems (e.g., torque limits, obstacle avoidance), which existing solvers, based on DDP, often struggle to handle. Effectively addressing path constraints still poses optimization challenges (e.g., numerical stability, efficiency, accuracy of constraint satisfaction) that we propose to solve by combining advances in numerical optimization with the foundational efficiency of DDP. In this article, we leverage proximal methods for constrained optimization and introduce a DDP-like method to achieve fast, constrained trajectory optimization with an efficient warm-starting strategy particularly suited for MPC applications. Compared to earlier solvers, our approach effectively manages hard constraints without warm-start limitations and exhibits commendable convergence accuracy. Additionally, we leverage the computational efficiency of DDP, enabling real-time resolution of complex problems such as whole-body quadruped locomotion. We provide a complete implementation as part of an open-source and flexible C++ trajectory optimization library called ALIGATOR. These algorithmic contributions are validated through several trajectory planning scenarios from the robotics literature and the real-time whole-body MPC of a quadruped robot.**

***Index Terms*—Optimization and Optimal Control, Legged Robots, Model-Predictive Control**

I. INTRODUCTION

TRAJECTORY OPTIMIZATION is an efficient and generic approach for controlling complex dynamical systems such as robots. It is a principled framework for describing desired behaviors and generating motion. A workhorse in modern robotics, it has become a crucial ingredient in both kinodynamic planning and model predictive control (MPC) over the past decade, enabled by the increasing performance of computer chips and algorithmic enhancements alleviating previous computational bottlenecks. Notably, recent progress in both software and hardware has enabled the real-time computation of numerical quantities commonly involved in trajectory optimization (e.g., rigid-body kinematics, dynamics and their derivatives [1]–[3]).

[*]Corresponding author. wjallet@laas.fr
[1]Inria - Département d'Informatique de l'École normale supérieure, PSL Research University.
[2]LAAS-CNRS, 7 av. du Colonel Roche, 31400 Toulouse.

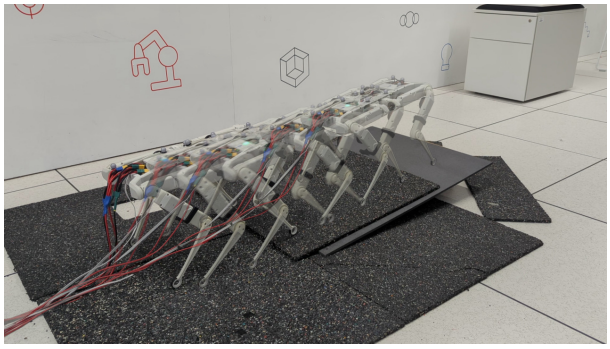


Fig. 1. **Solo-12 walking on an *unmodelled* slope** using the whole-body MPC framework based on primal-dual augmented Lagrangian techniques.

Optimal control problems (OCPs) are, by nature, infinite-dimensional optimization problems, which are largely not solvable in closed form. However, they can be solved numerically. On the one hand, there are *indirect* methods for OCPs, based on deriving their optimality conditions [4]. On the other hand, there are *direct* methods [5] which transcribe OCP problems into nonlinear programs (NLPs) of finite dimensions.

Direct methods, whichever the method of transcription, attempt resolution by utilizing a nonlinear programming approach, either leveraging general-purpose and off-the-shelf solvers such as IPOPT [6] or SNOPT [7], or a more tailored solution. Several approaches are considered in the literature to solve them in practice. We will argue why, in the robotics community, differential dynamic programming-based solvers are seen as a promising research direction, notably for deploying receding horizon control schemes for real-time robot control. One transcription method is *collocation*, which approximates the OC problem using a finite-dimensional basis functions such as polynomials. Another transcription method is *shooting methods*. They use a discretization of the system dynamics through numerical integration, which is generic, efficient, and easy to implement. For these shooting methods, there exist structure-exploiting solvers leveraging Riccati recursion [8]–[11], such as the differential dynamic programming (DDP) [12] algorithm. DDP is one of the earliest such methods and a reference in nonlinear trajectory optimization, known to have quadratic convergence [13], and has several variants such as the iterative linear quadratic regulator (iLQR) [14], [15] (which is in fact a simplified version of DDP). DDP methods are a popular choice in robotics as they are relatively easy to implement with performance in mind [15], which is a crucial requirement in the field. They have enabled complex applications such as whole-body nonlinear MPC in robotics [16], [17].

However, standard Riccati recursion, and by extension DDP, is not meant for constrained problems. A classic approach to handle this in robotics is to use soft penalty methods [16], [18]. The issue with this approach is having to tune the penalty weights to avoid ill-conditioning issues while adequately enforcing the constraints and still getting the desired behavior in the system. After years of practice with advanced MPC in various experimental setups, we firmly believe that stricter enforcement of constraints is required to bring MPC on complex robots to move outside the labs. The central objective of this paper is to enrich DDP with a sound method for handling hard constraints.

Several attempts have been made in that direction. Prior work has extended the Riccati recursion to the equality-constrained case [19]–[21] using factorizations such as LU or QR decompositions; these methods can be used for solving subproblems in DDP or sequential quadratic programming (SQP) schemes. Unfortunately, these approaches do not tackle inequality constraints and would need to be combined with another approach for this purpose. Furthermore, the factorizations at each stage of the algorithm require additional assumptions on the problem constraints (e.g. positive definite reduced Hessians at every stage), which an off-the-shelf sparse solver such as Cholmod [22] would not need when applied to the entire problem. Projection approaches have also been used for box-inequality constraints such as torque limits [23], [24] by solving a quadratic program (QP) in a modified DDP backward pass. Other approaches for inequality constraints, such as interior-point methods, have been employed in HPIPM [25], [26] and by Pavlov [27] (the two former focusing on for QP- and QCQP-like optimal control problems). Although interior-point methods can obtain good precision in nonlinear programming settings [6], the approach is not always suitable for MPC applications due to having limited warm-starting capabilities [28]. Closely related are barrier methods [29], which have been used in trajectory optimization by [30], [31] and MPC by Grandia [32] using a "relaxed" barrier approach. However, these barrier approaches do not solve inequality constraints exactly: they penalize them more or less aggressively, leaving some margin of error in the optimality conditions or letting through infeasible trajectories, and like penalty methods, they can suffer from ill-conditioning. The aforementioned work still leaves us with the need for a TO solver to deal with generic constraints with less need for penalty tuning, more accurate constraint satisfaction, and better warm-starting capacities, especially for real-time robot control.

In nonlinear constrained optimization, methods based on the augmented Lagrangian (AL) [33], [34] are a broad class of exact penalty methods that have been popular with implementations such LANCELOT [35]. Their application to trajectory optimization problems has been discussed [36], [37], namely with methods such as ALTRO [38], [39] for equality and inequality constraints. Sleiman et al. [40] have also employed AL methods for integrating inequality constraints in continuous-time DDP, studying multiple smooth and non-smooth penalty choices. One difficulty with AL methods is that they can be numerically brittle when encountering ill-conditioned systems of equations, and hence, obtaining good convergence in AL methods has been an issue: [38], [41] have to use a second, projection-based stage after AL to refine the solutions to convergence. Recently, [42] leveraged a penalty-scheduling [43] as well as a primal-dual system of equations similar to [44], [45] to ensure more reliable convergence for these methods in the equality-constrained case. In our prior work [46], [47], we have highlighted the link between such AL approaches and proximal methods in optimization, and we argue in this paper that they provide a strong candidate for a framework integrating AL with DDP-based solvers.

In this article, we propose a primal-dual proximal-point formulation for constrained trajectory optimization and derive an efficient structure-exploiting algorithm combining AL and DDP. In Section II, we first provide a background on nonlinear optimization and augmented Lagrangian methods, with an accent on primal-dual proximal methods. Section III recalls the principles behind trajectory optimization with DDP, which will be useful in formulating our contribution, the proximal DDP algorithm presented in Section IV. Our method is provided with its own C++ implementation, a novel trajectory optimization library named ALIGATOR, discussed in Section VI, with experiments showcasing it and the method's capabilities in Section VII.

This article is an extended version of our previous work on implicit differentiable dynamic programming [46] and constrained differentiable dynamic programming [47]. Compared to our previous work, this article provides (i) a unified view on combining primal-dual proximal augmented Lagrangian techniques within DDP-based solvers, (ii) a more appropriate merit function inspired by [48] for better globalization, (iii) an open-source and highly efficient implementation of the proposed solution within a new C++ framework for trajectory optimization, as well as (iv) further additional experiments, especially some real-world MPC control applications of the proposed approach with real-time control of the SOLO quadrupedal robot.

II. Background on nonlinear optimization and augmented Lagrangian methods

This section provides a general overview of nonlinear optimization and augmented Lagrangian methods. These optimization techniques are at the core of the approach developed in this article. This approach is similar to prior work leveraging semi-smooth Newton methods for the augmented Lagrangian [49] (for general NLPs) and [44], [45], [48] (for QPs). Its novelty is the use of a primal-dual semi-smooth Newton method for inequality constraints in the augmented Lagrangian for general NLPs (used for QPs by [45], [48]), along with the use of the parameter update rule of [43] in this setting. This specific variant of the augmented Lagrangian method is used as a basis for our main contribution in Section IV. A further detailed comparison with prior work is provided at the end of this section (Subsection II-E).

A. Nonlinear constrained optimization

Trajectory optimization problems may often be transcribed as nonlinear programming problems (NLPs). NLPs with both

equality and inequality constraints correspond to optimization problems of the form:

$$\min_{z\in\mathcal{Z}} \; f(z) \tag{1a}$$
$$\text{s.t. } g(z) = 0, \tag{1b}$$
$$h(z) \leqslant 0, \tag{1c}$$

with $z \in \mathcal{Z}$ the optimization variable, where $\mathcal{Z} = \mathbb{R}^{n_z}$, $f\colon \mathcal{Z} \to \mathbb{R}$ is the twice-differentiable cost function and $g\colon \mathcal{Z} \to \mathbb{R}^{n_e}$ (resp. $h\colon \mathcal{Z} \to \mathbb{R}^{n_i}$) are the differentiable equality (resp. inequality) constraints.

The first-order necessary conditions of optimality are given by the so-called Karush-Kuhn-Tucker (KKT) conditions [29], which is that for an optimal point z, there exist Lagrange multipliers (λ, ν) satisfying:

$$\nabla_z f(z) + \left(\frac{\partial g}{\partial z}\right)^\top \lambda + \left(\frac{\partial h}{\partial z}\right)^\top \nu = 0, \tag{2a}$$
$$g(z) = 0, \tag{2b}$$
$$h(z) \leqslant 0 \perp \nu \geqslant 0. \tag{2c}$$

The *complementarity conditions* (2c), mean that for every i, $h_j(z) \leqslant 0$, $\nu_i \geqslant 0$ and either one or the other is zero. When $h_j(z) = 0$, $\nu_j > 0$ and the i^{th} constraint is said to be *active*.

The nonlinear programming problem can be seen as that of finding a saddle-point [29], [50] of the so-called Lagrangian function:

$$\mathcal{L}(z, \lambda, \nu) = \begin{cases} f(z) + \lambda^\top g(z) + \nu^\top h(z) & \text{if } \nu \geqslant 0, \\ -\infty & \text{otherwise.} \end{cases} \tag{3}$$

B. Augmented Lagrangian methods

Our approach is fundamentally based on augmented Lagrangian-type methods [33], [34]. We consider the classical Powell-Hestenes-Rockafellar (PHR) augmented Lagrangian penalty function, which is defined as the value of the dual proximal point of the Lagrangian:

$$\mathcal{L}_\mu(z; \lambda_e, \nu_e) = \max_{\substack{(\lambda,\nu)\\ \nu\geqslant 0}} \mathcal{L}(z, \lambda, \nu) - \tfrac{\mu}{2}\|(\lambda, \nu) - (\lambda_e, \nu_e)\|_2^2 \tag{4}$$

where $\mu > 0$ is the augmented Lagrangian penalty parameter.

The maximizers (see Appendix A) in (λ, ν) are the following mappings, called *first-order multiplier estimates*

$$\lambda^+(z) \overset{\text{def}}{=} \lambda_e + \frac{1}{\mu} g(z), \tag{5a}$$
$$\nu^+(z) \overset{\text{def}}{=} \left[\nu_e + \frac{1}{\mu} h(z)\right]_+. \tag{5b}$$

Injecting the new updated multiplier values into the Lagrangian function leads to:

$$\begin{aligned}\mathcal{L}_\mu(z; \lambda_e, \nu_e) = f(z) &+ \tfrac{1}{2\mu}\|g(z) + \mu\lambda_e\|^2 - \tfrac{\mu}{2}\|\lambda_e\|^2 \\ &+ \tfrac{1}{2\mu}\left\|[h(z) + \mu\nu_e]_+\right\|^2 - \tfrac{\mu}{2}\left\|\nu_e\right\|^2.\end{aligned} \tag{6}$$

The gradient of the augmented Lagrangian with respect to the primal variable z reads:

$$\nabla_z \mathcal{L}_\mu(z; \lambda_e, \nu_e) = \nabla_z \mathcal{L}(z, \lambda^+(z), \nu^+(z)). \tag{7}$$

The solution z^* to $\nabla_z \mathcal{L}_\mu = 0$ and the multipliers $(\lambda^+(z^*), \nu^+(z^*))$ can be summarized as the following set of equations:

$$\mathcal{T}_\mu(z, \lambda, \nu; \lambda_e, \nu_e) = \begin{bmatrix} \nabla\mathcal{L}(z, \lambda, \nu) \\ g(z) + \mu(\lambda_e - \lambda) \\ [h(z) + \mu\nu_e]_+ - \mu\nu \end{bmatrix} = 0, \tag{8}$$

where $\mathcal{T}_\mu$ is the proximal-KKT operator [44], [45], [51].

The augmented Lagrangian method. The prototypical augmented Lagrangian method or *method of multipliers* [33] consists of minimizing the augmented Lagrangian (6), updating the dual variables, and repeating until convergence. This process is outlined in the abstract Algorithm 1. Decreasing the penalty parameter μ_k can accelerate convergence [29]. Global convergence of the method in the convex case (under feasibility of the dual problem) was proven by Rockafellar [52, Theorem 4] using proximal-point arguments. It was also shown that the penalty parameter μ_k can be kept bounded from below in this setting. Work on generalizing these theoretical results is still an active research topic [53].

Algorithm 1: Prototypical ALM algorithm.

Data: objective f, constraint functions g, h, initial guess z_0

// Outer loop

1 **for** $k = 0$ *to* $k_{\max}$ **do**
2 Find a minimizer z_{k+1} of $\mathcal{L}_{\mu_k}(z; \lambda_k, \nu_k)$ *// inner loop*
// first-order multiplier updates
3 $\lambda_{k+1} = \lambda_k + \frac{1}{\mu_k} g(z_{k+1})$;
4 $\nu_{k+1} = [\nu_k + \frac{1}{\mu_k} h(z_{k+1})]_+$;
5 **if** *converged* **then**
6 **return** $(z_{k+1}, \lambda_{k+1}, \nu_{k+1})$;
7 Choose $\mu_{k+1} > 0$;

Stopping criteria. The algorithm is deemed to have converged when $g(z_k)$ has converged to zero and there exists ν^* such that $\nu_k \to \nu^*$ and the following equivalent conditions are satisfied:

(i) $h(z^*) \leqslant 0 \perp \nu^* \geqslant 0$ (complementarity)
(ii) $\max(\nu^*, -h(z^*)) = 0$
(iii) $\nu^* = \left[\nu^* + \frac{1}{\mu} h(z^*)\right]_+$.

