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 has been a popular choice for motion generation and control in robotics for at least a decade. Several numerical approaches have exhibited the required speed to enable online computation of trajectories for real-time of various systems, including complex robots. Many of these said are based on the differential dynamic programming (DDP) algorithm – initially designed for unconstrained trajectory optimization problems – and its variants, which are relatively easy to implement and provide good runtime performance. However, several problems in robot control call for using constrained formulations (e.g. torque limits, obstacle avoidance), from which several difficulties arise when trying to adapt DDP-type methods: numerical stability, computational efficiency, and constraint satisfaction. In this article, we leverage proximal methods for constrained optimization and introduce a DDP-type method for fast, constrained trajectory optimization suited for model-predictive control (MPC) applications with easy warm-starting. Compared to earlier solvers, our approach effectively manages hard constraints without warm-start limitations and exhibits good convergence behavior. 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 (TO) is a versatile and still-evolving 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. Recent progress in both software and hardware has enabled computing the quantities in trajectory optimisation (TO) (e.g., rigid-body kinematics, dynamics and their derivatives [1]–[3]).

Continuous-time optimal control problems (OCPs) are infinite-dimensional optimization problems, which are generally not solvable in closed form. One approach to solving them approximately is by casting them into nonlinear programs (NLPs) of finite dimension, using, e.g., *direct* methods [4]. Direct methods solve discretized OCPs with nonlinear programming solvers, either general-purpose and off-the-shelf solvers such as IPOPT [5] or SNOPT [6], or a more tailored solution. In robotics, OCPs are often transcribed using *shooting methods*, which discretizes the system dynamics by numerical integration. For these shooting methods, most structure-exploiting approaches leverage Riccati recursion [7]–[10], such as differential dynamic programming (DDP) [11]. DDP is one of the earliest such methods and a reference in nonlinear trajectory optimization [12] and has several variants, such as the iterative linear-quadratic iterator (iLQR) [13], [14]. DDP methods are a popular choice in robotics as they are relatively easy to implement with performance in mind [14]. They have enabled complex applications such as whole-body nonlinear MPC in robotics [15], [16].

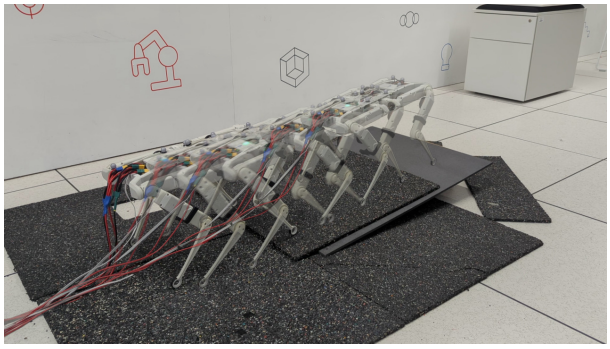


Fig. 1. **Solo-12 walking on a slope** using whole-body MPC based on the ALIGATOR solver. The slope is not accounted for in the MPC formulation.

However, standard Riccati recursion and (by extension) DDP are meant for unconstrained problems. A classical workaround is to add soft penalty terms to the OCP objective [15], [17]. Yet, penalty methods are known to suffer from numerical conditioning issues when tight constraint satisfaction is required [18]. They require tuning the penalty weights to avoid these conditioning issues.

Recently, several efforts have been made in the direction of handling hard constraints directly in the Riccati recursion. A first line of work has looked at the equality-constrained case [19]–[21] using rank-revealing decompositions (e.g., LU, QR). However, these approaches do not directly handle inequality constraints and must be combined with other methods (e.g., sequential quadratic programming (SQP) or interior-point). Furthermore, the backward pass of the Riccati recursion requires additional assumptions on the problem (e.g., positive-

*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.

definite reduced Hessians at every stage), which generic sparse linear solvers such as CHOLMOD [22] do not need (beyond the weaker requirement that the full linear system be invertible). To handle inequality constraints, projection approaches have been proposed to account for control bounds [23], [24] using a quadratic program (QP) in a modified Riccati backward pass. Other approaches, such as interior-point methods, have been employed in HPIPM [25], [26] and by Pavlov [27]. Although interior-point methods can obtain good precision in nonlinear programming settings [5], the approach is not always suitable for MPC applications due to limited warm-starting capabilities [28]. Closely related to these are barrier methods [18], "relaxed" versions of which have been used in trajectory optimization and MPC [29]–[31]. As penalty methods, they might thus suffer from conditioning issues (if the barriers are "tightened") and leave some margin of error, allowing infeasible trajectories to pass through.

In nonlinear constrained optimization, methods based on the augmented Lagrangian (AL) [32], [33] are another broad class of exact penalty methods with popular implementations such as LANCELOT [34]. One main difficulty with AL methods is they can lead to ill-conditioned linear systems for high penalty values, impacting these methods' convergence. For instance, [35], [36] have to use a second, projection-based stage after AL to refine the solutions to convergence.

Prior work in the NLP literature has proposed strategies for mitigating this issue. Conn et al. [37] introduced a heuristic to limit the increase of the penalty parameter, and reject some multiplier updates when they do not contribute to tighter constraint tolerance. In quadratic programming, Hermans et al. [38] and others [39], [40] have introduced primal-dual formulations for the (semi-smooth) Newton direction in AL-based methods, showing better numerical conditioning and solver behavior.

Contributions and paper plan

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 AL methods and solving subproblems with a primal-dual semi-smooth Newton method. The algorithm introduced in that section is implemented in a new library, PROXSUITE-NLP. 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 conference papers on constrained and implicit differentiable dynamic programming [41], [42]. Compared to our previous work, this article provides (i) a unified view of combining primal-dual augmented Lagrangian techniques within DDP-like solvers, (ii) open-source implementations of our algorithms as part of two new C++ libraries[1,2], as well as (iii) further additional experiments, including whole-body MPC on a quadruped robot, (iv) benchmarks against other nonlinear solvers.

II. BACKGROUND ON NONLINEAR OPTIMIZATION AND AUGMENTED LAGRANGIAN METHODS

This section provides a general overview of nonlinear optimization and augmented Lagrangian methods, which are at the core of the TO solver presented in this paper.

A. Nonlinear constrained optimization

Trajectory optimization problems may often be transcribed as NLPs. NLPs with equality and inequality constraints are optimization problems of the form:

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

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 Karush-Kuhn-Tucker (KKT) conditions [18], 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 j, $h_j(z) \leqslant 0$, $\nu_j \geqslant 0$ and either one or the other is zero. When $h_j(z) = 0$, $\nu_j > 0$ and the j^{th} constraint is said to be *active*.

The nonlinear programming problem can be seen as finding a saddle-point [18], [43] of the 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 based on augmented Lagrangian-type methods [32], [33]. We consider the Powell-Hestenes-Rockafellar (PHR) augmented Lagrangian penalty function, which can be defined as the value of the proximal point of the Lagrangian in its dual variables [44]:

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

where $\mu > 0$ is the AL penalty parameter, and (λ_e, ν_e) are the previous iterates of the proximal point method.

The maximizer of (4) in (λ, ν) is given by:

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

[1] ALIGATOR: https://github.com/Simple-Robotics/aligator/

[2] PROXSUITE-NLP: https://github.com/Simple-Robotics/proxsuite-nlp/

which are known as the *first-order multiplier estimates*. The corresponding maximum value of (4) gives the following expression for the AL function:

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

$\mathcal{L}_\mu$ is continuously differentiable, but not twice differentiable – but it admits semi-smooth *generalized* Hessians [45].

The augmented Lagrangian method. The prototypical augmented Lagrangian method, also known as *the method of multipliers* [32], consists of minimizing the augmented Lagrangian (6), updating the dual variables following (5), and repeating until convergence. This process is outlined in the prototype Algorithm 1.

Decreasing the penalty parameter μ_k can accelerate convergence [18]. Global convergence of the method in the convex case (under feasibility of the dual problem) was proven by Rockafellar [44, 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 ongoing [46].

A minimizer of step 2 of Algorithm 1, must be a *critical point* of $\mathcal{L}_\mu$ with respect to the primal variable z. Such a critical point z^* is a zero of the gradient:

$$\nabla_z \mathcal{L}_\mu(z;\lambda_e,\nu_e) = \nabla_z \mathcal{L}(z,\lambda^+(z),\nu^+(z)) = 0. \tag{7}$$

Together, a critical point z^* and the corresponding multiplier estimates $(\lambda^+(z^*),\nu^+(z^*))$ satisfy the following set of equations in (z,λ,ν):

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

where $\mathcal{T}_\mu$ is the proximal-KKT operator [38], [39], [47]. These are *necessary conditions* for z to minimize (6).

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 $\nu_{k+1} = \left[\nu_k + \frac{1}{\mu_k} h(z_{k+1})\right]_+$;

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. In this case, 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 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 together, this implies that (z^*,λ^*,ν^*) is a KKT point.

The augmented Lagrangian method is an exact penalty method [18, chap. 17], and can also be interpreted as a sequence of shifted penalty methods. 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 two questions: (i) how accurate this minimizer needs to be, and (ii) how a suitable inexact minimization approach would work (iii) how one could update μ_k in the outer loop for better convergence, adapted to how the problem feasibility progresses. All these questions are addressed in the next subsection.

C. Inner loop: 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 generic NLPs [33], [37], [48]. In particular, the authors of [37] have introduced *bound-constrained Lagrangian (BCL)* for controlling both the inexactness in the AL minimization and the penalty schedule, with convergence guarantees.

For minimizing the AL function, both first-order methods [49], and second-order methods (such as [34] in the equality-constrained case, using slacks for inequalities) have been investigated. Recently, a bigger focus has been put on second-order methods for the inequality-constrained case without using slack variables [50]. For instance, the QP solvers QPALM [38], PROXQP [39], and QPDO [40] rely on semi-smooth Newton methods applied to the augmented Lagrangian function, which is the approach we will be using here.

Linear system. The semi-smooth Newton method applied to minimizing (6) 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 the Hessian of the augmented Lagrangian function (6) 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 the Lagrangian Hessian[3] $\nabla_z^2\mathcal{L}(z,\lambda_k^+(z),\nu_k^+(z))$, $A = \partial g/\partial z, B = \partial h/\partial z$ are the constraint Jacobian matrices and

[3]Or an approximation thereof, for instance one which does not use the constraint Hessians such as the *Gauss-Newton* approximation.

$B_{\mathcal{I}}$ 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

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

Although this is a straightforward approach to obtaining descent directions for the AL function, the linear system (10) often suffers from poor conditioning. Recent work [38], [51] suggest rewriting it as a primal-dual system:

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

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. Given any (λ, ν), equation (13) can be rewritten as an equivalent primal-dual system [39]

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

This system also provides a search direction for finding a zero of (8).

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 direction. As mentioned above, system (14) suggests also including search directions for (8) in (λ, ν), as done by [40], [52]. In each subproblem, the dual variables are initialized as

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

Then, these variables are updated in SQP fashion using the directions from (14)

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

where α is (for simplicity) the same step size over the dual variables as the one over primal variables.

Linesearch and the merit function. To enforce global convergence, we adopt a linesearch procedure [18, chap. 3] using a merit function. Depending on the chosen scheme, one may use the augmented Lagrangian $\mathcal{L}_{\mu_k}$ for linesearch over the primal variable z. An alternative solution is the primal-dual augmented Lagrangian introduced by Gill and Robinson [52] and recently extended to the inequality-constrained case by De Marchi [40] (for QPs):

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

which allows search in both primal and dual variables (z, λ, ν).

Whichever the merit function $\Phi_k(w)$ and set of search variables $w = z$ or $w = (z, \lambda, \nu)$, we can use backtracking linesearch to produce step-length candidates and check for the Armijo condition $\Phi_k(w_{\text{cand}}) \leqslant \Phi_k(w) + c_1 \alpha \nabla \Phi_k(w)^\top \delta w$. (This is the approach from our prior papers [41], [42]). Another option is to use a nonmonotone linesearch scheme, which generally produces larger step-lengths by modifying the Armijo condition–a new contribution to this paper is that we have implemented a nonmonotone moving average linesearch [53]. Algorithm 2 provides an overview of this nonmonotone linesearch strategy. When the weight parameter $\gamma = 0$, this becomes equivalent to monotone backtracking linesearch.

Algorithm 2: Nonmonotone backtracking linesearch.

Input: Merit function Φ, current point w^j, search direction d^j, previous moving average $\bar{\Phi}^j$, merit gradient $\nabla \Phi(w^j)$, current weight Q^j, initial step length α_0, moving average parameter $\gamma \in [0, 1]$, $c_1 \in (0, 1)$

```
if j = 0 then
    Q^0 ← 0, Φ̄^0 ← 0;                                   // Initialize
h ← 0;
repeat
    α ← α_0 β^h;
    w_cand ← w^j + α d^j;
    h ← h + 1;
until Φ(w_cand) ⩽ Φ̄^j + α c_1 ∇Φ(w^j)^⊤ d^j;
w^{j+1} ← w_cand;
Q^{j+1} ← γ Q^j + 1;
Φ̂^{j+1} ← (γ Q^j Φ̄^j + Φ(w^{j+1}))/Q^{j+1};
```

Output: w^{j+1}, $\bar{\Phi}^{j+1}$, Q^{j+1}

Inner-loop stopping criterion. When using the primal-dual step, an appropriate metric on the primal-dual convergence is given by the magnitude of the proximal-KKT operator (8):

$$r_k(z, \lambda, \nu) \stackrel{\text{def}}{=} \left\| \begin{bmatrix} \nabla_z \mathcal{L}(z, \lambda, \nu) \\ \mu_k(\lambda_k^+(z) - \lambda) \\ \mu_k(\nu_k^+(z) - \nu) \end{bmatrix} \right\|_\infty. \tag{18}$$

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 an approximate stationarity point of subproblem (9) is found, 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 either branch of the algorithm is taken, the subproblem tolerances (ω_k, η_k) are updated following a geometric rule. This is the BCL penalty-scheduling heuristic from [37] and deployed in LANCELOT [34]. While LANCELOT [34] uses slack variables and a non-negativity constraint $z \geqslant 0$, in this article, we apply BCL to inequality-constrained problems, akin to [39] for QPs.

We use as constraint violation measures $\|g(z^{k+1})\|_\infty$ for equality constraints and $\|h(z^{k+1}) - [h(z^{k+1}) + \mu_k \nu_k]_-\|_\infty$ for inequalities (accounting for 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 3. We also provide an efficient C++ implementation of this primal-dual proximal NLP solver, called PROXNLP[4] and detailed in Section VI. For default parameters, we set $\mu_\mathrm{f} = 10^{-2}$, $\alpha_\omega = \beta_\omega = 1$, $\alpha_\eta = 0.1$, $\beta_\eta = 0.9$, nonmonotone linesearch parameters (see Algorithm 2) $\gamma = 0.85$ and $c_1 = 10^{-4}$.