These two sets of conditions imply that the multiplier sequence (λ_k, ν_k) has converged (this stems from using the first-order update). In addition, since the multipliers converged, we have $\nabla\mathcal{L}_\mu(z^*; \lambda^*, \nu^*) = \nabla\mathcal{L}(z^*, \lambda^*, \nu^*) = 0$. All put together, this implies that (z^*, λ^*, ν^*) is a KKT point.

The augmented Lagrangian method is an exact penalty method [29, chap. 17], and can also be interpreted as a sequence of shifted penalty methods as discussed in Appendix A. Outside of specific settings (such as equality-constrained QPs), closed-form minimization of the augmented Lagrangian $\mathcal{L}_\mu$ (Alg. 1, Line 2) is impossible in practice. Thus, the algorithm has to rely on finding an approximate minimizer of:

$$\min_z \mathcal{L}_{\mu_k}(z; \lambda_k, \nu_k), \tag{9}$$

corresponding to the "inner loop" of the prototypical ALM algorithm 1. This raises the question of how accurate this

minimizer needs to be, and several approaches might be considered, such as inexact minimization strategies, as detailed below. Furthermore, for quick convergence, one needs to carefully adapt μ_k according to the progress of the problem feasibility, which should be done in the outer loop, as done, for instance, in the bound-constrained Lagrangian update strategy [43], that we recall below.

C. Solving the inner loop via inexact minimization

A practical algorithm has to perform approximate minimization of $\mathcal{L}_\mu$. The use of inexact minimizers in the AL method has been largely studied and proven in the optimization literature, leading to efficient solutions for solving general NLP problems [34], [43], [54]. In particular, the authors of [43] provide a practical method named *bound-constrained Lagrangian* (BCL) for controlling both the inexactness in the AL minimization and the penalty schedule, with convergence guarantees. The minimization is done within a scheduled tolerance $\omega_k > 0$ on a stopping criterion for the subproblem.

First-order methods for minimizing the augmented Lagrangian function have been investigated [55], as well as second-order methods in the equality-constrained case [35]. Recently, a bigger focus has been put on second-order methods for the inequality-constrained case without using slack variables [49]. For instance, the QP solvers QPALM [44], PROXQP [45], and QPDO [48] rely on semi-smooth Newton methods applied to the augmented Lagrangian function.

Linear system. The semi-smooth Newton method solves the following linear system

$$H_{\mu_k}\delta z = -\nabla_z \mathcal{L}_{\mu_k}(z;\lambda_k,\nu_k), \tag{10}$$

where H_{μ_k} is defined by

$$H_{\mu_k} = H + \frac{1}{2\mu_k}(A^\top A + B_{\mathcal{I}}^\top B_{\mathcal{I}}) \tag{11}$$

with $H = \nabla_z^2 \mathcal{L}(z,\lambda_k^+(z),\nu_k^+(z))$ is the Lagrangian Hessian[1], $A = \partial g/\partial z, B = \partial h/\partial z$ are the constraint Jacobian matrices and $B_{\mathcal{I}}$ is the set of rows of B corresponding to the *active set* of constraints $\mathcal{I} = \mathcal{I}_k(z)$, defined at point $z \in \mathcal{Z}$ as

$$\mathcal{I}_k(z) \overset{\text{def}}{=} \{1 \leqslant j \leqslant m \mid h_j(z) + \mu_k \nu_{k,j} \geqslant 0\}. \tag{12}$$

The linear system (10) is often poorly conditioned. It is suggested [29], [42], [44], to rewrite it as a primal-dual system of normal equations:

$$\begin{bmatrix} H & A^\top & B_{\mathcal{I}}^\top \\ A & -\mu_k I & \\ B_{\mathcal{I}} & & -\mu_k I \end{bmatrix} \begin{bmatrix} \delta z \\ * \\ * \end{bmatrix} = - \begin{bmatrix} \nabla \mathcal{L}_{\mu_k} \\ 0 \\ 0 \end{bmatrix} \tag{13}$$

where the second and third block unknowns are not used – only δz is used. Equations (13) and (10) are equivalent, where the latter can be obtained from the former by Schur complement.

[1]In practice, we can use an approximation of the Lagrangian Hessian which does not use the second constraint derivatives – this corresponds to the *Gauss-Newton* approximation of the augmented Lagrangian Hessian.

Furthermore [45], equation (13) can be rewritten, given any (λ,ν), as an equivalent SQP iteration-like system

$$\begin{bmatrix} H & A^\top & B_{\mathcal{I}}^\top \\ A & -\mu_k I & \\ B_{\mathcal{I}} & & -\mu_k I \end{bmatrix} \begin{bmatrix} \delta z \\ \delta\lambda \\ \delta\nu \end{bmatrix} = - \begin{bmatrix} \nabla f + A^\top\lambda + B_{\mathcal{I}}^\top \nu \\ \mu_k(\lambda^+(z) - \lambda) \\ \mu_k(\nu^+(z) - \nu) \end{bmatrix}. \tag{14}$$

The derivation of a primal-dual step akin to SQP methods highlights the possibility of incorporating the dual variables into the line-search merit function, enabling the search of optimal solutions for primal and dual variables simultaneously.

Given the search direction δz and $\alpha \in (0,1]$, a candidate point z is given by

$$z \leftarrow z + \alpha \delta z. \tag{15}$$

This is the classical scheme, where the appropriate stopping criterion would be the AL gradient $\|\nabla \mathcal{L}_{\mu_k}(z;\lambda_k,\nu_k)\|_\infty \leqslant \omega_k$.

Primal-dual search. Yet, the equivalent system (14) suggests also including search directions in (λ,ν), as done by [48], [56]. In each subproblem, the dual variables are to be initialized as

$$\lambda \leftarrow \lambda_k,\ \nu \leftarrow \nu_k.$$

Then, these variables are updated in SQP fashion using the dual step computed from (14)

$$\lambda \leftarrow \lambda + \alpha\delta\lambda,\ \nu \leftarrow \nu + \alpha\delta\nu, \tag{16}$$

where α is the same step size over the dual variables as the one over primal variables, which we choose for simplicity.

Linesearch and the merit function. The globalization strategy we adopt is based on a linesearch procedure [29, chap. 3] using a merit function, a function for which the computed search direction is a descent direction. Several merit functions can be envisaged. Depending on the chosen scheme, one may use the augmented Lagrangian $\mathcal{L}_{\mu_k}$ for linesearch over the primal variable z only. An alternative solution is the primal-dual augmented Lagrangian introduced by Gill and Robinson [56] and recently extended to the inequality-constrained case by De Marchi [48] (we provide a derivation in Appendix B):

$$\begin{aligned} \mathcal{M}_\mu(z,\lambda,\nu;\lambda_e,\nu_e) = \\ f(z) + \frac{1}{\mu}\|g(z) + \mu(\lambda - \lambda_e/2)\|^2 + \frac{\mu}{4}\|\lambda\|^2 \\ + \frac{1}{\mu}\|[h(z) + \mu(\nu - \nu_e/2)]_+\|^2 + \frac{\mu}{4}\|\nu\|^2. \end{aligned} \tag{17}$$

In this case, the linesearch is performed over the primal and dual variables (z,λ,ν).

The linesearch procedure checks for the Armijo condition: at a new proposal primal-dual point (z^+,λ^+,ν^+) given by a trial step size $\alpha \in (0,1]$, we check if

$$\begin{aligned} \mathcal{M}_\mu(z^+,\lambda^+,\nu^+;\lambda_e,\nu_e) \leqslant \\ \mathcal{M}_\mu(z,\lambda,\nu;\lambda_e,\nu_e) + c_1\alpha\nabla\mathcal{M}_\mu^\top(\delta z,\delta\lambda,\delta\nu) \end{aligned} \tag{18}$$

where $c_1 \in (0,1/2)$ is a given hyperparameter; typical values in the literature [29], [43] include $\{10^{-2},10^{-3},10^{-4}\}$. In practice, our implementation of the linesearch procedure

uses up to third-order polynomial interpolation as described in [29, ch. 3]. To solve the interpolation problem, we use a QR decomposition of the linear system instead of the explicit formula in [29, ch. 3], which we found less numerically stable and prone to numerical underflow.

Inner-loop stopping criterion. When using the primal-dual search, an appropriate metric on the primal-dual convergence is given by the following infinity norm:

$$r_k(z,\lambda,\nu) \stackrel{\text{def}}{=} \left\| \begin{bmatrix} \nabla_z \mathcal{L}(z,\lambda,\nu) \\ \mu_k(\lambda_k^+(z)-\lambda) \\ \mu_k(\nu_k^+(z)-\nu) \end{bmatrix} \right\|_\infty . \tag{19}$$

As stopping criteria, we consider (z,λ,ν) to be sufficiently optimal for the inner loop when $r_k(z,\lambda,\nu) \leqslant \omega_k$.

D. Outer-loop parameter updates and stopping criterion

Once the subproblem (9) is approximately solved, either the multiplier update (5) is applied, or the penalty parameter μ_k is decreased to $\mu_{k+1} = \mu_\text{f}\mu_k$, depending on if the new constraint violation $\|(g, h - [h + \mu_k\nu_k]_-)\|_\infty$ is within another tolerance $\eta_k > 0$. When one or the other algorithm branch is taken, the subproblem tolerances (ω_k, η_k) are updated following a geometric rule.

Akin to [45], our approach uses the penalty-scheduling BCL method of [35], [43], which, to the best of our knowledge, had not been used with alongside a primal-dual semi-smooth Newton for the augmented Lagrangian in the inequality constrained case. In contrast, the reference paper and solver for BCL [35], [43] consider non-negativity ($z \geqslant 0$) or box constraints ($l \leqslant z \leqslant u$) on the optimization variable z.

We use the constraint violation measure $\|g(z^{k+1})\|_\infty$ for equality constraints to assess overall convergence. For inequalities, this measure is $\|h(z^{k+1}) - [h(z^{k+1}) + \mu_k\nu_k]_-\|_\infty$, which fully accounts for the complementarity conditions (2c).

E. Summary of the approach and software implementation

The overall approach, including how the subproblem tolerances and penalty are updated, is summarized in Algorithm 2. We also provide an efficient C++ implementation of this primal-dual proximal NLP solver, called PROXNLP[2] and detailed in Section VI, which notably operates on generic manifolds [57], unlike existing off-the-shelf NLP solvers such as IPOPT [6] or SNOPT [7].

Algorithm 2: AL method with inexact subproblem minimization.

Input: Initial guesses (z_0, λ_0, ν_0), $\mu_\text{f} \in (0,1)$, parameters $(\alpha_\omega, \alpha_\eta, \beta_\omega, \beta_\eta)$, target tolerance ϵ_tol

// Outer loop

for $k = 0$ *to* $k_\max$ **do**

  // initialize inner loop

  $(\hat{z}, \hat{\lambda}, \hat{\nu}) \leftarrow (z_k, \lambda_k, \nu_k)$;

  // Inner loop

  repeat

    $(\delta z, \delta\lambda, \delta\nu) \leftarrow$ Solve linear system (14);

    Find step-size α by linesearch on (17);

    $(\hat{z}, \hat{\lambda}, \hat{\nu}) \leftarrow (\hat{z}, \hat{\lambda}, \hat{\nu}) + \alpha(\delta z, \delta\lambda, \delta\nu)$ *// eq.* (15)–(16)

  until $r_k(\hat{z}, \hat{\lambda}, \hat{\nu}) \leqslant \omega_k$ *// until eq.* (19);

  Set $z_{k+1} = \hat{z}$;

  // Constraint violation measure; see subsection II-D

  Set $c_{k+1} = \|(g(z_{k+1}), \max(h(z_{k+1}), -\mu_k\nu_k))\|_\infty$;

  if $c_{k+1} \leqslant \eta_k$ **then**

    // accept multipliers

    $\lambda_{k+1} = \hat{\lambda}$;

    $\nu_{k+1} = \hat{\nu}$;

    $\mu_{k+1} = \mu_k$;

    if $\max(r_k, c_{k+1}) \leqslant \epsilon_{tol}$ **then**

      return $(z_{k+1}, \lambda_{k+1}, \nu_{k+1})$;

    // update tolerances following BCL (success)

    $\omega_{k+1} = \omega_k \mu_{k+1}^{\alpha_\omega}$;

    $\eta_{k+1} = \eta_k \mu_{k+1}^{\alpha_\eta}$;

  else

    $\lambda_{k+1} = \lambda_k$;

    $\nu_{k+1} = \nu_k$;

    // decrease penalty

    $\mu_{k+1} = \mu_\text{f}\mu_k$;

    // update tolerances following BCL (failure)

    $\omega_{k+1} = \omega_k \mu_{k+1}^{\beta_\omega}$;

    $\eta_{k+1} = \eta_k \mu_{k+1}^{\beta_\eta}$;

  $\omega_{k+1} = \max(\omega_{k+1}, 10^{-2}\epsilon_\text{tol})$

III. BACKGROUND ON TRAJECTORY OPTIMIZATION AND DIFFERENTIAL DYNAMIC PROGRAMMING

We now move to extend the algorithm presented in Section II to resolve constrained optimal control problems. Before presenting this new approach in Section IV, we first review the fundamentals of trajectory optimization and the classic DDP algorithm it is based on.

[2]https://github.com/Simple-Robotics/proxnlp

A. Problem statement

Continuous time formulation. The goal of nonlinear trajectory optimization is to approximately solve continuous-time optimal control problems of the form:

$$\min_{\underline{x},\underline{u}} \; J(\underline{x},\underline{u}) = \int_0^T \ell_t(x(t),u(t))\,\mathrm{d}t + \ell_T(x(T)) \tag{20a}$$

$$\text{s.t. } \dot{x}(t) = f_t(x(t),u(t)), \tag{20b}$$

$$x(0) = \hat{x}_0, \tag{20c}$$

$$h_t(x(t),u(t)) \leqslant 0, \tag{20d}$$

$$h_T(x(T)) \leqslant 0, \tag{20e}$$

where $\hat{x}_0 \in \mathcal{X}$ is a given initial state, g_t, g_T, h_t, and h_T are constraint-defining functions, f is the time-dependant system dynamics, and ℓ_t and ℓ_T are the running and terminal cost functions, respectively. The objective J is called the *cost functional*. $\underline{x}$ and $\underline{u}$ are the state and control trajectories respectively. Problem (20) is an infinite-dimensional problem

that, in most cases, cannot be directly solved.

Discrete time formulation. The approximate resolution of the problem can be tackled using *direct methods* [5]. These methods rely on a direct *transcription* of problem (20), which is a discretization of the original problem as a finite-dimensional, nonlinear program (NLP):

$$\min_{\boldsymbol{x},\boldsymbol{u}} \; J(\boldsymbol{x},\boldsymbol{u}) \stackrel{\text{def}}{=} \sum_{t=0}^{N-1} \ell_t(x_t,u_t) + \ell_T(x_N) \tag{21a}$$

$$\text{s.t. } f_t(x_t,u_t,x_{t+1}) = 0, \tag{21b}$$

$$x_0 = \hat{x}, \tag{21c}$$

$$h_t(x_t,u_t) \leqslant 0, \tag{21d}$$

$$h_N(x_N) \leqslant 0, \tag{21e}$$

where t now denotes the discrete-time variable, a notation we will keep in the sequel. We also abuse notation by keeping the same notations f, h, ℓ, although in the discrete setting, they could be the result of numerical integration of (20) (e.g. using Euler or RK integration [58]). This class of NLP is also called a discrete-time optimal control (DTOC) problem due to its inherent structure [5]. Here, we use the most generic discretization of the dynamics as an implicit integration scheme [46], which also covers explicit formulations that are more common in the DDP literature. This formulation encompasses path equality constraints as inequality constraints, although these are implemented separately in practice.