Algorithm 3: AL method with inexact subproblem minimization.

Data: Initial guesses (z_0, λ_0, ν_0), parameters $(\alpha_\omega, \alpha_\eta, \beta_\omega, \beta_\eta)$, $\mu_\mathrm{f} \in (0,1)$, μ_0, 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.* (18);
- 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**
  - $\lambda_{k+1} = \lambda_k$, $\nu_{k+1} = \nu_k$;
  - *// update penalty*
  - $\mu_{k+1} = \mu_\mathrm{f} \mu_k$;
  - *// update tolerances following BCL (failure)*
  - $\omega_{k+1} = \omega_k \mu_{k+1}^{\beta_\omega}$;
  - $\eta_{k+1} = \eta_k \mu_{k+1}^{\beta_\eta}$;
- $\omega_{k+1} = \max(\omega_{k+1}, 10^{-2}\epsilon_\mathrm{tol})$

III. BACKGROUND ON TRAJECTORY OPTIMIZATION AND DIFFERENTIAL DYNAMIC PROGRAMMING

In this section, we introduce notations related to trajectory optimization and review the standard DDP approach that we later extend in Sec. IV to handle path constraints through a primal-dual AL method.

Notation. From this section onwards, 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 $(z_k, \dots, z_n)$.

A. Problem statement

Discrete-time optimal control. This class of problem arises when using *direct methods* [4] to solve continuous-time OCPs. These methods *transcribe* a continuous-time problem into a discrete-time OCP by discretizing states and controls:

$$\min_{\boldsymbol{x},\boldsymbol{u}} J(\boldsymbol{x},\boldsymbol{u}) \overset{\text{def}}{=} \sum_{t=0}^{N-1} \ell_t(x_t, u_t) + \ell_N(x_N) \tag{19a}$$

$$\text{s.t. } f_t(x_t, u_t, x_{t+1}) = 0, \tag{19b}$$

$$x_0 = \hat{x}, \tag{19c}$$

$$h_t(x_t, u_t) \leqslant 0, \tag{19d}$$

$$h_N(x_N) \leqslant 0, \tag{19e}$$

where t is the discrete time, f_t is the discrete dynamics (written in implicit form in (19)), $\hat{x}$ is some initial condition and the h_t are constraint functions. Here, we use the most generic discretization of the dynamics as an implicit integration scheme [41], which also covers explicit formulations that are more common in the DDP literature. This formulation encompasses equality and inequality path constraints, although many solvers treat these separately in general.

The Lagrangian associated with problem (19) reads:

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\lambda},\boldsymbol{\nu}) = J(\boldsymbol{x},\boldsymbol{u}) + \lambda_0^\top(x_0 - \hat{x}) + \sum_{t=0}^{N-1} \left(\lambda_{t+1}^\top f_t(x_t,u_t,x_{t+1}) + \nu_t^\top h_t(x_t,u_t)\right) + \nu_N^\top h_N(x_N), \tag{20}$$

where $\boldsymbol{\lambda} = (\lambda_t)_{t=0,\dots,N}$ are the co-states (the multipliers for dynamics and 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 (19e)).

B. KKT optimality conditions

The necessary optimality conditions given by the so-called Lagrangian stationarity conditions [18] read:

$$\begin{bmatrix} \nabla_{\boldsymbol{x}}\mathcal{L} \\ \nabla_{\boldsymbol{u}}\mathcal{L} \end{bmatrix} = 0. \tag{21}$$

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

$$f_{t-1,y}^\top \lambda_t + \ell_{t,x} + f_{t,x}^\top \lambda_{t+1} + h_{t,x}^\top \nu_t = 0, \tag{22a}$$

$$\ell_{t,u} + f_{t,u}^\top \lambda_{t+1} + h_{t,u}^\top \nu_t = 0, \tag{22b}$$

$$\ell_x + f_{0,x}^\top \lambda_1 + h_{0,t}^\top \nu_0 = 0 \tag{22c}$$

$$f_{t,y}^\top \lambda_N + \ell_{N,x} + h_{x,N}^\top \nu_N = 0, \tag{22d}$$

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. The feasibility and complementarity conditions read:

$$f_t(x_t,u_t,x_{t+1}) = 0 \text{ and } 0 \leqslant \nu_t \perp -h_t(x_t,u_t) \geqslant 0. \tag{23}$$

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

C. Differential dynamic programming

Differential dynamic programming (DDP) [11], and its variants such as iLQR [13], [14] are a class of Newton or Gauss-Newton methods which rely on Bellman's principle of optimality to compute descent directions and efficiently exploit the sparsity induced by time.

In the classical setting for DDP, we consider unconstrained instances of (19) of the form:

$$\begin{aligned} V_0(x_0) \overset{\text{def}}{=} \min_{\boldsymbol{x},\boldsymbol{u}} J(\boldsymbol{x},\boldsymbol{u}), \\ \text{s.t. } x_{t+1} = f_t(x_t,u_t),\ 0 \leqslant t < N, \end{aligned} \tag{24}$$

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

$$V_t(x) \overset{\text{def}}{=} \min_{u_t,\dots,u_{N-1}} \sum_{i=t}^{N-1} \ell_i(x_i,u_i) + \ell_N(x_N), \tag{25}$$

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:

$$V_t(x) = \min_u Q_t(x,u),\ Q_t(x,u) \overset{\text{def}}{=} \ell_t(x,u) + V_{t+1}(f_t(x,u)). \tag{26}$$

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 except for specific cases, such as linear-quadratic problems. However, a local quadratic expansion of its minimum around x can be obtained given expansions of ℓ and V_{t+1}.

By fully exploiting the time-induced sparsity of (19) in the backward and forward passes, the DDP algorithm is quite efficient with linear complexity with respect to the horizon length. From a software perspective, it is also relatively easy to implement. These two points make it a popular choice for implementing MPC on complex robot systems [15], [54], [55]. It has proven strong convergence properties [7] and is available in modern software libraries such as CROCODDYL [17] or OCS2 [56]. However, the nominal algorithm lacks support for hard path constraints. Another drawback is the fact that every iterate $(\boldsymbol{x},\boldsymbol{u})$ is forced to be feasible. Next, we propose combining the ideas of the proximal NLP algorithm of Sec. II with the dynamic programming principle to produce an effective and numerically robust algorithm for constrained trajectory optimization that achieves real-time MPC on complex robots.

IV. Proximal Differential Dynamic Programming

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

A. Proximal reformulation of the discrete-time OCP

Following Sec. II, our approach is to solve (19) with an AL scheme, by iteratively minimizing:

$$\begin{aligned} \mathcal{L}_{\mu_k}(\boldsymbol{x},\boldsymbol{u},\boldsymbol{\lambda}_k,\boldsymbol{\nu}_k) = J(\boldsymbol{x},\boldsymbol{u}) + \tfrac{1}{2\mu_k}\|x_0 - \hat{x} + \mu_k\lambda_0\|^2 \\ + \sum_{t=0}^{N-1} \Big(\tfrac{1}{2\mu_k}\|f_t(x_t,u_t,x_{t+1}) + \mu_k\lambda_{k,t+1}\|^2 \\ + \tfrac{1}{2\mu_k}\|[h_t(x_t,u_t) + \mu_k\nu_{k,t}]_+\|^2 \Big) \\ + \tfrac{1}{2\mu_k}\|[h_N(x_N) + \mu_k\nu_{k,N}]_+\|^2. \end{aligned} \tag{27}$$

We now detail how the equations of Sec. II can be implemented using dynamic programming, following a backward-pass/forward-pass strategy like DDP. The first-order conditions can be restated by introducing the Q-function

$$\begin{aligned} Q_t(x,u,y,\lambda,\nu) = \ell(x,u) + \lambda^\top f_t(x,u,y) \\ + \nu^\top h_t(x,u) + V_{t+1}(y), \end{aligned} \tag{28}$$

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

$$Q_x = \ell_x + f_x^\top\lambda + h_x^\top\nu, \tag{29a}$$
$$Q_u = \ell_u + f_u^\top\lambda + h_u^\top\lambda, \tag{29b}$$
$$Q_y = V'_x + f_y^\top\lambda. \tag{29c}$$

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

$$Q_{xx} = \ell_{xx} + \lambda\cdot f_{xx} + \nu\cdot h_{xx}, \tag{29d}$$
$$Q_{xu} = Q_{ux}^\top = \ell_{xu} + \lambda\cdot f_{xu} + \nu\cdot h_{xu}, \tag{29e}$$
$$Q_{uu} = \ell_{uu} + \lambda\cdot f_{uu} + \nu\cdot h_{uu}, \tag{29f}$$
$$Q_{xy} = Q_{yx}^\top = \lambda\cdot f_{xy} + \nu\cdot h_{xy}, \tag{29g}$$
$$Q_{uy} = Q_{yu}^\top = \lambda\cdot f_{uy} + \nu\cdot h_{uy}, \tag{29h}$$
$$Q_{yy} = V'_{xx} + \lambda\cdot f_{yy}. \tag{29i}$$

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

$$\mathcal{T}_{t,k}(u,y,\lambda,\nu;x) = \begin{bmatrix} Q_{t,u} \\ Q_{t,y} \\ f_t + \mu_k(\lambda_k - \lambda) \\ [h_t + \mu_k\nu_k]_+ - \mu_k\nu \end{bmatrix} = 0, \tag{30}$$

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):

$$\begin{aligned} \lambda_{k,t+1}^+ = \lambda_{k,t+1} + \tfrac{1}{\mu_k} f_t(x_t,u_t,x_{t+1}), \\ \nu_{k,t}^+ = \Big[\nu_{k,t} + \tfrac{1}{\mu_k} h_t(x_t,u_t)\Big]_+. \end{aligned} \tag{31}$$

B. Backward pass

Next, 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.

1) Terminal stage

In this proximal formulation, the terminal value function is

$$\begin{aligned} V_N(x) &= \max_{\nu \geqslant 0} \ \ell_N(x) + \nu^\top h_N(x) - \frac{\mu_k}{2}\|\nu - \nu_{k,N}\|^2 \\ &= \ell_N(x) + \frac{1}{2\mu_k}\left\|[h_N(x) + \mu_k \nu_{k,N}]_+\right\|^2 - \frac{\mu_k}{2}\|\nu_{k,N}\|^2. \end{aligned} \tag{32}$$

It is a continuously differentiable function in x. It is not twice-differentiable (due to nonsmooth terms) but has a *generalized* Hessian[45]. The quadratic model of this value function involves its gradient:

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

and choosing an element of its generalized Hessian [57],

$$\hat{V}_{xx,N} = \nabla^2 \ell_N + \frac{1}{\mu_k} h_{x,N}^\top P h_{x,N} + \frac{1}{\mu_k} \nu_{k,N}^+ \cdot h_{xx,N}. \tag{33}$$

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.

2) Middle stages

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 [58] to $\mathcal{T}_{t,k}(w)$. This leads to solving a linear system of equations

$$\mathcal{K}_\mu \begin{bmatrix} \delta u \\ \delta y \\ \delta\lambda \\ \delta\nu \end{bmatrix} = - \begin{bmatrix} Q_u + Q_{ux}\delta x \\ Q_y + Q_{yx}\delta x \\ \mu_k(\lambda_k^+ - \lambda) + f_x \delta x \\ \mu_k(\nu_k^+ - \nu_t) + (h_x)_{\mathcal{I}} \delta x \end{bmatrix}, \tag{34}$$

where $\mathcal{K}_\mu$ is the following matrix, an element of the generalized Jacobian $\partial \mathcal{T}_{t,k}(w)$ of (30)

$$\mathcal{K}_\mu = \begin{bmatrix} Q_{uu} & Q_{uy} & f_u^\top & h_u^\top \\ Q_{yu} & Q_{yy} & f_y^\top & \\ f_u & f_y & -\mu I & \\ (h_u)_{\mathcal{I}} & & & -\mu I \end{bmatrix}. \tag{35}$$

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

$$\mathcal{K}_\mu \begin{bmatrix} k & K \\ c & C \\ \xi & \Xi \\ \zeta & Z \end{bmatrix} = - \begin{bmatrix} Q_u & Q_{ux} \\ Q_y & Q_{yx} \\ \mu_k(\lambda_k^+ - \lambda) & f_x \\ \mu_k(\nu_k^+ - \nu_t) & (h_x)_{\mathcal{I}} \end{bmatrix}. \tag{36}$$

$\mathcal{K}_\mu$ is *not* a symmetric matrix, and thus cannot be factorized using symmetric matrix decompositions (e.g. indefinite Cholesky). However, (36) can be transformed into a symmetric system by exploiting the fact that the last line implies that we can solve for the inactive multipliers: $(\delta\nu)_{\overline{\mathcal{I}}} = -\nu_{\overline{\mathcal{I}}}$ and $h_u = (h_u)_{\mathcal{I}} + (h_u)_{\overline{\mathcal{I}}}$. Now, by defining the *symmetric* matrix

$$\underline{\mathcal{K}}_\mu \overset{\text{def}}{=} \begin{bmatrix} Q_{uu} & Q_{uy} & f_u^\top & (h_u)_{\mathcal{I}}^\top \\ Q_{yu} & Q_{yy} & f_y^\top & \\ f_u & f_y & -\mu I & \\ (h_u)_{\mathcal{I}} & & & -\mu I \end{bmatrix}, \tag{37}$$

the system (36) above is now equivalent to

$$\underline{\mathcal{K}}_\mu \begin{bmatrix} k & K \\ c & C \\ \xi & \Xi \\ \zeta & Z \end{bmatrix} = - \begin{bmatrix} Q_u + (h_u)_{\overline{\mathcal{I}}}^\top \nu & Q_{ux} \\ Q_y & Q_{yx} \\ \mu_k(\lambda_k^+ - \lambda) & f_x \\ \mu_k(\nu_k^+ - \nu) & (h_x)_{\mathcal{I}} \end{bmatrix} \tag{38}$$

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 (38) should *not* be used as a stopping criterion for the subproblem. It is merely a proxy to get a symmetric linear system. Here, the proximal regularization in the dual blocks of (37) guarantees it is always well-conditioned, enhancing the method's convergence (see [18], chap. 17).

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

$$V_x = Q_x + Q_u^\top K + Q_y^\top C + f^\top \Xi + h^\top Z. \tag{39a}$$

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

$$V_{xx} = Q_{xx} + Q_{xu} K + Q_{xy} C + f_x^\top \Xi + (h_x)_{\mathcal{I}}^\top Z. \tag{39b}$$

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 $\delta w = (\delta u, \delta y, \delta\lambda, \delta\nu)$,

$$V_x + V_{xx}\delta x = Q_x + Q_{xx}\delta x + Q_{xw}\delta w, \tag{40}$$

and finally, replacing the expression of δw.