We introduce the following Lagrange multipliers: $\boldsymbol{\lambda} = (\lambda_t)_{t=0,\dots,N}$ are the co-states (i.e. the multipliers for dynamics, with λ_0 corresponding to the initial condition $x_0 = \hat{x}$), $\boldsymbol{\nu} = (\nu_t)_{t=0,\dots,N} \geqslant 0$ for the path constraints (with ν_N corresponding to the terminal constraint (20c)). The Lagrangian of the problem reads

$$\begin{aligned}\mathcal{L}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\lambda},\boldsymbol{\nu}) &= J(\boldsymbol{x},\boldsymbol{u}) + \lambda_0^\top(x_0 - \hat{x}) \\ &+ \sum_{t=0}^{N-1} \lambda_{t+1}^\top f_t(x_t,u_t,x_{t+1}) + \nu_t^\top h_t(x_t,u_t) \\ &+ \nu_N^\top h_N(x_N).\end{aligned} \tag{22}$$

B. KKT conditions and Hamiltonian

The necessary optimality conditions given by the so-called Lagrangian stationarity conditions [29] read:

$$\begin{bmatrix}\nabla_{\boldsymbol{x}}\mathcal{L} \\ \nabla_{\boldsymbol{u}}\mathcal{L}\end{bmatrix} = 0. \tag{23}$$

Expanding these conditions for each of the variables x_t, u_t leads to:

$$\ell_u + f_{t,u}^\top \lambda_{t+1} + h_{t,u}^\top \nu_t = 0 \tag{24a}$$

$$f_{t,y}^\top \lambda_t + \ell_x + f_{t,x}^\top \lambda_{t+1} + h_{t,x}^\top \nu_t = 0, \tag{24b}$$

and

$$\ell_x + f_{t,x}^\top \lambda_1 + h_{0,t}^\top \nu_0 = 0 \tag{24c}$$

$$f_{t,y}^\top \lambda_N + \ell_{N,x} + h_{x,N}^\top \nu_N = 0, \tag{24d}$$

where we now use the shorthands $f_{t,x} = \partial f_t/\partial x$, $f_{t,u} = \partial f_t/\partial u$ and so on. It is worth noticing that, in the case of explicit dynamics where $f_{t,y} = -I$ (where y denotes x_{t+1}), we recover the equations of a discrete-time Pontryagin's principle of optimality. The feasibility and complementarity conditions read:

$$f_t(x_t,u_t,x_{t+1}) = 0 \text{ and } 0 \leqslant \nu_t \perp -h_t \geqslant 0. \tag{25}$$

C. Differential dynamic programming

Notation. In this section, for a vector $\boldsymbol{z} = (z_0,\dots,z_n)$, $\boldsymbol{z}_{\leqslant k}$ denotes the prefix $(z_0,\dots,z_k)$ while $\boldsymbol{z}_{\geqslant k}$ denotes the suffix $(z_k,\dots,z_n)$.

Differential dynamic programming (DDP) [12], and its variants such as iLQR [14], [15] are a class of second-order gradient methods which rely on Bellman's principle of optimality to compute descent directions and efficiently exploit the sparsity induced by time.

In its classical formulation recalled below, DDP is only able to handle unconstrained instances of (21) of the form:

$$\begin{aligned}V_0(x_0) \stackrel{\text{def}}{=} &\min_{\boldsymbol{x},\boldsymbol{u}} J(\boldsymbol{x},\boldsymbol{u}), \\ &\text{s.t. } x_{t+1} = f_t(x_t,u_t),\ t = 0,\dots,N-1,\end{aligned} \tag{26}$$

where $\boldsymbol{x} \stackrel{\text{def}}{=} (x_0,\dots,x_N)$ and $\boldsymbol{u} \stackrel{\text{def}}{=} (u_0,\dots,u_{N-1})$. The value function V_t starting at time t is defined as

$$V_t(x) \stackrel{\text{def}}{=} \min \sum_{i=t}^{N-1} \ell_i(x_i,u_i) + \ell_T(x_N), \tag{27}$$

in which the relationship $x_{t+1} = f_t(x_t,u_t)$ still holds and $x_t = x$. Bellman's principle of optimality states that each subproblem is linked with the solution of the next subproblem through the Bellman recursion:

$$V_t(x) = \min_u \underbrace{\ell_t(x,u) + V_{t+1}(f_t(x_t,u_t))}_{\stackrel{\text{def}}{=} Q_t(x,u)}. \tag{28}$$

Once this backward recursion is solved from $t = N-1$ down to $t = 0$, the solution $(\boldsymbol{x},\boldsymbol{u})$ can be reconstructed by a forward pass, as the initial state x_0 is known.

In general, the recursion cannot be solved in closed form as a function of x, except for specific cases such as the linear-quadratic (LQ) problems. However, a local (second-order) expansion of its minimum around x can be obtained if expansions of ℓ and V_{t+1} are available.

Backward pass. The Bellman equation can be rewritten as follows:

$$V_t(x) = \min_u Q_t(x,u). \tag{29}$$

The idea of DDP is to solve for a quadratic model of Q_t, leveraging the derivatives

$$Q_{a,t} = \ell_a + f_{t,a}^\top V_{x,t+1}, \tag{30a}$$

$$Q_{ab,t} = \ell_{ab} + V_{x,t+1} \cdot f_{t,ab} + f_a^\top V_{xx,t+1} f_{t,b}, \tag{30b}$$

where the symbols $a, b \in \{x,u\}$. In this perspective, the iterative LQR by Tassa [15] neglects the terms of the second-order dynamic $\nabla V \cdot f_{ab}$ – which can be interpreted as a *Gauss-Newton approximation* of the action-value derivatives.

Minimizing the quadratic model involved in (29) with respect to both x and u leads to the relation between the variations of δu_t and δx_t:

$$Q_{uu,t}\,\delta u_t^\star = -Q_{u,t} - Q_{ux,t}\,\delta x_t. \tag{31}$$

Inverting this equation relies on the assumption that the Hessian Q_{uu} is positive-definite. In practice, some regularization is needed to enforce $Q_{uu} \succ 0$ as in [15]. The resulting relationship is

$$\delta u_t^\star = k_t + K_t \delta x_t, \tag{32}$$

where

$$k_t = -Q_{uu,t}^{-1} Q_{u,t}, \quad K_t = -Q_{uu,t}^{-1} Q_{ux,t}. \tag{33}$$

Forward pass. A nonlinear search is adopted in the classical DDP algorithm [12], [15]. A candidate solution $(\boldsymbol{x}^\alpha, \boldsymbol{u}^\alpha)$ is defined by a rollout of the nonlinear dynamics with a linesearch parameter $\alpha \in (0, 1]$ on the control feedforward term:

$$u_t^\alpha = \alpha k_t + K_t(x_t^\alpha - x_t) \tag{34a}$$
$$x_{t+1}^\alpha = f_t(x_t^\alpha, u_t^\alpha),\ t = 0, \dots N-1. \tag{34b}$$

The line-search parameter $\alpha \in (0, 1]$ is chosen to satisfy an Armijo sufficient decrease condition [29, ch. 3] on the cost function, or some second-order version [15], [18].

D. Discussion and limitations of vanilla DDP

The strength of DDP is its efficiency, which is due to clever exploitation of the time sparsity structure of (21). It is also relatively easy to implement. These two elements make this algorithm a popular choice for implementing MPC on complex robotic systems [4], [16], [59]. It has proven strong convergence properties [8] and is available in modern software libraries such as CROCODDYL [18] or OCS2 [60]. We recall that the weakness of the nominal algorithm is its inability to enforce strict constraints and the lack of a proper globalization strategy. Attempts have been made to use modified DDPs as a backbone for constrained solvers, using augmented Lagrangian methods, for instance [36], [38], [42]. Our proposal, in the next section, is to bring the ideas of the proximal framework we introduced in Section II together with DDP-like recursions to produce an efficient and numerically stable algorithm for constrained trajectory optimization that can run on real robots in real-time.

IV. Proximal Differential Dynamic Programming

In this section, we detail the central contribution of this article, leveraging proximal and augmented Lagrangian techniques to incorporate the strengths of DDP whilst extending its range to more general constrained trajectory optimization problems.

A. Proximal reformulation of DTOCP

Similarly as in Section II, we propose to solve the DTOC (21) with an augmented Lagrangian scheme, where we iteratively minimize the AL function:

$$\begin{aligned}\mathcal{L}_{\mu_k}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}_k, \boldsymbol{\nu}_k) = J(\boldsymbol{x}, \boldsymbol{u}) + \tfrac{1}{2\mu_k}\|x_0 - \hat{x} + \mu_k\lambda_0\|^2 \\ + \sum_{t=0}^{N-1} \Big(\tfrac{1}{2\mu_k}\|f_t(x_t, u_t, x_{t+1}) + \mu_k\lambda_{k,t+1}\|^2 \\ + \tfrac{1}{2\mu_k}\|[h_t(x_t, u_t) + \mu_k\nu_{k,t}]_+\|^2\Big) \\ + \tfrac{1}{2\mu_k}\|[h_N(x_N) + \mu_k\nu_{k,N}]_+\|^2.\end{aligned} \tag{35}$$

We now detail how the equations of Section II can be implemented efficiently using dynamic programming, following a backward-pass/forward-pass strategy just like DDP.

The first-order conditions can be restated by introducing the following Q-function:

$$Q_t(x, u, y, \lambda, \nu) = \ell(x, u) + \lambda^\top f_t(x, u, y) + \nu^\top h_t(x, u) + V_{t+1}(y), \tag{36}$$

where the V_t are the cost-to-go functions for the augmented Lagrangian. Dropping the subscript t, derivatives of Q_t are given by:

$$Q_x = \ell_x + f_x^\top\lambda + h_x^\top\nu, \tag{37a}$$
$$Q_u = \ell_u + f_u^\top\lambda + h_u^\top\lambda, \tag{37b}$$
$$Q_y = V_x' + f_y^\top\lambda. \tag{37c}$$

For completeness, we also write the second-order derivatives:

$$Q_{xx} = \ell_{xx} + \lambda\cdot f_{xx} + \nu\cdot h_{xx}, \tag{37d}$$
$$Q_{xu} = Q_{ux}^\top = \ell_{xu} + \lambda\cdot f_{xu} + \nu\cdot h_{xu}, \tag{37e}$$
$$Q_{uu} = \ell_{uu} + \lambda\cdot f_{uu} + \nu\cdot h_{uu}, \tag{37f}$$
$$Q_{xy} = Q_{yx}^\top = \lambda\cdot f_{xy} + \nu\cdot h_{xy}, \tag{37g}$$
$$Q_{uy} = Q_{yu}^\top = \lambda\cdot f_{uy} + \nu\cdot h_{uy}, \tag{37h}$$
$$Q_{yy} = V_{xx}' + \lambda\cdot f_{yy}. \tag{37i}$$

Using these notations and following the approach of Section II, the stationarity of the augmented Lagrangian (6) can be rewritten as nonsmooth, proximal KKT equations for $t = 0, \dots, N-1$:

$$\mathcal{T}_{t,k}(u, y, \lambda, \nu; x) = \begin{bmatrix} Q_{t,u} \\ Q_{t,y} \\ f_t + \mu_k(\lambda_k - \lambda) \\ [h_t + \mu_k\nu_k]_+ - \mu_k\nu \end{bmatrix} = 0 \tag{38}$$

where $f_t = f_t(x_t, u_t, x_{t+1})$ and $h_t = h_t(x_t, u_t)$ are the constraint values. This operator $\mathcal{T}_{t,k}$ is the dynamic programming equivalent to (8).

In the sequel, we reintroduce the notation for the first-order multiplier estimates associated with the dynamics and the path constraints, similar to (5):

$$\begin{aligned}\lambda_{k,t+1}^+ &= \lambda_{k,t+1} + \tfrac{1}{\mu_k} f_t(x_t, u_t, x_{t+1}), \\ \nu_{k,t}^+ &= \big[\nu_{k,t} + \tfrac{1}{\mu_k} h_t(x_t, u_t)\big]_+.\end{aligned} \tag{39}$$

B. Backward pass

In the sequel, we denote by $f_t = f_t(x_t, u_t, x_{t+1})$ the value of the dynamical constraint for $t \leqslant 1$, $h_t = h_t(x_t, u_t)$ the value of the inequality constraints, $\tilde{f}_t = f_t + \mu_k(\lambda_{k,t} - \lambda_t)$ the shifted value of the dynamics constraint, and $\tilde{h}_t = [h_t + \mu_k\nu_{k,t}]_+ - \mu_k\nu_t$ the shifted inequality constraint.

In the spirit of DDP, we now show that we can compute a search direction efficiently through dynamic programming by solving a set of Riccati-type equations.

Terminal stage. In this proximal formulation, the terminal value function is

$$\begin{aligned} V_N(x) &= \max_{\nu \geqslant 0} \ \ell_T(x) + \nu^\top h_N(x) - \frac{\mu_k}{2}\|\nu - \nu_{N,k}\|^2 \\ &= \ell_T(x) + \frac{1}{2\mu_k}\|[h_N(x) + \mu_k\nu_{k,N}]_+\|^2 - \frac{\mu_k}{2}\|\nu_{k,N}\|^2. \end{aligned} \tag{40}$$

It is a continuously differentiable function in x, and has a generalized Hessian. The quadratic model of this value function involves its gradient,

$$\begin{aligned} \nabla_x V_N &= \nabla\ell_T + \frac{1}{\mu_k} h_{x,N}^\top [h_N + \mu_k\nu_{k,N}]_+ \\ &= \nabla\ell_T + h_{x,N}^\top \nu_{k,N}^+, \end{aligned}$$

and choosing an element of its generalized Hessian [61],

$$\hat{V}_{xx,N} = \nabla^2\ell_T + \frac{1}{\mu_k} h_{x,N}^\top P h_{x,N} + \frac{1}{\mu_k}\nu_{k,N}^+ \cdot h_{xx,N}. \tag{41}$$

We choose P as a diagonal projection matrix onto the active set $\mathcal{I}$. The second term can be rewritten $h_x^\top P h_x = (h_x)_{\mathcal{I}}^\top (h_x)_{\mathcal{I}}$, and the third term is often neglected in Gauss-Newton approximations.