3) Initial stage

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

$$\min_{x_0} \max_{\lambda_0} V_0(x_0) + \lambda_0^\top (x_0 - \hat{x}) - \frac{\mu}{2}\|\lambda_0 - \lambda_{k,0}\|^2. \tag{41}$$

The corresponding Newton 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 B. 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 SQP step, and (ii) a nonlinear rollout, as classically done in DDP methods [11], [14].

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 (53), and for every $t = 0, \dots, N-1$,

$$\delta u_t = k_t + K_t \delta x_t \tag{42a}$$
$$\delta x_{t+1} = c_t + C_t \delta x_t \tag{42b}$$
$$\delta \lambda_{t+1} = \xi_t + \Xi_t \delta x_t \tag{42c}$$

This linear rollout was used in the first iLQR paper of Li and Todorov [13], however both the original DDP and the papers which follow Li and Todorov use a nonlinear rollout [14].

Nonlinear rollout. Liao and Shoemaker [59] 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, and was one of the enhancements to iLQR proposed by Tassa [14], 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:

$$x_0^\alpha \overset{\text{def}}{=} x_0 + \alpha \delta x_0, \tag{43a}$$
$$\lambda_0^\alpha \overset{\text{def}}{=} \lambda_0 + \alpha \delta \lambda_0, \tag{43b}$$
$$u_t^\alpha \overset{\text{def}}{=} u_t + \alpha k_t + K_t(x_t^\alpha - x_t), \tag{43c}$$
$$\lambda_{t+1}^\alpha \overset{\text{def}}{=} \lambda_{t+1} + \alpha \xi_{t+1} + \Xi_{t+1}(x_t^\alpha - x_t), \tag{43d}$$
$$x_{t+1}^\alpha \overset{\text{def}}{=} f^{\text{ex}}(x_t^\alpha, u_t^\alpha) + \mu_k(\lambda_{t+1}^\alpha - \lambda_{t+1}). \tag{43e}$$

Nonlinear rollout with implicit dynamics. In the implicit-dynamics case, we have to replace (43e) 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. [60]. 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

$$r_{\text{d}} = \left\| (\nabla_{\boldsymbol{x}} \mathcal{L}, \nabla_{\boldsymbol{u}} \mathcal{L}) \right\|_\infty, \tag{44}$$

corresponding to (22) and the primal residual (covering the complementarity conditions):

$$r_{\text{p}} = \left\| (f_t, [h_t + \mu_k \nu_{k,t}]_- - h_t)_t \right\|_\infty. \tag{45}$$

Furthermore, the globalization strategy we adopt is through linesearch, using either the augmented Lagrangian (27) or the primal-dual function (17).

E. The PROXDDP algorithm

The implementation of the PROXDDP method presented in this section corresponds to Algorithm 3, 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 in Tab. I.

TABLE I
EQUIVALENCE BETWEEN THE NLP METHOD OF SEC. II AND PROXDDP.

	PROXNLP	**PROXDDP**
Decision vars.	primal: z dual: (λ, ν)	primal: $(\boldsymbol{x}, \boldsymbol{u})$ dual: $(\boldsymbol{\lambda}, \boldsymbol{\nu})$
KKT operator	$\mathcal{T}_\mu(z, \lambda, \nu; \lambda_e, \nu_e)$ (8)	$\mathcal{T}_{t,k}$ + dyn. prog. (30)
Newton step	(14)	(38)
Update	(15) and (16)	(42) or (43)
Inner criterion	(18)	r_{p} (45) and r_{d} (44)

F. Practical details

Choice of rollouts. In our experiments, we used the nonlinear rollout formulation of the forward pass when explicit dynamics were considered (unless otherwise specified). Concurrent work in the controls literature [61]–[64] has looked at using a nonlinear dynamics-based projection of the Newton or SQP direction in trajectory-space. This literature suggests using the Riccati solution of the linearized system to define the projection operator. A link [62] has been suggested with the feasible SQP method introduced in [65] for MPC applications.

Multiple shooting. Unlike what the definition of problem (19) suggests, where the states $(x_t)_t$ are explicit decision variables, DDP method 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 use infeasible initial guesses for the x_t, which is troublesome for settings such as locomotion problems. They also require solving for the dynamics in the forward pass, which might be costly or numerically harmful in the case of implicit integrators stiff dynamics (e.g. DAEs). In contrast, multiple-shooting approaches create a more tractable optimization landscape by adding more variables and allowing infeasibility in the dynamics at some problem stages [4]. A combination of linear, SQP-type, and nonlinear, DDP-type rollouts also leads to a variant of multiple-shooting [10], as can the addition of slack variables [51] for the dynamical constraints. In our method, the slack variable is linked to the gap in convergence in the dual (co-state) variables, see Appendix A.

Further extensions. In Appendix B, we extend the algorithm to handle more general initial conditions $g_0(x_0) = 0$.

Furthermore, we implement a heuristic for the AL parameter μ just for the dynamical constraints, scaling it by 10^{-3}

to promote faster convergence of the AL method towards dynamically feasible trajectories – this is similar to the heuristic in PROXQP ([39], Sec. V) which scales μ by 10^{-2} for equality constraints. This same scaling is kept throughout the experiments and benchmarks.

V. RELATED WORK

In this section, we make a brief review of the different features and methods implemented in alternative solutions present in the state of the art, which are summarized in Tab. II.

The literature often treats constraints differently depending on their type, as inequality constraints are generally seen as more complex to handle due to the associated complementarity conditions. For equality constraints, recent papers [20], [21] solve equality-constrained LQR subproblems by using a nullspace method – the equivalent to the equality-constrained QP subproblem in SQP methods [18, chap. 18].

Some recent other works advocate for using augmented Lagrangian approaches for equality constraints combined with DDP-like recursion, which has led to the development of several methods in recent years [35], [36], [41], [51]. AL approaches for both equality and inequality constraints have been developed [35], [36], [70], [71], including the authors' prior work [42] of which this paper is an extension. In contrast to [35], [36], [70], our approach uses a primal-dual Newton step (38) 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 [51]) due to better numerical conditioning of the primal-dual system [39].

For inequality constraints, [25]–[27], [67], [72] employ interior-point methods, which have been a popular option in nonlinear programming through solvers such as IPOPT [5]. Interior-point methods solve a succession of barrier-penalized problems which are a relaxation of the KKT complementarity slackness. They approach the real problem over multiple iterations by tightening 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 [18] leveraged by [29]–[31], [73]. Among these, the relaxed barrier approach [30] 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).

VI. SOFTWARE IMPLEMENTATION DETAILS

Over the past few years, several open-source TO libraries have been developed in the robotics community: CONTROL TOOLBOX [10], OCS2 [56], ALTRO [35], CROCODDYL [17], FATROP [67]. We contribute to this open-source initiative by developing two new software libraries for nonlinear optimization and optimal control in robotics:

1) PROXSUITE-NLP, 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.

PROXSUITE-NLP and ALIGATOR are both written in C++ using the Eigen [74] template library for linear algebra, which is leveraged for efficient computation with minimal dynamic memory allocation at runtime, enabling high-performance scenarios. The ALIGATOR library includes the following features:

- an implementation of the PROXDDP algorithm described in this paper;
- support for problems with Lie group state spaces, extending the mathematical framework detailed in this paper;
- 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 [75];
- API frontends in both the C++ and Python programming languages: these frontends are built using the same model-data pattern as the PINOCCHIO and CROCODDYL libraries for caching of previous computations;
- multithreading using OpenMP for problem quantities and their derivatives in parallel;
- compatibility with the CROCODDYL trajectory optimization library. ALIGATOR comes with its own implementation of the FDDP algorithm, initially introduced in [17].

Both libraries are templated on the scalar type (by default, double precision floating point). In the future, we plan to add the following: integration with CasADi [76] and CppAD for automatic differentiation and code generation (which is be useful for prototyping); support for high-precision floating point types using MPFR; a release on package managers such as APT, rospkg, Homebrew, Conda[5], and pip.

VII. EXPERIMENTS

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

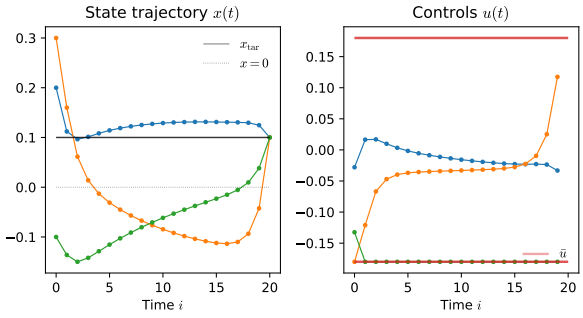
Constrained LQR. We first show that our solver can solve a linear-quadratic regulator (LQR) problem with a terminal constraint and control bounds $-\bar{u} \leqslant u \leqslant \bar{u}$ in a reasonable number of iterations. The optimized state-control trajectory and convergence of primal-dual residuals are shown in Fig. 2.

[5]Already available at https://anaconda.org/conda-forge/aligator.

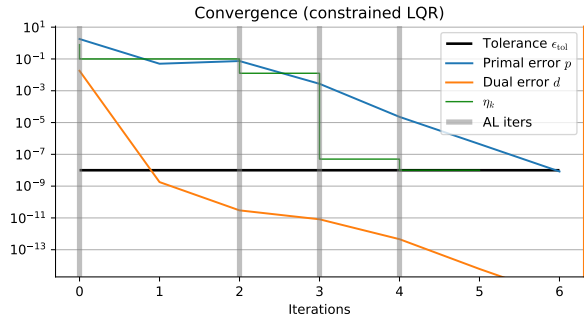
TABLE II
COMPARISON OF DIFFERENT METHODS AND SOFTWARE LIBRARIES FOR SOLVING OCPS.

Library	Solver	Dynamics handling	Forward pass	Constraints (*approach*)	Globalization
-	Stagewise Newton [7]	Single-shooting	Open-loop	✗	Linesearch
-	Li's iLQR [13]	Single-shooting	Open-loop	✗	Linesearch
CROCODDYL	Vanilla DDP/Tassa's iLQR [14]	Single-shooting	Closed-loop	✗	Linesearch
	FDDP [17]	Multiple-shooting[1]	Closed-loop	✗	Linesearch
	BOX-DDP [23]	Single-shooting	Closed-loop	Control limits (*projection*)	Linesearch
	INTRO[2] [66]	Multiple-shooting	Closed-loop	✓(*projection*)	Linesearch
ALTRO [35]		Multiple-shooting[3]	Closed-loop	✓(*A.L. + projection*)	Linesearch
CONTROL TOOLBOX	GNMS/iLQR-GNMS [10]	Multiple-shooting	Linear/Closed-loop[4]	✗	Linesearch[5]
FATROP [67]		Multiple-shooting	Linear	✓(*interior-point*[6])	Filter[6]
ALIGATOR	**PROXDDP (ours)**	Multiple-shooting	Linear/Closed-loop	✓(*A.L.*)	Linesearch
OCS2 [56]	Constrained SLQ [55]	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 [68], [69]	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 [5].



(a) Optimized state and control trajectory. The controls saturate some of their limits.



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

Fig. 2. **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.

Cartpole. The problem of swinging-up an inverted pendulum on top of a cartpole is a simple example of a very nonlinear non-convex problem. The discretization timestep is $\Delta t = 10\,\text{ms}$, and the time horizon is $T = 5\,\text{s}$. The constrained cartpole task is defined to have the terminal position of the end-effector $p_{\text{ee}}(x_N)$ be at $\overline{p} = (0, 1)^\top$ in the yz-plane, and torque limits $-\tau_{\max} \leqslant \tau \leqslant \tau_{\max}$. See Fig. 3 for the resulting state-control and end-effector trajectories and convergence of the iterates. The initial AL parameter is set to $\mu_0 = 10^{-8}$, promoting rapid convergence.

Acrobot. In the first setting, we impose a terminal constraint on the state. The optimized trajectories are shown in Fig. 4. The system state is $x = (e^{i\theta_1}, e^{i\theta_2}, \dot{\theta}_1, \dot{\theta}_2) \in \text{SO}(2)^2 \times \mathbb{R}^2$. The second setting comes with torque limits and relax the target reaching task into a quadratic cost: the resulting trajectories are shown in Fig. 5. The integrator is 2nd-order Runge-Kutta with time step $\Delta t = 5\,\text{ms}$. The initial AL parameter is the solver default $\mu_0 = 10^{-2}$.

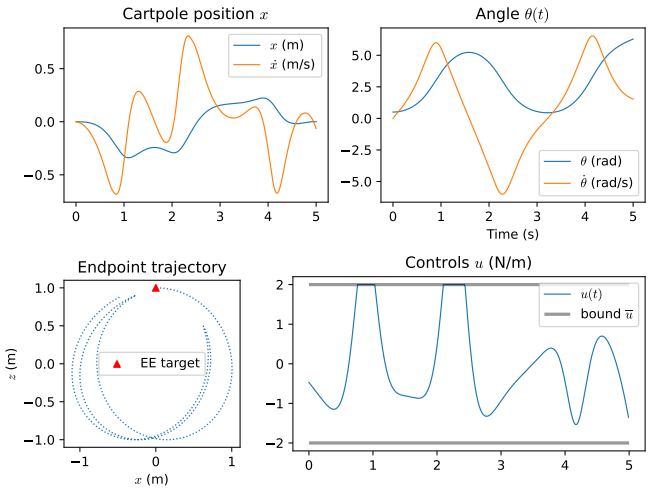
B. Ballistics with a manipulator

This second experiment demonstrates a 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, implemented as a constraint.

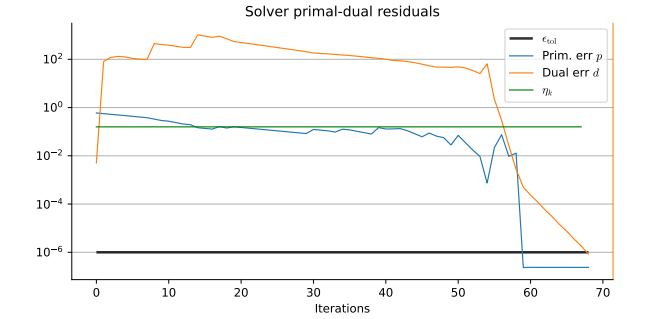
System model. The system is modelled using PINOCCHIO [2]. The chosen manipulator is a UR10 arm, and a projectile (a mug) is added to the robot's model tree with a free-flyer joint. The system's configuration vector $q = (q_{\text{ob}}, q_{\text{a}})$ consists of the joint configuration $q_{\text{a}} \in \mathbb{R}^{n_{q1}}$ and projectile pose $q_{\text{ob}} \in \text{SE}(3)$. We use a whole-body model for the system dynamics

$$M(q)\ddot{q} + b(q, \dot{q}) + J_c(q)^\top \lambda_c = B\tau \tag{46a}$$
$$c(q) = 0. \tag{46b}$$

with $M(q)$ the joint space inertia matrix, $\tau \in \mathbb{R}^6$ the input torque, c the 6D constraint between the end-effector and the projectile's frames, $J_c(q) \in \mathbb{R}^{6\times 12}$ its Jacobian, and $\lambda_c \in \mathbb{R}^6$ is



(a) Optimized state and control trajectories and trajectory of the cartpole's end-effector.