Computing feedforward and feedback gains. For convenience, we will drop the time index t. At each step t, we look for a primal-dual search direction $\delta w = (\delta u, \delta y, \delta\lambda, \delta\nu)$ by applying a semi-smooth Newton method [62] to $\mathcal{T}_{t,k}(w)$. This leads to solving a linear system of equations

$$\mathcal{K}_\mu \begin{bmatrix} \delta u \\ \delta y \\ \delta\lambda \\ \delta\nu \end{bmatrix} = - \begin{bmatrix} Q_u + Q_{ux}\delta x \\ Q_y + Q_{yx}\delta x \\ \mu_k(\lambda_k^+ - \lambda) + f_x\delta x \\ \mu_k(\nu_k^+ - \nu_t) + (h_x)_{\mathcal{I}}\delta x \end{bmatrix}. \tag{42}$$

where $\mathcal{K}_\mu$ is the following matrix, an element of the generalized Jacobian $\partial\mathcal{T}_{t,k}(w)$ of (38)

$$\mathcal{K}_\mu = \begin{bmatrix} Q_{uu} & Q_{uy} & f_u^\top & h_u^\top \\ Q_{yu} & Q_{yy} & f_y^\top & \\ f_u & f_y & -\mu I & \\ (h_u)_{\mathcal{I}} & & & -\mu I \end{bmatrix}. \tag{43}$$

As δx is unknown, we need to solve the sensitivities' system

$$\mathcal{K}_\mu \begin{bmatrix} k & K \\ c & C \\ \xi & \Xi \\ \zeta & Z \end{bmatrix} = - \begin{bmatrix} Q_u & Q_{ux} \\ Q_y & Q_{yx} \\ \mu_k(\lambda_k^+ - \lambda) & f_x \\ \mu_k(\nu_k^+ - \nu_t) & (h_x)_{\mathcal{I}} \end{bmatrix}. \tag{44}$$

Obtaining a symmetric linear system. It is important to notice that $\mathcal{K}_\mu$ is *not* a symmetric matrix, disqualifying it from using symmetric linear solvers, e.g., Cholesky decomposition. However, the system can be transformed into a symmetric one by exploiting the fact that the last line implies that we can solve for the inactive multipliers satisfy $(\delta\nu)_{\overline{\mathcal{I}}} = -\nu_{\overline{\mathcal{I}}}$ and that $h_u = (h_u)_{\mathcal{I}} + (h_u)_{\overline{\mathcal{I}}}$. Now we define the *symmetric* matrix as

$$\underline{\mathcal{K}}_\mu \stackrel{\text{def}}{=} \begin{bmatrix} Q_{uu} & Q_{uy} & f_u^\top & (h_u)_{\mathcal{I}}^\top \\ Q_{yu} & Q_{yy} & f_y^\top & \\ f_u & f_y & -\mu I & \\ (h_u)_{\mathcal{I}} & & & -\mu I \end{bmatrix}. \tag{45}$$

The system (42) from above is now equivalent to

$$\underline{\mathcal{K}}_\mu \begin{bmatrix} k & K \\ c & C \\ \xi & \Xi \\ \zeta & Z \end{bmatrix} = - \begin{bmatrix} Q_u + (h_u)_{\overline{\mathcal{I}}}^\top \nu & Q_{ux} \\ Q_y & Q_{yx} \\ \mu_k(\lambda_k^+ - \lambda) & f_x \\ \mu_k(\nu_k^+ - \nu) & (h_x)_{\mathcal{I}} \end{bmatrix} \tag{46}$$

As $\nu = \nu_{\mathcal{I}} + \nu_{\overline{\mathcal{I}}}$, the top-left block on the right-hand side reads

$$\ell_u + f_u^\top\lambda + h_u^\top\nu_{\mathcal{I}} = \ell_u + f_u^\top\lambda + (h_u)_{\mathcal{I}}^\top\nu.$$

The right-hand side of (46) should *not* be used as a stopping criterion for the subproblem. It is merely a proxy to get a symmetric linear system. Inverting $\underline{\mathcal{K}}_\mu$ is similar to inverting a regularization of Q_{uu} in classical DDP. The control feedback gains in (33) also have an equivalent in (46). Here, the proximal regularization in the dual blocks of (45) guarantees it is always well-conditioning, enhancing the method's convergence.

Propagating the value function model. The gradient of the value function is given by

$$V_x = Q_x + Q_u^\top K + Q_y^\top C + f^\top\Xi + h^\top Z. \tag{47a}$$

since $Q_\lambda = f$ and $Q_\nu = h$. Its Hessian is

$$V_{xx} = Q_{xx} + Q_{xu}K + Q_{xy}C + f_x^\top\Xi + (h_x)_{\mathcal{I}}^\top Z. \tag{47b}$$

These equations can be derived by taking the optimality condition

$$\nabla_x V_t = \nabla_x Q_t$$

and writing out its Taylor expansion, introducing the shorthand $\delta w = (\delta u, \delta y, \delta\lambda, \delta\nu)$,

$$V_x + V_{xx}\delta x = Q_x + Q_{xx}\delta x + Q_{xw}\delta w, \tag{48}$$

and finally, replacing the expression of δw.

Initial stage. We suggest here to handle the initial condition $x_0 = \hat{x}$ using the same proximal scheme by considering the saddle point

$$\min\max\, V_0(x_0) + \lambda_0^\top(x_0 - \hat{x}) - \frac{\mu}{2}\|\lambda_0 - \lambda_{k,0}\|^2 \tag{49}$$

The corresponding SQP step solves the system of equations

$$\begin{bmatrix} V_{xx0} & I \\ I & -\mu I \end{bmatrix} \begin{bmatrix} \delta x_0 \\ \delta\lambda_0 \end{bmatrix} = - \begin{bmatrix} V_{x0} + \lambda_0 \\ \mu_k(\lambda_{k,0} - \lambda_0) + x_0 - \hat{x} \end{bmatrix}.$$

A benefit of our method is that it generalizes to other initial stage constraints, as discussed in Appendix C. We can also discard x_0 (and λ_0) from the decision variables of our problem and force $x_0 = \hat{x}$ in the algorithm's execution – this saves some computation but might be counter-productive in some scenarios such as warm-starting MPC schemes.

C. Forward pass

We discuss two ways of performing the forward pass: (i) a linear rollout computing the sequential quadratic programming (SQP) step, and (ii) a nonlinear rollout like DDP methods [12], [15].

Linear rollout. A linear rollout reconstructs the SQP step $(\delta\boldsymbol{x}, \delta\boldsymbol{u}, \delta\boldsymbol{\lambda})$ by using the feedback and feedforward gains computed in the backward pass: compute $(\delta x_0, \delta\lambda_0)$ by solving (65), and for every $t = 0, \dots, N-1$,

$$\delta u_t = k_t + K_t \delta x_t \tag{50a}$$
$$\delta x_{t+1} = c_t + C_t \delta x_t \tag{50b}$$
$$\delta \lambda_{t+1} = \xi_t + \Xi_t \delta x_t \tag{50c}$$

This linear rollout was used in the first iLQR paper [14], but it is not used in DDP papers for robotics which follow [12], [15].

Nonlinear rollout. Liao and Shoemaker [63] argue that one of the strengths of DDP methods regarding their convergence regime is the use of the full nonlinear dynamics in the forward rollout instead of the linearized dynamics, which recover the SQP or Newton directions. This is also used in the estimation literature, e.g., in extended Kalman filters (EKFs), and was one of the enhancements to iLQR proposed by Tassa [15], which brought iLQR closer to the original DDP algorithm. A nonlinear rollout can be defined as follows, in the case of explicit system dynamics $y = f^{\text{ex}}(x, u)$: given a step size $\alpha \in (0, 1]$, the new trajectory $(\mathbf{x}^\alpha, \mathbf{u}^\alpha, \boldsymbol{\lambda}^\alpha)$ is given by:

$$x_0^\alpha \overset{\text{def}}{=} x_0 + \alpha \delta x_0, \tag{51a}$$
$$\lambda_0^\alpha \overset{\text{def}}{=} \lambda_0 + \alpha \delta \lambda_0, \tag{51b}$$
$$u_t^\alpha \overset{\text{def}}{=} u_t + \alpha k_t + K_t(x_t^\alpha - x_t), \tag{51c}$$
$$\lambda_{t+1}^\alpha \overset{\text{def}}{=} \lambda_{t+1} + \alpha \xi_{t+1} + \Xi_{t+1}(x_t^\alpha - x_t), \tag{51d}$$
$$x_{t+1}^\alpha \overset{\text{def}}{=} f^{\text{ex}}(x_t^\alpha, u_t^\alpha) + \mu_k(\lambda_{t+1}^\alpha - \lambda_{t+1}). \tag{51e}$$

Nonlinear rollout with implicit dynamics. In the implicit-dynamics case, we have to replace (51e) by solving the equation

$$f_t(x_t^\alpha, u_t^\alpha, y) + \mu_k(\lambda_{t+1}^\alpha - \lambda_{t+1}) = 0$$

in the unknown y. In practice, this can be done approximately using a root-finding algorithm such as the Newton-Raphson method, similar to Chatzinikolaidis et al. [64]. Having to do this in the case of implicit dynamics is a clear limitation but is necessary if the user wants to perform nonlinear rollouts. From an algorithmic perspective, the only way we have seen of circumventing this is to use a linear rollout instead.

D. Convergence criterion

The convergence criterion for the solver is given by checking for the KKT conditions through both the dual residual

$$r_{\text{d}} = \left\| (\nabla_{\boldsymbol{x}} \mathcal{L}, \nabla_{\boldsymbol{u}} \mathcal{L}) \right\|_\infty, \tag{52}$$

corresponding to (24) and the primal residual (covering the complementarity conditions):

$$r_{\text{p}} = \left\| (f_t, [h_t + \mu_k \nu_{k,t}]_- - h_t)_t \right\|_\infty. \tag{53}$$

Furthermore, the globalization strategy we adopt is through linesearch, using either the augmented Lagrangian (35) or the primal-dual function (17).

E. The PROXDDP algorithm

The implementation of the PROXDDP method presented in this section corresponds to Algorithm 2, where the equations associated with the Newton step in the NLP are replaced by the dynamic programming-based approach outlined previously. A comparison of the decision variables and equations in both methods is given by table I.

TABLE I
EQUIVALENCE WITH THE NLP METHOD OF SECTION II AND THE PROXDDP METHOD.

	PROXNLP	**PROXDDPU**
Decision vars.	z (primal) (λ, ν) (dual)	$(\boldsymbol{x}, \boldsymbol{u})$ (primal) $(\boldsymbol{\lambda}, \boldsymbol{\nu})$ (dual)
KKT operator	$\mathcal{T}_\mu(z, \lambda, \nu; \lambda_e, \nu_e)$ (8)	$(\mathcal{T}_{t,k})_t$ + dyn. prog. (38)
Newton step	(13)	(46)
Update	(15) and (16)	(50) or (51)
Inner criterion	(19)	r_{p} (53) and r_{d} (52)

F. Practical details

Choice of rollouts. In our experiments, we used the nonlinear rollout formulation of the forward pass when explicit dynamics were considered, adhering to the philosophy of classical DDP. Parallel work in the controls community [65]–[68] has taken a look at using a specific nonlinear dynamics-based projection of the Newton or SQP direction in trajectory-space (initially for continuous-time trajectories [65]). This literature suggests using the Riccati solution of the linearized system to define the projection operator. A link [66] has been suggested with the feasible SQP method introduced by [69], which was showcased in the latter paper in the context of MPC. Further research should be done on interpreting the nonlinear rollout in this setting and its real-world performance as applied to complex, nonlinear systems in robotics, which we leave to future work.

Multiple shooting. Unlike what the definition of problem (21) suggests, where the unknown states $\boldsymbol{x} = (x_t)_t$ are an explicit decision variable, DDP methods classically reduce it in practice to a single shooting formulation where the x_t are dependent on $\boldsymbol{u}$ at every step of the algorithm. Single-shooting formulations have several drawbacks. One of them is their inability to properly use infeasible initial guess trajectories, which is troublesome for settings such as locomotion problems. Furthermore, they require solving for the dynamics in the forward pass, which might be costly or numerically harmful in the case of implicit integration schemes or DAEs. In contrast, multiple-shooting approaches help create a more tractable optimization landscape by adding more variables

and allowing an amount of infeasibility at some problem nodes [5]. A combination of linear, SQP-type, and nonlinear, DDP-type rollouts also leads to a variant of multiple shooting, as explored in [11]. Standard DDP can also be augmented into a multiple-shooting algorithm by introducing slack variables [18], [42] for the dynamical constraints. In our method, the slack variable is linked to the gap in convergence in the dual (co-state) variables, and the link with the former approaches is mathematically explored in Appendix D.

Further extensions. In Appendix C, we provide extensions to the algorithm to account for other initial stage constraints than $x_0 = \hat{x}$. We also show how to modify the algorithm to allow for individual scaling of each constraint, which we found useful to tune the convergence behavior of the solver in difficult scenarios.

V. Discussion on other methods

The treatment of constraints in the literature is often dependent on the type of constraints, as inequality constraints are generally seen as more complex due to the nature of their optimality conditions. For equality constraints, some recent works [20], [21] adopt a nullspace-based method for the resolution of equality-constrained LQR subproblems – the equivalent to the equality-constrained QP subproblem in sequential quadratic programming (SQP) methods [29, chap. 18].

Some recent other works advocate for using augmented Lagrangian (AL) approaches for equality constraints combined with DDP-like recursion, which has led to the development of several methods in recent years [38], [41], [42], [46]. Augmented Lagrangian approaches for both equality and inequality constraints have been developed [36], [38], [41], [74], including the authors' prior work [47] of which this paper is an extension. In contrast to [36], [38], [41], our approach uses a primal-dual Newton step (46) instead of a purely primal step akin to (10). Furthermore, the AL methods in the former papers only produce a coarse solution, which is refined through SQP steps. In contrast, the primal-dual approach allows for convergence to higher accuracy (see [42]) due to better numerical conditioning of the primal-dual system [45].

For inequality constraints, [25]–[27], [71], [75] employ interior-point methods which have been hugely successful in nonlinear programming through popular solvers such as IPOPT [6]. Interior-point methods solve a succession of log barrier-penalized problems that correspond to a relaxation of the KKT complementarity slackness. They can approach the real problem satisfactorily over multiple iterations and tightening of the relaxation. However, they have limited warm-starting capabilities from a previous problem's solution due to resetting the relaxation parameters when the underlying problem has shifted. Related methods include barrier function methods [29] leveraged by [30]–[32], [76]. Among these, the relaxed barrier approach [31] allows use in the MPC context by switching to a quadratic barrier when the underlying inequality constraints are violated. These barrier methods suffer from different limitations: tuning of the barrier parameter, the trade-off with numerical conditioning, and proper satisfaction of the constraints (the relaxed barrier allows violation, and the log barrier does not allow constraints to saturate).

An overview of the different features and methods around alternative DDP/iLQR/Riccati-based approaches of state of the art is summarized in Table II.

VI. Software Implementation Details

Over the past few years, several open-source TO libraries have been developed in the robotics community: CONTROL TOOLBOX [11], OCS2 [60], ALTRO [38], CROCODDYL [18], FATROP [71]. We contribute to this open-source initiative by developing two new software libraries for nonlinear optimization and optimal control in robotics:

1) PROXNLP, which focuses on generic nonlinear programming on manifolds using the method of Section II;
2) ALIGATOR, a library for constrained trajectory optimization, following the approach of Section IV.

These two libraries can be exploited in contexts beyond robotics (e.g., automatic control, biomechanics, etc.). These two new libraries are motivated by the need for efficiency and the lack of flexibility of existing frameworks to easily implement or benchmark new algorithms.

PROXNLP and ALIGATOR are written in C++ using the Eigen [77] linear algebra template library, which is leveraged for efficient computation with minimal dynamic memory allocation at runtime, enabling high-performance scenarios. More specifically, the ALIGATOR package includes the following features:

- an implementation of the ProxDDP algorithm described in this paper;
- the support for problems formulated on Lie groups, which extends the framework that we have presented;
- the support for the PINOCCHIO rigid-body dynamics library [2], which implements analytical derivatives [3], several standard Lie groups in robotics and the support of constrained-based models [78];
- an API frontends in both C++ and Python programming languages: these frontends are built using the same model-data pattern as PINOCCHIO and CROCODDYL for easy caching of previous computations;
- the compatibility with the CROCODDYL [18] trajectory optimization library. ALIGATOR comes with its own implementation of the FDDP algorithm, initially introduced in [18].

Both libraries are templated on the desired scalar type (by default, double precision floating point). In the future, we plan to add the following: support and integration with CasADi [79] and CppAD is planned for automatic differentiation and code generation; support for higher precision floating point types using MPFR; multithreading support through OpenMP; a release on package managers such as APT, rospkg, Homebrew, Conda, and pip.

VII. Experiments

In this section, we present a set of experiments, ranging from simple toy problems to complex trajectory optimization

TABLE II
COMPARISON OF DIFFERENT METHODS AND SOFTWARE LIBRARIES FOR SOLVING OCPS.

Library	Solver	Dynamics handling	Forward pass	Constraints (*approach*)	Globalization
-	Stagewise Newton [8]	Single-shooting	Open-loop	✗	Linesearch
-	Li's iLQR [14]	Single-shooting	Open-loop	✗	Linesearch
CROCODDYL	Vanilla DDP/Tassa's iLQR [15]	Single-shooting	Closed-loop	✗	Linesearch
	FDDP [18]	Multiple-shooting[1]	Closed-loop	✗	Linesearch
	BOX-DDP [23]	Single-shooting	Closed-loop	Control limits (*projection*)	Linesearch
	INTRO[2] [70]	Multiple-shooting	Closed-loop	✓(*projection*)	Linesearch
ALTRO [38]		Multiple-shooting[3]	Closed-loop	✓(*A.L. + projection*)	Linesearch
CONTROL TOOLBOX	GNMS/ILQR-GNMS [11]	Multiple-shooting	Linear/Closed-loop[4]	✗	Linesearch[5]
FATROP [71]		Multiple-shooting	Linear	✓(*interior-point*[6])	Filter[6]
ALIGATOR	**PROXDDP (ours)**	Multiple-shooting	Linear/Closed-loop	✓(*A.L.*)	Linesearch
OCS2 [60]	Constrained SLQ [4]	Single-shooting	Closed-loop	✓(*projection + barrier*)	Linesearch
	Interior-point method	Multiple-shooting	Linear	✓(*interior-point*)	Linesearch
	SQP with HPIPM [25], [26]	Multiple-shooting	Linear	✓(*interior-point* in QP)	Linesearch
ACADOS [72], [73]	SQP/SCQP with HPIPM	Multiple-shooting	Linear	✓(*interior-point*)	Linesearch

[1] The dynamics gap is not taken as a slack variable or included in the linesearch through a merit function.
[2] Multiple-shooting similar to FDDP. This is meant for inverse-dynamics formulations on multibody systems (it exploits the structure in these formulations).
[3] Using a slack variable with a quadratic penalty.
[4] Full GNMS uses the linearized dynamics in the forward sweep (for which open-loop and closed-loop coincide), and the iLQR variants use the nonlinear dynamics in closed-loop.
[5] In the implementation; the paper only deals with full-step iterations.
[6] Equality constraints solved in SQP fashion using equality-constrained LQR [21]. Filter implementation similar to IPOPT [6].

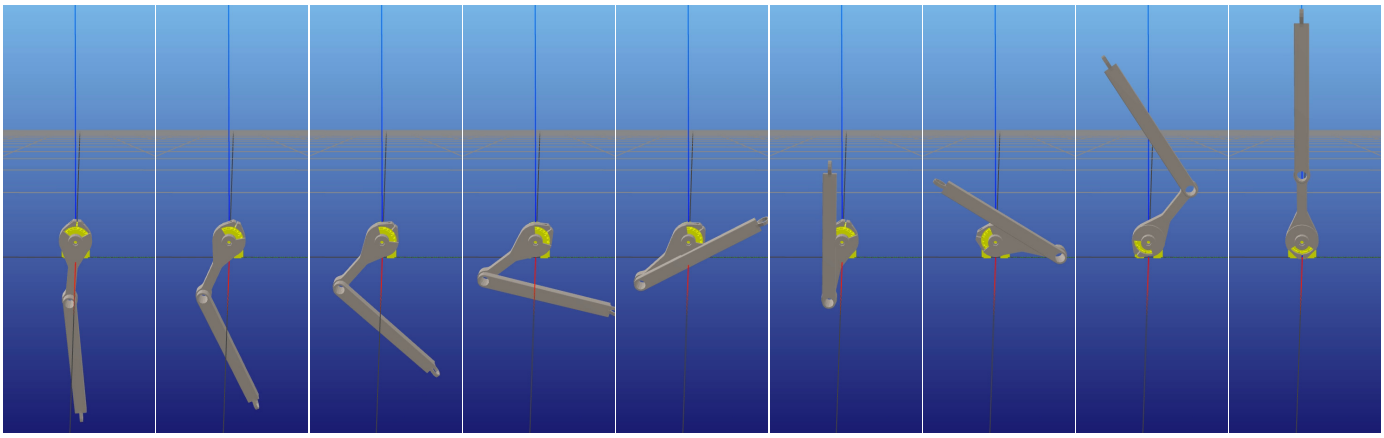


Fig. 2. **3D rendering of the acrobot swing-up task with the terminal constraint.** Rendered using the MeshCat visualizer.

on robotic systems and model-predictive control (MPC) on a real quadruped robot. These experiments showcase the capabilities of our formulation and software contribution regarding flexibility and real-time abilities.

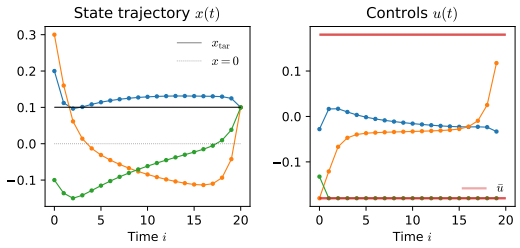
A. Toy examples

Constrained LQR. The linear-quadratic regulator (LQR) is a classical system from the optimal control literature and the most well-known system that can be solved in semi-closed form – in its simplest instance without stagewise constraints such as control limits. Our first example is to show that our solver can solve an LQR with a terminal constraint and additional control bounds $-\bar{u} \leqslant u \leqslant \bar{u}$ in a reasonable number of iterations. The optimized state-control trajectory and convergence of primal-dual residuals are shown in Fig. 3 for the terminal-constraint *and* control-bounded problem.

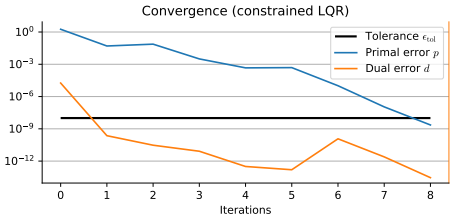
Pendulum. We add both torque limits with $\tau_{\max} = -\tau_{\min} = 20\,\mathrm{N.m}^{-1}$, a terminal constraint on the pendulum's tip, and no terminal cost $\ell_T \equiv 0$. The state-control trajectories are shown in Fig. 4a, and convergence is shown in Fig. 4b. The timestep is $\Delta t = 10\,\mathrm{ms}$.

Cartpole. A case for a very nonlinear, non-convex problem is that of the inverted pendulum on top of a cartpole. The discretization timestep is $\Delta t = 10\,\mathrm{ms}$, and the time horizon is $T = 5\,\mathrm{s}$ The constrained cartpole task is defined to have the terminal position of the end-effector $p_{\mathrm{ee}}(x_N)$ be at $\overline{p} = (0, 1)^\top$ in the yz-plane, and torque limits $-\tau_{\max} \leqslant \tau \leqslant \tau_{\max}$. See Fig. 5 for the resulting state-control and end-effector trajectories and convergence of the iterates.

Acrobot and double pendulum. In the first setting, we impose a terminal constraint on the state. The optimized state-control trajectories are shown in Fig. 6, with a 3D rendering shown in Fig. 2. The state of these systems is $x = ((\cos\theta_1, \sin\theta_1), (\cos\theta_2, \sin\theta_2), \dot{\theta}_1, \dot{\theta}_2) \in \mathrm{SO}(2)^2 \times \mathbb{R}^2$. In the second setting, we impose control torque limits and relax the target reaching task into a quadratic cost: the resulting

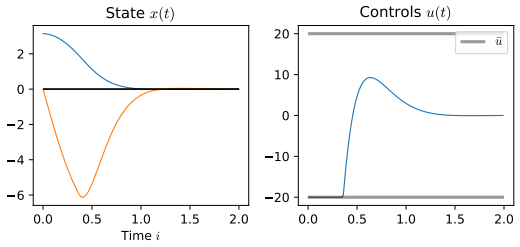


(a) Optimized state and control trajectory. The third coordinate of the control inputs (in green) saturates the control bound in the entire solution save for the first stage.

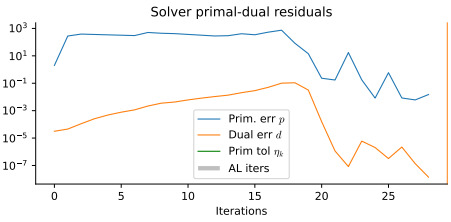


(b) Primal-dual residuals of the problem at each iteration of the semi-smooth Newton method.

Fig. 3. **LQR problem with terminal constraint and control bounds.** Here, the initial ALM parameter is chosen as $\mu_0 = 10^{-6}$, promoting rapid convergence if the subproblem can be solved accurately.



(a) Optimized state and control trajectories. The system state converges to the target state $x = 0$, and the control inputs saturate.



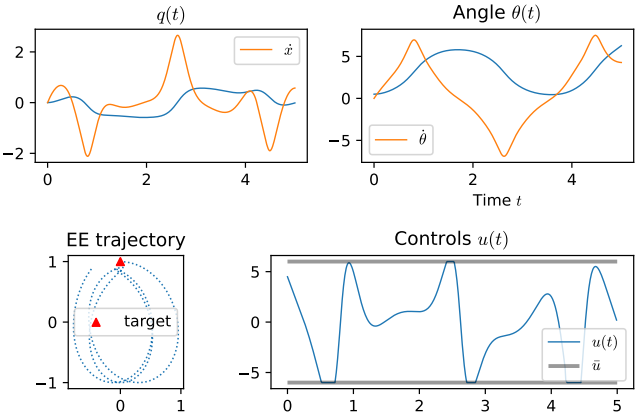
(b) Convergence of the problem primal and dual residuals.

Fig. 4. **Pendulum swing-up problem with terminal constraint and torque limits.** The initial AL parameter is $\mu_0 = 1$.

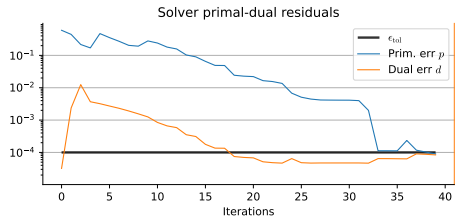
trajectories are shown in Fig. 7. In both settings, the dynamics AL parameter is scaled by 10^{-3} to promote faster convergence on the dynamics constraint. The integration time step is $\Delta t = 5\,\text{ms}$, and the integrator is second-order Runge-Kutta.

B. Ballistics with a manipulator

This second experiment demonstrates the formulation of a simple ballistics task on a manipulator, *without the user specifying a launch velocity* but by specifying a target position $\hat{p}_T \in \mathbb{R}^3$ for the thrown object to reach.



(a) Optimized state and control trajectories and trajectory of the cartpole's end-effector. The thick black lines are the control bounds, which saturate thrice along the time horizon.



(b) Primal-dual residuals along the optimization iterates.

Fig. 5. **Cartpole with both torque limits and a terminal constraint.** We chose an initial parameter of $\mu_0 = 10^{-5}$. The torque limits were chosen to be $\tau_{\max} = 6\,\text{N.m}^{-1}$.

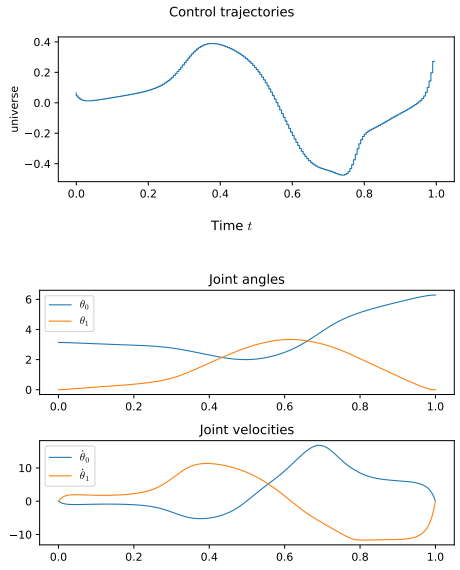


Fig. 6. **Acrobot with terminal constraint.** Optimized state and control trajectories with a terminal constraint $x(T) = ((1,0),(1,0),0,0)$ (i.e., $\theta_1 = \theta_2 = 0 \mod 2\pi$). The initial AL parameter is $\mu_0 = 10^{-3}$.

System model. The system is modelled using PINOCCHIO [2]. The chosen manipulator is a UR10 arm, and a projectile (a mug) is added to the robot's model tree with a free-flyer joint.

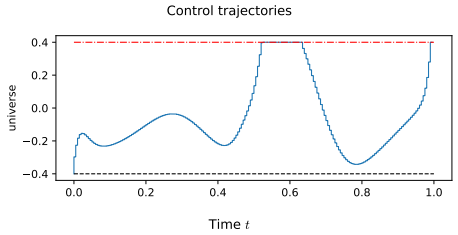


Fig. 7. **Acrobot with control limits.** Optimized control trajectory with control bounds. The initial AL parameter is $\mu_0 = 10^{-3}$.

The system's configuration vector $q = (q_{\text{ob}}, q_{\text{a}})$ consists of the joint configuration $q_{\text{a}} \in \mathbb{R}^{n_{q1}}$ and projectile pose $q_{\text{ob}} \in \text{SE}(3)$. We use a whole-body model for the system dynamics

$$M(q)\ddot{q} + b(q, \dot{q}) + J_c(q)^\top \lambda_c = B\tau \tag{54a}$$
$$c(q) = 0. \tag{54b}$$

where $M(q)$ is the joint space inertia matrix, $\tau \in \mathbb{R}^6$ is the input torque, c is the 6D constraint function between a frame attached to the end-effector and the projectile's frame, $J_c = \partial c(q)/\partial q \in \mathbb{R}^{6\times 12}$ its Jacobian, and $\lambda_c \in \mathbb{R}^6$ is the constraint force between the end-effector and the projectile. These are underactuated dynamics with a selection matrix B which imposes that there is no actuation of the projectile.

Since (54b) is a position-level constraint while the dynamics are solved for acceleration $\ddot{q}$, it is replaced with an acceleration-level constraint:

$$\ddot{c} = J(q)\ddot{q} + \gamma = -K_p c(q) - K_d \dot{c}, \tag{55}$$

where (K_p, K_d) are the Baumgarte stabilization gains [80]. The constrained dynamics model is implemented and solved using the constrained forward dynamics algorithm of [78] implemented in PINOCCHIO.