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

Fig. 3. **Cartpole with both torque limits and a terminal constraint.** $\tau_{\max} = 2\,\mathrm{N.m}^{-1}$. Desired tolerance $\epsilon_{\text{tol}} = 10^{-6}$.

the constraint force between the end-effector and the projectile. These are underactuated dynamics (with selection matrix B) since the projectile is not actuated. Since (46b) is a position-level constraint while the dynamics are solved for acceleration $\ddot{q}$, it is replaced with an acceleration-level constraint, where (K_p, K_d) are Baumgarte stabilization gains [77]:

$$\ddot{c} = J(q)\ddot{q} + \gamma(q, \dot{q}) = -K_p c(q) - K_d \dot{c}. \tag{47}$$

The constrained dynamics model is implemented and solved using the constrained forward dynamics algorithm of [75] 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. For $t > t_c$, the constraint (46b) and contact force λ_c are removed, the robot and projectile are decoupled, and the projectile is in flight phase.

Control regularization and initial guess. The gravity-compensating torque for a manipulator without constraints is $\tau_0 = b_{\text{a}}(q_0, v_0)$, and can be computed using RNEA [1]. When a body is attached to the end-effector, the extra mass means applying the torque τ_0 will still create a nonzero acceleration. We can resolve this by solving the static equation in (λ_c, τ):

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

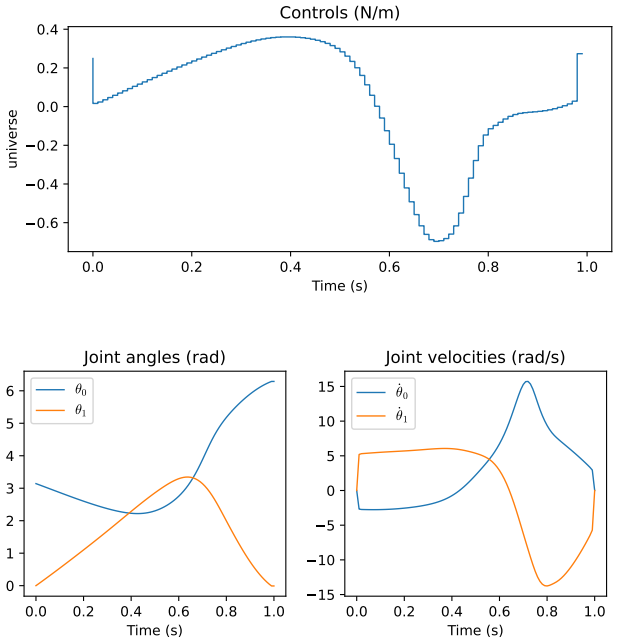


Fig. 4. **Acrobot with terminal constraint.** Optimized state and control trajectories with a terminal constraint $x(T) = ((1,0),(1,0),0,0)$ (i.e., $\theta_1 = \theta_2 = 0 \mod 2\pi$).

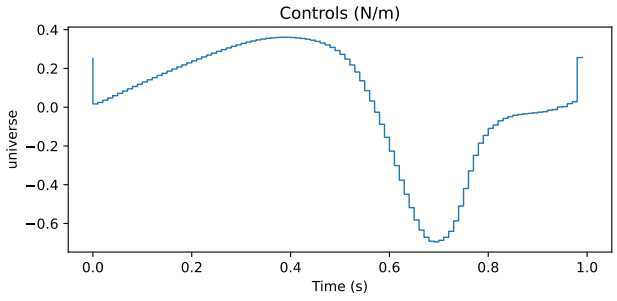


Fig. 5. **Acrobot with control limits.** Optimized control trajectory with control bounds.

where $J_c = [J_{c,\text{a}}\ J_{c,\text{ob}}]$. Once equipped with an appropriate τ_0, we use it to construct a quasistatic initial guess for the solver and 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. There are no target position or target launch velocity costs, only a hard constraint on the final position of the projectile. We expect the optimizer to find a feasible trajectory that satisfies this constraint. Additionally, we add limits on the manipulator input torque $-\tau_{\max} \leqslant u \leqslant \tau_{\max}$.

In this example, we used a UR10 manipulator; the projectile is a mug. The optimized control trajectory is shown in Fig. 7, and joint velocities in Fig. 8. The desired tolerance is $\epsilon_{\text{tol}} = 10^{-4}$, and we used an initial parameter $\mu_0 = 0.2$. The graphs show very high-bandwidth behavior in the controls, which is not reflective of what can be achieved on an actual system –

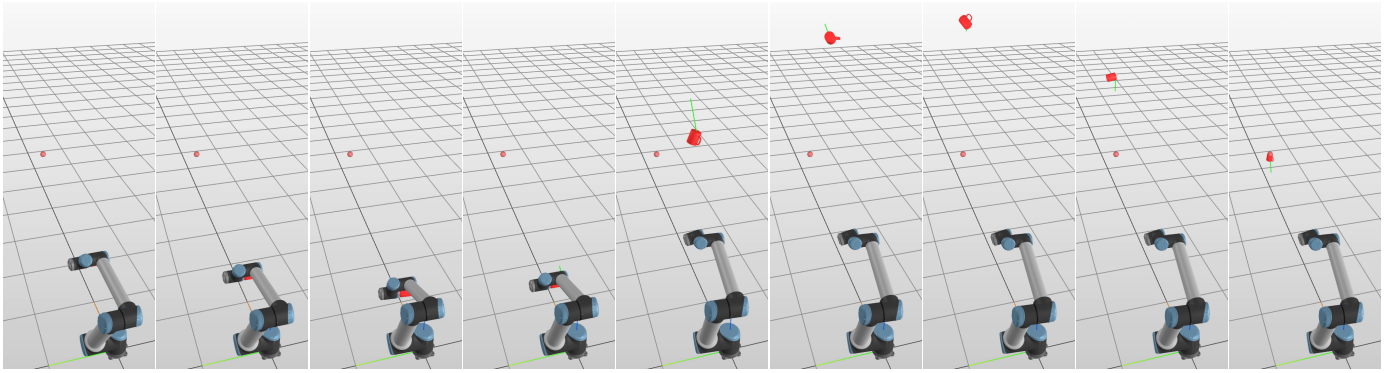


Fig. 6. Optimized motion for the UR10 ballistic task, rendered using the MeshCat visualizer.

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 6 as well as the supplementary video accompanying this submission.