After a set time $t_c \in [0, T]$ in the time horizon, the projectile detaches from the end-effector and is launched. Thus, for $t > t_c$, the constraint (54b) and contact force λ_c are removed, the robot and projectile dynamics are decoupled, and the projectile is in a flight phase.

Control regularization and initial guess. The gravity-compensating torque (with zero acceleration) for a manipulator without constraints is given by $\tau_0 = b_{\text{a}}(q_0, v_0)$. It can be computed using the recursive Newton-Euler algorithm (RNEA) [1]. When a body is attached to the end-effector, this is not appropriate anymore, and the extra mass means applying the torque τ_0 will still create a nonzero acceleration. We can solve this problem by separating the dynamics with zero acceleration:

$$\begin{bmatrix} b_{\text{a}} \\ b_{\text{ob}} \end{bmatrix} + \begin{bmatrix} J_{c,\text{a}} & J_{c,\text{ob}} \end{bmatrix}^\top \lambda_c = \begin{bmatrix} \tau \\ 0 \end{bmatrix}$$

where $J_{c,\text{a}}$ is the Jacobian with respect to q_{a}. Once equipped with an appropriate τ_0, we use it to construct a quasistatic initial guess, which is used to cold-start the solver, and also use it for the control regularization term in the cost function:

$$\ell_{\text{reg},u}(u) = \frac{1}{2}\|u - \tau_0\|_R^2.$$

The other cost function term is a regularizer on the manipulator joint velocities and, with a lesser weight, the joint configurations. As alluded to at the beginning of this subsection, there are no target position or even target launch velocity costs, and only a hard constraint on the terminal position 3D of the projectile. We expect the optimizer to find a feasible trajectory that satisfies this constraint in this setting. Additionally, we add limits on the manipulator input torque:

$$-\tau_{\max} \leqslant u \leqslant \tau_{\max},$$

where the value for $\tau_{\max}$ is the robot's maximum actuation.

In this example, we used a UR10 manipulator; the projectile is a mug. The optimized control trajectory is shown in Figure 8, with the joint velocities in 9. The graphs show very high-bandwidth behavior in the controls, which is not reflective of what can be achieved on an actual system – modeling the control inputs using, e.g., a spline with a penalty on the ℓ_2 of the controls' time derivative could lead to better behavior. A video rendering of the solution found by the solver is provided in Figure 13 as well as the supplementary video accompanying this submission.

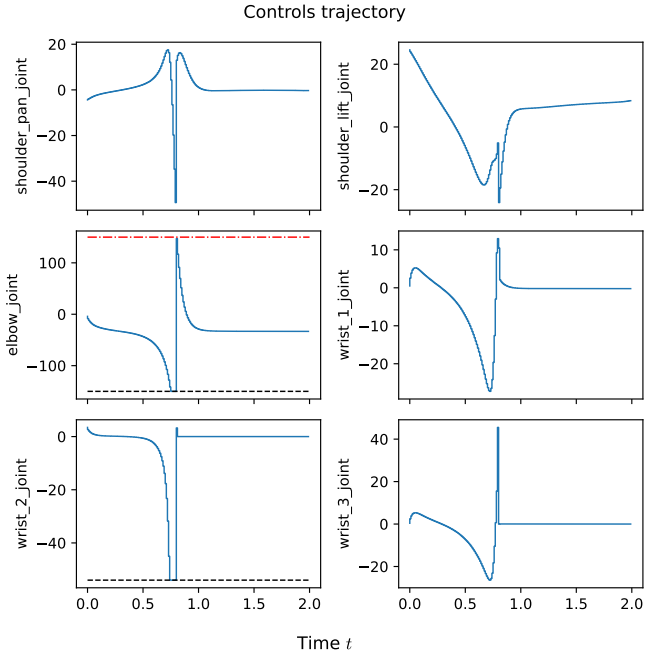


Fig. 8. **Optimized control trajectory for the UR10 ballistic task.** The red (resp. black) dashed line represents the torque upper (resp. lower) limit. The effort (torque) limits were imposed using a hard box constraint in the problem formulation.

C. Motion planning on a quadrotor

We use the Hector quadrotor model[3]. Its configuration variable is its base pose $q \in \text{SE}(3)$, and its dynamics are standard inertial dynamics actuated through the rotors. We build two example problems for the quadrotor system: (i) thrust control limits and (ii) obstacle avoidance.

[3]Its URDF is provided in the example-robot-data package: https://github.com/Gepetto/example-robot-data

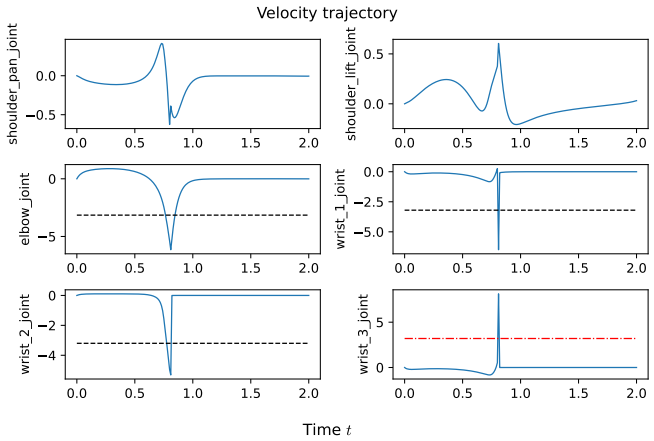


Fig. 9. **Computed joint velocities for the ballistic task on the UR10.** The red (resp. black) dashed line represents the velocity upper (resp. lower) limit. Here, the velocity limits were *not* enforced through constraints or even accounted for in the cost functions.

Thrust limits. In this variant, we add thrust limits to the quadrotor:

$$-\tau_{\max} \leqslant \tau \leqslant \tau_{\max}.$$

The initial AL penalty parameter is set to $\mu_0 = 0.1$. The converged controls and convergence regime in Fig. 10.

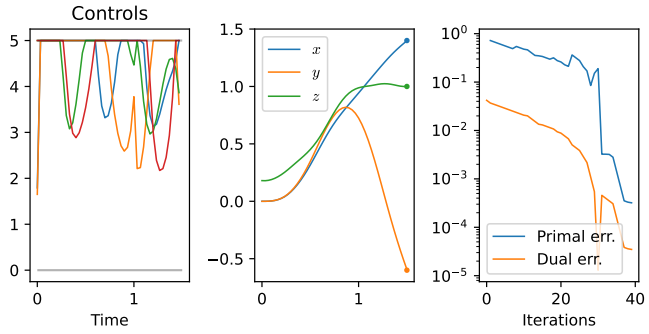


Fig. 10. **Optimized control trajectory and convergence of the quadrotor problem with thrust limits.** The thrust limits often saturate in this setting due to the tight rightward turn needed to move to the next target. The AL penalty scale for the thrust limits is $w_{\text{t.lims}} = 10$.

Obstacles. In this variant, we want the quadrotor to reach an endpoint while "slaloming" between two obstacles represented as cylinders. We use a simple collision model between the cylinders and a bounding sphere around the quadrotor. To solve this problem, we had to switch to a *linear* rollout to keep the solver stable. We set the initial AL penalty parameter in both settings to $\mu_0 = 1$. We obtain the obstacle-avoiding trajectory as rendered in Fig. 14. The controls and convergence regime are shown in Fig. 11.

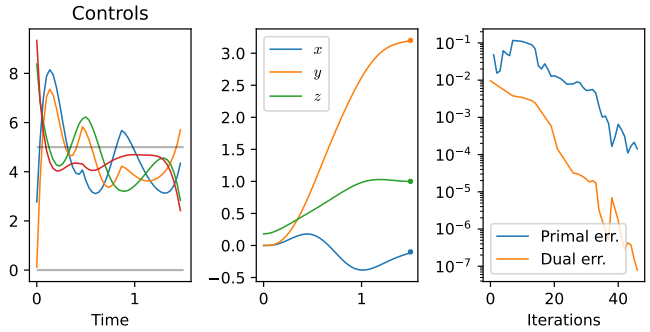


Fig. 11. **Optimized trajectory and convergence of the quadrotor problem with obstacles.** In this setting, the torque limits were not enforced using constraints. The corresponding rendered trajectory is shown in Fig. 14. The AL penalty for the obstacles is scaled down by a factor $w_{\text{obs}} = 0.1$.

D. Whole-body planning of a jump on a quadruped

The system is also modeled using a whole-body model with bilateral contact constraints as in (54b):

$$M(q)\ddot{q} + b(q,\dot{q}) + \sum_{c\in\mathcal{C}_t} J_c(q)^\top \lambda_c = B\tau \tag{56a}$$

$$c(q) = 0, \tag{56b}$$

where $\mathcal{C}_t$ is the set of contact points (feet) at time t. The contact constraint is a 3-dimensional position constraint concerning points on the ground. The problem is modeled in three phases: (i) all four feet in contact, (ii) no contact (flight phase), and (iii) contact once again (landing phase). The phases are chosen to start at times $t = 0$, $t = t_0 > 0$, and $t = t_1 > t_0$, respectively.

In this problem, the constraint is also replaced by an acceleration constraint with Baumgarte gains

$$\ddot{c} = J(q)\ddot{q} + \gamma = -K_p c(q) - K_d \dot{c}.$$

Our desired behavior is to have zero constraint spatial acceleration and velocity and enforce that the altitude of the feet is $z_c(q) = 0$. Thus, the xy-plane position-level gains in K_p are set to zero. To have the planned foot motion hit the ground at $t = t_1$, we add a position-level hard constraint:

$$z_c(q(t_1)) = 0,$$

which the proximal optimizer can account for.

The generated motion is illustrated in Figure 15. The convergence of the problem residuals as the solver progresses is shown in Fig. 12.

E. Model-predictive control on the SOLO-12 *quadruped*

To demonstrate the capabilities of our package when it comes to real-time whole-body MPC on the SOLO quadruped on uneven terrain, as illustrated on the screenshot sequences in Fig. 16 and in the companion video associated with this article. In this demonstration, we solve whole-body optimal control problems without pre-defining reference foot trajectories, as done in Assirelli et al. [17]. The desired robot base linear and angular velocities are provided using a gamepad.

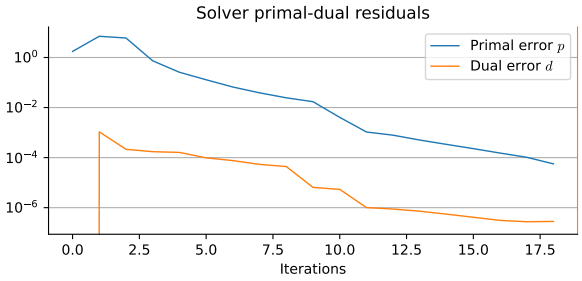


Fig. 12. **Convergence of the primal-dual residuals for the SOLO-12 jumping task.** The initial AL parameter is $\mu_0 = 10^{-4}$.

Hardware. The Solo quadruped is connected over Ethernet to a 2019 Dell XPS 13 laptop running under Ubuntu Linux 20.04, patched with a real-time kernel. The whole-body MPC runs in a separate ROS [81] node on an Apple Mac Studio M1 Ultra desktop computer, with communication between the two over LAN.

Control framework. We have adapted the framework developed in [17] to leverage the real-time computational efficiency of the PROXDDP algorithm implemented in ALIGATOR. This framework is composed of:

- a state estimation module using a complementary filter initially described in [82], which estimates the robot base position and linear velocity,
- a lower-level, higher-frequency whole-body controller using the first Riccati gain and feedforward control, which outputs a desired torque:

$$\overline{\tau}(t) \stackrel{\text{def}}{=} u_0 + K_0 \left(\hat{x}(t) \ominus \widehat{x}_{\text{ocp}}(t) \right) \tag{57}$$

  where $\widehat{x}_{\text{ocp}}$ is an upsampling of the OCP solution $\boldsymbol{x}_{\text{ocp}}$ using e.g. interpolation or integrating the system dynamics.

In our experiments, the MPC loop frequency is $\Delta t_{\text{mpc}} = 12\,\text{ms}$ and runs over a prediction horizon of 0.5s. The MPC formulation accounts for robot position, velocity, and torque limits. The cost function is defined by:

$$\begin{aligned} \ell(x,u) = \ell_{\text{fly}}(x) + \ell_{\text{reg}}(x,u) + \ell_g(x) \\ + \ell_{bv}(x) + \ell_{bu}(u) + \ell_{\text{base}}(x; \hat{v}_{\text{base}}) \end{aligned} \tag{58}$$

All the cost function terms are summarized in Table III. The index j spans over the quadruped's four feet, z^j is the altitude of the j-th foot, $v^j \in \mathbb{R}^3$ its linear velocity (and $v^j_{\parallel}$ its component along the xy-plane), λ are the contact forces for all legs (in contact).

The low-level Riccati control (57) is sampled at a frequency of $1\,\text{kHz}$. Each MPC update consists of a single step of the trajectory optimization algorithm; in practice, it was solved on the Mac M1 hardware in less than $6\,\text{ms}$. The code associated with this MPC framework is publicly available[4].

In Fig. 16 and in the companion video, we can observe that SOLO-12 exhibits smooth behavior and successfully walks on uneven terrain even though the terrain is assumed to be flat in the prediction horizon.

[4]https://gitlab.laas.fr/gepetto/quadruped-reactive-walking/-/tree/wjallet/devel

VIII. CONCLUSION

In this paper, we propose a unified formulation for solving constrained trajectory optimization problems leveraging a primal-dual augmented Lagrangian approach while exploiting the underlined temporal structure of OCPs. This results in two novel algorithms: (i) PROXNLP for generic nonlinear programming problems on manifolds, and (ii) PROXDDP specifically suited for constrained trajectory optimization. We have empirically demonstrated the efficiency of our solver in coping with several real-size optimal control problems for various classes of robots requiring hard constraint satisfaction. Used inside an MPC scheme to perform whole-body control on a quadrupedal robot, we have particularly demonstrated that our solver has no difficulty taking advantage of a warm-start. We are also releasing an efficient C++ implementation with Python bindings to allow reproduction of our experiments, on which we hope other teams will be able to build more complex behaviors on various robots. We hope this work will entice future developments in robust, real-time capable, constrained trajectory optimization solvers, especially their application to model-predictive control schemes.

In future work, we will consider delving deeper into a couple of subjects leveraging the proposed framework. First, we plan to extend these software packages, taking feedback from the community and extending its range of applications towards more real-world scenarios. We expect this to be a continuous work akin to what is being done with both PINOCCHIO and CROCODDYL. We expect ALIGATOR to become a catalyzer in robotics, facilitating the deployment of real-time receding horizon schemes, thus advancing and simplifying the generation of complex motions on modern robots. Second is further work on structure-exploiting linear algebra solvers, both for the subproblems (46) as well as parallelizing of the backward pass similar to [83], [84]. We expect this to accelerate the timing performance of our library significantly, making possible online whole-body model predictive control on more complex robotic systems such as humanoids and dexterous hands or facilitating the deployment of MPC solutions on embedded devices with limited computational resources.

REFERENCES

[1] R. Featherstone, *Rigid Body Dynamics Algorithms*. Boston, MA: Springer US, 2008.

[2] 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 International Symposium on System Integrations (SII)*, 2019.

[3] J. Carpentier and N. Mansard, "Analytical derivatives of rigid body dynamics algorithms," in *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, Jun. 2018.

[4] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE Press, May 2017, pp. 93–100.

[5] M. Diehl, H. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast Motions in Biomechanics and Robotics*, M. Diehl and K. Mombaur, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, vol. 340, pp. 65–93.

[6] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006.

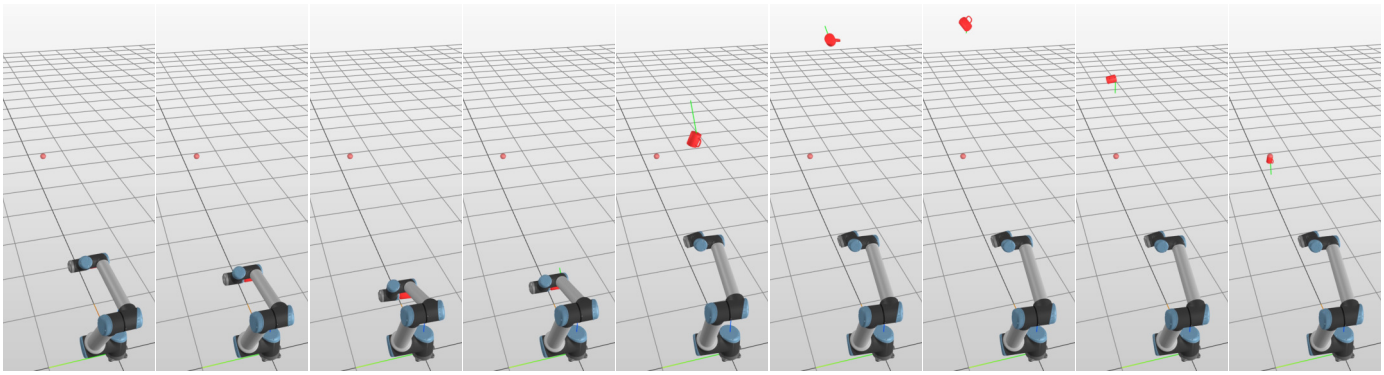


Fig. 13. Optimized motion for the UR10 ballistic task, rendered using the MeshCat visualizer.

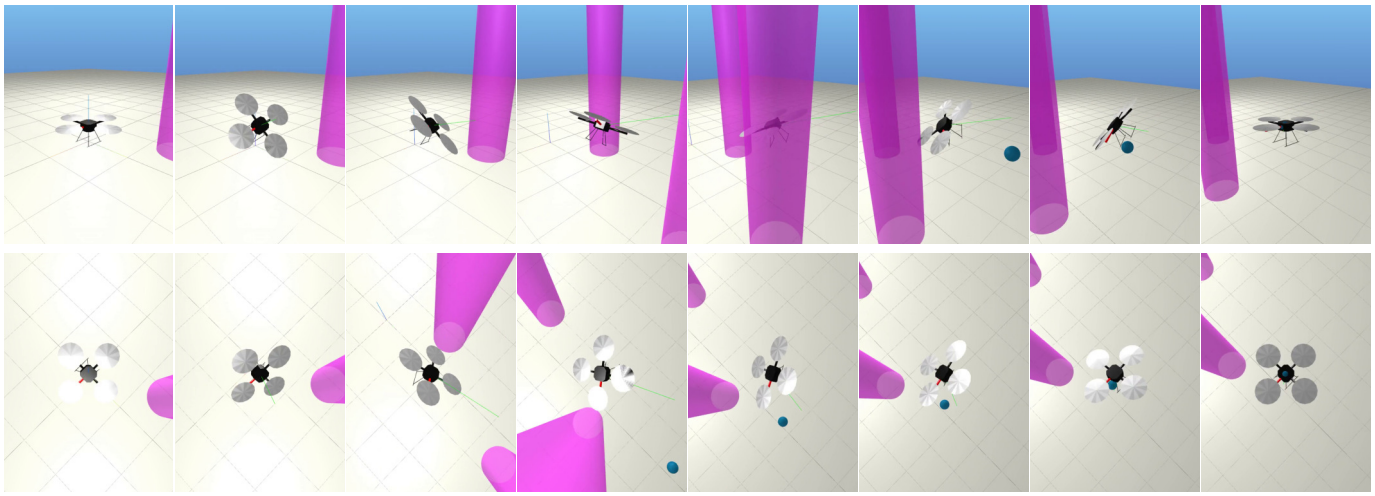


Fig. 14. Trajectory of the quadrotor avoiding obstacles, rendered using the MeshCat visualizer. Above and below are two different viewpoints.

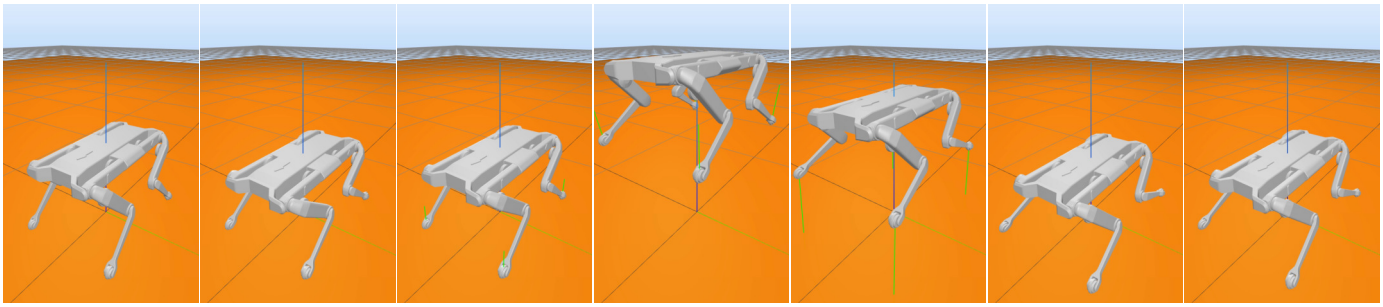


Fig. 15. 3D rendering of the SOLO-12 jumping task. Rendered using the MeshCat visualizer. The time step is $\Delta t = 20\,\text{ms}$ and time horizon $T = 1.2\,\text{s}$.

[7] P. Gill, W. Murray, and M. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, vol. 12, pp. 979–1006, Apr. 2002.

[8] J. F. A. De O. Pantoja, "Differential dynamic programming and newton's method," *International Journal of Control*, vol. 47, no. 5, pp. 1539–1553, May 1988.

[9] J. C. Dunn and D. P. Bertsekas, "Efficient dynamic programming implementations of newton's method for unconstrained optimal control problems," *Journal of Optimization Theory and Applications*, vol. 63, no. 1, pp. 23–38, Oct. 1989.

[10] M. Neunert, C. Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1398–1404.

[11] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

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

[13] D. M. Murray and S. J. Yakowitz, "Differential dynamic programming and newton's method for discrete optimal control problems," *Journal of Optimization Theory and Applications*, vol. 43, no. 3, pp. 395–414, Jul. 1984.

[14] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics*. Setúbal, Portugal: SciTePress - Science and and Technology Publications, 2004, pp. 222–229.

[15] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.*,

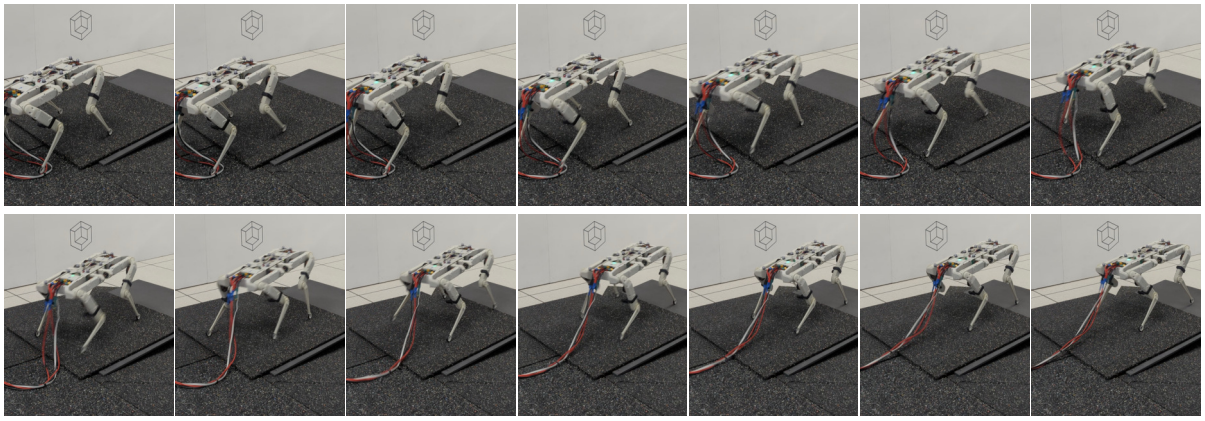


Fig. 16. SOLO-12 walking task. The time step is $\Delta t = 12\,\text{ms}$ and time horizon $T = 0.480\,\text{s}$. The slope and bumps of the terrain are unmodeled and act as perturbations.

TABLE III
TERMS OF THE COST FUNCTION FOR THE MPC EXPERIMENT.

Name & symbol	Expression	Parameters
Fly-high cost ℓ_{fly}	$\sum_j e^{-\gamma z^j} \|v_\parallel^j\|^2_{w_{\text{fly}}}$	$\gamma = 50$, $w_{\text{fly}} = 5.10^4$
Ft. height bound ℓ_g	$\sum_j \|\max(z^j, z^{\text{ref}})\|^2_{w_g}$	$w_g = 10^3$
Spatial vel. tracking ℓ_{base}	$\|v_{\text{base}} - \hat{v}_{\text{base}}\|^2_{w_{\text{base}}}$	$w_{\text{base}} = 8.10^5$
Vel. bounds ℓ_{bv}	$\|\max(\min(v, \underline{v}), \overline{v})\|^2_{w_{b_v}}$	$w_{b_v} = 10 \times [\mathbf{0}_6;\ \mathbf{1}_{n_v-6}]$
Ctrl. bounds ℓ_{bu}	$\|\max(\min(u, \underline{u}), \overline{u})\|^2_{w_{b_u}}$	$w_{b_u} = 10^4$
Regularization ℓ_{reg}	$\|x - x^{\text{ref}}\|^2_{w_x} + \|u\|^2_{w_u} + \underbrace{\|\lambda - \lambda^{\text{ref}}\|^2_{w_\lambda}}_{\text{contact forces}}$	$w_u = 10^4$, $w_\lambda = 10^2$, $\lambda^{\text{ref}} = (0, 0, \frac{mg}{n_{\text{contacts}}})^\top$
Impact cost $\ell_{\text{imp.}}$	$\sum_j \|z^j - z^{\text{ref}}\|_{w_h} + \|v^j\|_{w_i}$	$w_h = w_i = 10^4$

Oct. 2012, pp. 4906–4913.

[16] E. Dantec, R. Budhiraja, A. Roig, T. Lembono, G. Saurel, O. Stasse, P. Fernbach, S. Tonneau, S. Vijayakumar, S. Calinon, M. Taix, and N. Mansard, "Whole body model predictive control with a memory of motion: Experiments on a torque-controlled talos," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 8202–8208.

[17] A. Assirelli, F. Risbourg, G. Lunardi, T. Flayols, and N. Mansard, "Whole-body mpc without foot references for the locomotion of an impedance-controlled robot."

[18] 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," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2536–2542, May 2020.

[19] M. Giftthaler and J. Buchli, "A projection approach to equality constrained iterative linear quadratic optimal control," *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pp. 61–66, Nov. 2017.

[20] F. Laine and C. Tomlin, "Efficient computation of feedback control for equality-constrained lqr," in *2019 International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 6748–6754.

[21] L. Vanroye, J. De Schutter, and W. Decré, "A generalization of the riccati recursion for equality-constrained linear quadratic optimal control," Jun. 2023.

[22] T. A. Davis, "Algorithm 849: A concise sparse cholesky factorization package," *ACM Transactions on Mathematical Software*, vol. 31, no. 4, pp. 587–591, Dec. 2005.

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

[24] C. Mastalli, W. Merkt, J. Marti-Saumell, H. Ferrolho, J. Solà, N. Mansard, and S. Vijayakumar, "A feasibility-driven approach to control-limited ddp," *Autonomous Robots*, vol. 46, no. 8, pp. 985–1005, Dec. 2022.

[25] G. Frison and M. Diehl, "Hpipm: A high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, Jan. 2020.

[26] G. Frison, J. Frey, F. Messerer, A. Zanelli, and M. Diehl, "Introducing the quadratically-constrained quadratic programming framework in hpipm," in *2022 European Control Conference (ECC)*, Jul. 2022, pp. 447–453.

[27] A. Pavlov, I. Shames, and C. Manzie, "Interior point differential dynamic programming," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 6, pp. 2720–2727, Nov. 2021.

[28] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[29] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research. New York: Springer, 2006.

[30] J. Hauser and A. Saccon, "A barrier function method for the optimization of trajectory functionals with constraints," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Dec. 2006, pp. 864–869.

[31] C. Feller and C. Ebenbauer, "Relaxed logarithmic barrier function based model predictive control of linear systems," *IEEE Transactions on Automatic Control*, vol. 62, Mar. 2015.

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

[33] M. R. Hestenes, "Multiplier and gradient methods," *Journal of Optimization Theory and Applications*, vol. 4, no. 5, pp. 303–320, Nov. 1969.

[34] R. T. Rockafellar, "The multiplier method of hestenes and powell applied to convex programming," *Journal of Optimization Theory and Applications*, vol. 12, no. 6, pp. 555–562, Dec. 1973.

[35] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Lancelot: A Fortran Package*

for Large-Scale Nonlinear Optimization (Release A), 1st ed. Springer Publishing Company, Incorporated, Nov. 2010.

[36] M. Toussaint, "A novel augmented lagrangian approach for inequalities and convergent any-time non-central updates," Dec. 2014.

[37] ——, "A tutorial on newton methods for constrained trajectory optimization and relations to slam, gaussian process smoothing, optimal control, and probabilistic inference," in *Geometric and Numerical Foundations of Movements*, J.-P. Laumond, N. Mansard, and J.-B. Lasserre, Eds. Cham: Springer International Publishing, 2017, vol. 117, pp. 361–392.

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

[39] B. E. Jackson, T. Punnoose, D. Neamati, K. Tracy, R. Jitosho, and Z. Manchester, "Altro-c: A fast solver for conic model-predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 7357–7364.

[40] J.-P. Sleiman, F. Farshidian, and M. Hutter, "Constraint handling in continuous-time ddp-based model predictive control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 8209–8215.

[41] Y. Aoyama, G. Boutselis, A. Patel, and E. A. Theodorou, "Constrained differential dynamic programming revisited," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 9738–9744.

[42] S. Kazdadi, J. Carpentier, and J. Ponce, "Equality constrained differential dynamic programming," in *ICRA 2021 - IEEE International Conference on Robotics and Automation*, May 2021.

[43] A. Conn, N. Gould, and P. Toint, "A globally convergent augmented lagrangian algorithm for optimization with general constraints and simple bounds," *SIAM Journal on Numerical Analysis*, vol. 28, Apr. 1991.

[44] B. Hermans, A. Themelis, and P. Patrinos, "Qpalm: A newton-type proximal augmented lagrangian method for quadratic programs," *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4325–4330, Dec. 2019.

[45] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "Prox-qp: Yet another quadratic programming solver for robotics and beyond," in *Robotics: Science and Systems XVIII*. Robotics: Science and Systems Foundation, Jun. 2022.

[46] W. Jallet, N. Mansard, and J. Carpentier, "Implicit differential dynamic programming," in *2022 International Conference on Robotics and Automation (ICRA)*. Philadelphia, United States: IEEE Robotics and Automation Society, May 2022.

[47] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, "Constrained differential dynamic programming: A primal-dual augmented lagrangian approach," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyoto, Japan, Oct. 2022.

[48] A. De Marchi, "On a primal-dual newton proximal method for convex quadratic programs," *Computational Optimization and Applications*, vol. 81, no. 2, pp. 369–395, Mar. 2022.

[49] E. G. Birgin and J. Martínez, "Improving ultimate convergence of an augmented lagrangian method," *Optimization Methods and Software*, vol. 23, no. 2, pp. 177–195, Apr. 2008.

[50] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[51] A. De Marchi, "Augmented lagrangian methods as dynamical systems for constrained optimization," in *2021 60th IEEE Conference on Decision and Control (CDC)*. Austin, TX, USA: IEEE, Dec. 2021, pp. 6533–6538.

[52] R. T. Rockafellar, "Augmented lagrangians and applications of the proximal point algorithm in convex programming," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 97–116, 1976.

[53] ——, "Advances in convergence and scope of the proximal point algorithm," *Journal of Nonlinear and Convex Analysis*, vol. 22, no. 11, pp. 2347–2375, Nov. 2021.

[54] ——, "Monotone operators and the proximal point algorithm," *SIAM Journal on Control and Optimization*, vol. 14, no. 5, pp. 877–898, Aug. 1976.

[55] E. G. Birgin, J. M. Martínez, and M. Raydan, "Nonmonotone spectral projected gradient methods on convex sets," *SIAM Journal on Optimization*, vol. 10, no. 4, pp. 1196–1211, Jan. 2000.

[56] P. E. Gill and D. P. Robinson, "A primal-dual augmented lagrangian," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 1–25, Jan. 2012.

[57] N. Boumal, *An Introduction to Optimization on Smooth Manifolds*. Cambridge University Press, Mar. 2023.

[58] D. Bertsekas, *Dynamic Programming and Optimal Control*, ser. Athena Scientific optimization and computation series. Athena Scientific, 2000, no. v. 2. [Online]. Available: https://books.google.fr/books?id=9WUZSAAACAAJ

[59] 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 *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Hamburg, Germany: IEEE, Sep. 2015, pp. 3346–3351.

[60] F. Farshidian *et al.*, "OCS2: An open source library for optimal control of switched systems," [Online]. Available: https://github.com/leggedrobotics/ocs2.

[61] F. H. Clarke, *Optimization and nonsmooth analysis*. SIAM, 1990.

[62] M. Hintermuller, "Semismooth newton methods and applications," Tech. Rep.

[63] L.-Z. Liao and C. Shoemaker, "Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems," Feb. 1993.

[64] I. Chatzinikolaidis and Z. Li, "Trajectory optimization of contact-rich motions using implicit differential dynamic programming," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2626–2633, Apr. 2021.

[65] J. Hauser, "A projection operator approach to the optimization of trajectory functionals," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 377–382, Jan. 2002.

[66] F. Bayer, G. Notarstefano, and F. Allgöwer, *A Projected SQP Method for Nonlinear Optimal Control with Quadratic Convergence*, Dec. 2013.

[67] A. Saccon, J. Hauser, and A. P. Aguiar, "Optimal control on lie groups: The projection operator approach," *IEEE Transactions on Automatic Control*, vol. 58, no. 9, pp. 2230–2245, Sep. 2013.

[68] L. Sforni, S. Spedicato, I. Notarnicola, and G. Notarstefano, "Gopronto: A feedback-based framework for nonlinear optimal control," Aug. 2021.

[69] S. J. Wright and M. J. Tenny, "A feasible trust-region sequential quadratic programming algorithm," *SIAM Journal on Optimization*, vol. 14, no. 4, pp. 1074–1105, Jan. 2004.

[70] C. Mastalli, S. P. Chhatoi, T. Corbéres, S. Tonneau, and S. Vijayakumar, "Inverse-dynamics mpc via nullspace resolution," *IEEE Transactions on Robotics*, pp. 1–20, 2023.

[71] L. Vanroye, A. Sathya, J. De Schutter, and W. Decré, "Fatrop : A fast constrained optimal control problem solver for robot trajectory optimization and control," Mar. 2023.

[72] R. Verschueren, G. Frison, D. Kouzoupis, Niels van Duijkeren, Andrea Zanelli, R. Quirynen, and Moritz Diehl, "Towards a modular software package for embedded optimization," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 374–380, Jan. 2018.

[73] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, Niels van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "Acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, no. 1, pp. 147–183, Mar. 2022.

[74] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, Jun. 2018.

[75] T. A. Howell, S. L. Cleac'h, K. Tracy, and Z. Manchester, "Calipso: A differentiable solver for trajectory optimization with conic and complementarity constraints," May 2022.

[76] H. Li and P. M. Wensing, "Hybrid systems differential dynamic programming for whole-body motion planning of legged robots," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5448–5455, Oct. 2020.

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

[78] J. Carpentier, R. Budhiraja, and N. Mansard, "Proximal and sparse resolution of constrained dynamic equations," in *Robotics: Science and Systems XVII*. Robotics: Science and Systems Foundation, Jul. 2021.

[79] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[80] J. Baumgarte, "Stabilization of constraints and integrals of motion in dynamical systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 1, no. 1, pp. 1–16, Jun. 1972.

[81] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, *ROS: An Open-Source Robot Operating System*, Jan. 2009, vol. 3.

[82] P.-A. Léziart, T. Flayols, F. Grimminger, N. Mansard, and P. Souères, "Implementation of a reactive walking controller for the new open-hardware quadruped solo-12," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an, China: IEEE Press, May 2021, pp. 5007–5013.
[83] F. Laine and C. Tomlin, "Parallelizing lqr computation through endpoint-explicit riccati recursion," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, Dec. 2019, pp. 1395–1402.
[84] B. E. Jackson and Z. Manchester, "A parallell linear system solver for optimal control."

APPENDIX

A. Details on the augmented Lagrangian method

Expression of the first-order estimate. Recall the initial definition of the augmented Lagrangian:

$$\mathcal{L}_\mu(z;\lambda_e,\nu_e) = \max_{\substack{(\lambda,\nu)\\ \nu\geqslant 0}} \mathcal{L}(z,\lambda,\nu) - \tfrac{\mu}{2}\|(\lambda,\nu)-(\lambda_e,\nu_e)\|_2^2.$$

The stationarity conditions over λ read:

$$g(z)-\mu(\lambda-\lambda_e)=0 \Longleftrightarrow \lambda = \lambda_e + \tfrac{1}{\mu}g(z).$$

And the conditions over ν read:

$$h(z)-\mu(\nu-\nu_e)\in\mathcal{N}_{\mathbb{R}^n_+}(\nu)$$

where $\mathcal{N}_{\mathbb{R}^n_+}(\nu)$ is the normal cone to $\mathbb{R}^n_+$ at ν: it being non-empty implies $\nu\geqslant 0$. Rewriting this condition:

$$(\nu_e+\tfrac{1}{\mu}h(z))-\nu\in\mathcal{N}_{\mathbb{R}^n_+}(\nu)$$

which leads to applying the projection on $\mathbb{R}^n_+$:

$$\nu = \mathrm{proj}_{\mathbb{R}^n_+}\left(\nu_e+\tfrac{1}{\mu}h(z)\right).$$

Interpretation as a shifted-penalty method. The method of multipliers can also be seen as a standard ℓ_2-penalty method with varying penalty weights $(\mu_k)_k$ to solve a cascade of problems with shifted constraints of the form:

$$\begin{aligned}\min_{z\in\mathcal{Z}}\ & f(z)\\ \text{s.t. }& g(z)+\mu_k\lambda_k=0\\ & h(z)+\mu_k\nu_k\leqslant 0,\end{aligned}\tag{$\mathcal{P}_k$}$$

where $(\lambda_k)_{k\geqslant 0},(\nu_k)_{k\geqslant 0}$ are a sequence of vectors, so that the $(\mu_k\lambda_k,\mu_k\nu_k)$ can be interpreted as "slack variables". Applying the aforementioned penalty method leads to the sequence of unconstrained minimization problems:

$$\begin{aligned}&\min_{z\in\mathcal{Z}} f(z)+\mathscr{P}_k(z),\ \text{where}\\ &\mathscr{P}_k(z)\overset{\text{def}}{=}\frac{1}{2\mu_k}\left(\|g(z)+\mu_k\lambda_k\|_2^2+\|[h(z)+\mu_k\nu_k]_+\|_2^2\right).\end{aligned}$$

The objective $f(z)+\mathscr{P}_k(z)$ is precisely the augmented Lagrangian function $\mathcal{L}_{\mu_k}$.

B. Primal-dual augmented Lagrangian functions

In the NLP case, we can derive a primal-dual merit function as in De Marchi [48] by starting with a slack variable s and the Gill-Robinson primal-dual function [56] for the equality-constrained case. Let us consider the only-inequality constrained case for simplicity:

$$\min_x f(x)\quad \text{s.t. } g(x)\leqslant 0.$$

We introduce the slack variable $s\leqslant 0$ and add the constraint that $g(x)=s$. The primal-dual augmented Lagrangian for this problem is then:

$$\bar{\mathcal{M}}_k(x,s,\lambda) = f(x)+\frac{1}{\mu_k}\|g(x)-s+\mu_k(\lambda_k-\tfrac{\lambda}{2})\|_2^2+\frac{\mu_k}{4}\|\lambda\|^2.\tag{59}$$

Minimizing over $s\leqslant 0$ leads to the minimum point

$$s_k^* = [g(x)+\mu_k(\lambda_k-\lambda/2)]_-,\tag{60}$$

and minimum value

$$\begin{aligned}\mathcal{M}_k(x,\lambda)&\overset{\text{def}}{=}\bar{\mathcal{M}}_k(x,s^*,\lambda)\\ &= f(x)+\frac{1}{\mu_k}\|[g(x)+\mu_k(\lambda_k-\lambda/2)]_+\|+\frac{\mu_k}{4}\|\lambda\|^2.\end{aligned}\tag{61}$$

Define the primal-dual multiplier update:

$$\lambda_k^\circ(x,\lambda)=\frac{2}{\mu_k}(g+\mu_k(\lambda_k-\lambda/2)-s)=\frac{2}{\mu_k}[g+\mu_k(\lambda_k-\lambda/2)]_+\tag{62}$$

The gradients are

$$\begin{aligned}\nabla_x\mathcal{M}_k(x,\lambda)&=\nabla f(x)+J(x)^\top\lambda_k^\circ(x,\lambda)\\ \nabla_\lambda\mathcal{M}_k(x,\lambda)&=\frac{\mu_k}{2}(\lambda-\lambda_k^\circ(x,\lambda)).\end{aligned}\tag{63}$$

Note that the stationarity conditions in this primal-dual function at a point (x^*,λ^*) imply directly $\lambda^*=\lambda_k^\circ(x^*)$ and $\nabla f(x^*)+J(x^*)^\top\lambda_k^\circ(x^*)=0$. Again, the equality-constrained case of [56] reduces to $\nabla_\lambda\mathcal{M}_k=\mu_k(\lambda-\lambda_k^+(x))$.

The BCL-type update is given by:

$$\begin{aligned}\lambda_{k+1}&=\lambda_k+\frac{1}{\mu_k}\nabla_\lambda\mathcal{M}_k(x^*,\lambda^*)\\ &=\lambda_k+\frac{1}{2}\left(\lambda^*-\lambda_k^\circ(x^*,\lambda^*)\right).\end{aligned}\tag{64}$$

C. Extensions for the PROXDDP algorithm

More general initial conditions. We can generalize to problems that are not initial value problems but where x_0 is itself a decision variable, with a generic constraint $\omega(x_0)=0$. For instance, where only part of the state space is known (e.g., position but not velocity). The corresponding SQP step leads to solving the following linear system:

$$\begin{bmatrix}W_0 & \omega_x^\top\\ \omega_x & -\mu_k I\end{bmatrix}\begin{bmatrix}\delta x_0\\ \delta\lambda_0\end{bmatrix}=-\begin{bmatrix}V_{x0}+\omega_x^\top\lambda_0\\ \mu_k(\lambda_{k,0}-\lambda_0)+\omega(x_0)\end{bmatrix},\tag{65}$$

where $W_0\approx\nabla^2_{x_0}(V_0+\lambda_0^\top\omega)$ (e.g. $W=V_{xx0}$).

Per-constraint AL scaling. Different constraints in an optimization problem can have different scales regarding the values they take on the search space. For this reason, one may adopt different weights for each constraint. For simplicity, consider a weight w_d for the dynamical constraint w_i for the inequality constraints. Then, the AL parameter μ is replaced by $w_d\mu$ and $w_i\mu$ respectively. The KKT matrix and right-hand side of equation (43), as well as the linesearch penalties, are all adjusted accordingly.

D. Relationship between PROXDDP *and multiple shooting*

Similarly to [42], we can solve multiple shooting formulations with an equality constraint-capable DDP by using slack variables for the dynamics $\boldsymbol{s} = (s_t)_t$:

$$x_{t+1} = f(x_t, u_t) - s_t,\ 1 \leqslant i \leqslant N-1,$$

or equivalently $f(x_t, u_t) - x_{t+1} = s_t$, and add the constraint that $s_t = 0$. The trajectory optimization problem becomes:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{s}}\ & J(\boldsymbol{x},\boldsymbol{u}) \\ \text{s.t. } & x_{t+1} = f(x_t, u_t) - s_t, \\ & s_t = 0. \end{aligned} \tag{66}$$

To solve this using a DDP-like algorithm, let us introduce the following Hamiltonian function for this problem:

$$Q_t(x, u, s, \lambda) = \ell_t(x, u) + V_{t+1}(f(x, u) - s) + \lambda^\top s. \tag{67}$$

Using the shorthands $Q = Q_t$, $V' = V_{t+1}$, the first-order necessary conditions satisfied by the quantities $(u^\star, s^\star, \lambda^\star)$ at step i are:

$$\begin{aligned} 0 &= Q_u = \ell_u + f_u^\top V'_x \\ 0 &= Q_s = -V'_x + \lambda^\star \\ 0 &= s^\star. \end{aligned} \tag{68}$$

We extract from these optimality conditions (i) the standard stopping criterion from DDP, $\ell_u + f_u^\top V'_x = 0$, and (ii) a link between the co-state and value function gradient: $\lambda^\star = -V'_x$.

However, using an augmented Lagrangian formulation akin to [42], the optimality conditions in (u, s, λ) of the proximal subproblem are:

$$\begin{bmatrix} \ell_u + f_u^\top V'_x \\ -V'_x + \lambda \\ -s + \mu_k(\lambda_k - \lambda) \end{bmatrix} = 0.$$

When the inner subproblem is exactly solved, this formulation leads to the slack variable being *equal* to the outer iteration multiplier gap in the PROXDDP algorithm:

$$s^\star = \mu_k(\lambda_k - \lambda_{k+1}),$$

where $\lambda_{k+1} = \lambda^\star$. Thus, we have that the primal-dual scheme induces a slack in the system dynamics.