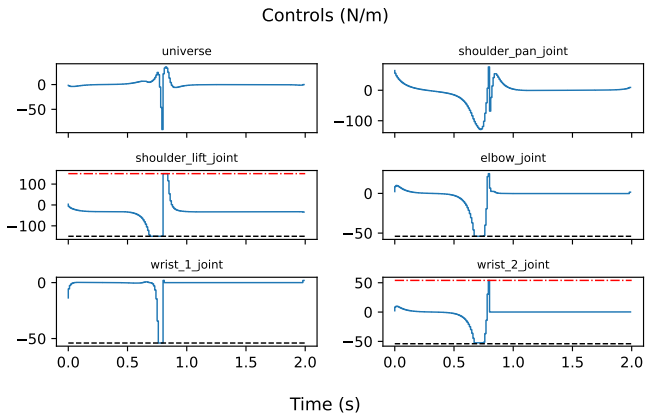


Fig. 7. **Optimized control trajectory for the UR10 ballistic task.** The red (resp. black) dashed line represents the upper (resp. lower) torque limit. These limits were imposed using a hard box constraint in the problem formulation.

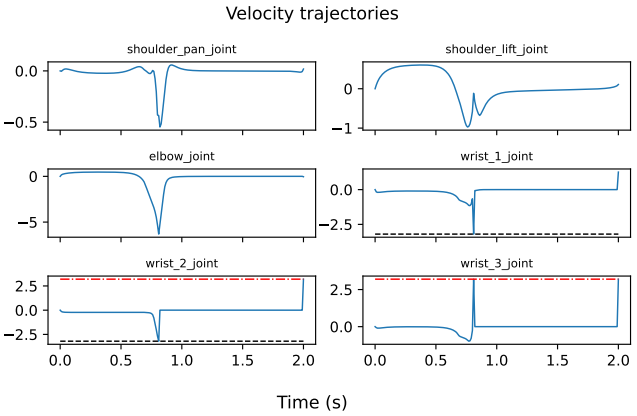


Fig. 8. **Computed joint velocities for the ballistic task on the UR10.** The red (resp. black) dashed line represents the velocity upper (resp. lower) limit. We impose joint velocity limits, except for the shoulder lift joint.

C. Motion planning on a quadrotor

We use the Hector quadrotor model[6]. The quadrotor's configuration is just the base 6D pose $q \in \mathrm{SE}(3)$, and we use standard inertial dynamics with rotor actuation. We build two example problems for the system: (i) thrust control limits and (ii) obstacle avoidance with thrust limits. In both settings, the desired tolerance is $\epsilon_{\text{tol}} = 10^{-4}$.

Thrust limits. We enforce thrust limits $-\tau_{\max} \leqslant \tau \leqslant \tau_{\max}$ to the quadrotor. The controls, drone 3D position, convergence regime in Fig. 9.

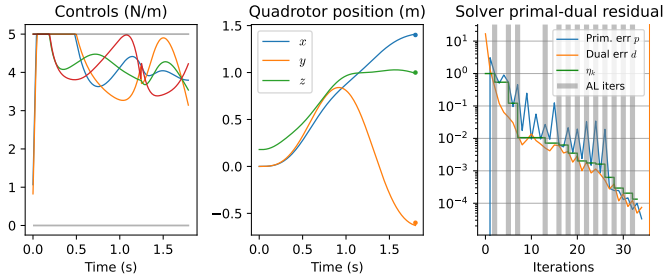


Fig. 9. **Optimized control trajectory and convergence of the quadrotor problem with thrust limits.** The thrust limits often saturate in this setting due to the tight rightward turn needed to move to the next target. Initial AL parameter $\mu_0 = 10$.

Obstacles. 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. We also impose thrust limits. We obtain the obstacle-avoiding trajectory as rendered in Fig. 12. The controls and convergence are shown in Fig. 10.

D. Whole-body jump on a quadruped

The system is also modeled using a whole-body model with bilateral contact constraints as in (46b) and (46): The contact constraint is a 3-dimensional positional constraint on the feet. It is a multiple contact phase problem: (i) all four feet in contact, (ii) no contact (flight phase) at $t = t_0$, and (iii) making contact once again (landing phase) at $t = t_1$.

[6]It can be found in the example-robot-data package: https://github.com/Gepetto/example-robot-data

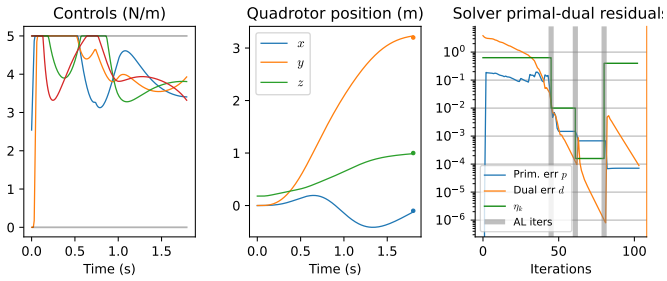


Fig. 10. **Optimized trajectory and convergence of the quadrotor problem with obstacles and thrust limits.** The corresponding rendered trajectory is shown in Fig. 12. Initial AL parameter $\mu_0 = 1$.

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

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

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 solver can account for.

The generated motion is illustrated in Fig. 13. The convergence of the problem residuals as the solver progresses is shown in Fig. 11. The target tolerance is $\epsilon_{\text{tol}} = 10^{-5}$, and the initial AL parameter is $\mu_0 = 10^{-5}$.

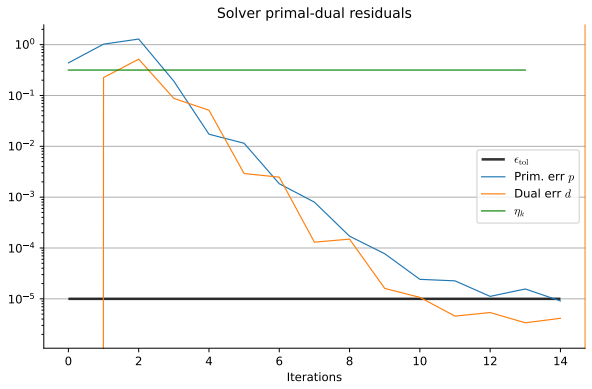


Fig. 11. **Convergence of the primal-dual residuals for the Solo-12 jumping task.** Target tolerance $\epsilon_{\text{tol}} = 10^{-5}$.

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-12 quadruped on uneven terrain, as illustrated on the screenshot sequences in Fig. 14 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. [16]. The desired robot base velocities are provided using a gamepad.

Hardware. The quadruped is connected over Ethernet to a laptop running Ubuntu 20.04 with real-time kernel patches, connected to a ROS [78] node on an Mac Studio M1 Ultra desktop running the MPC.

Control framework. We have adapted the framework developed by Assirelli et al. [16] to use our reference implementation of PROXDDP in ALIGATOR. It comprises (i) a state estimator with a complementary filter described in [79], which estimates the base position and linear velocity, (ii) a lower-level high-frequency whole-body controller using the first Riccati gain and feedforward control, which outputs a desired torque:

$$\overline{\tau}(t) \overset{\text{def}}{=} u_0 + K_0 \left(\hat{x}(t) \ominus \hat{x}_{\text{ocp}}(t) \right) \tag{48}$$

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

$$\begin{aligned} \ell(x,u) = \ell_{\text{fly}}(x) + \ell_{\text{reg}}(x,u) + \ell_g(x) \\ + \ell_{bv}(x) + \ell_{bu}(u) + \ell_{\text{base}}(x; \hat{v}_{\text{base}}) \end{aligned} \tag{49}$$

All the cost function terms are summarized in Tab. III. Index j spans over the quadruped's 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 (48) 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[7].

In Fig. 14 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.

VIII. BENCHMARKS

To assess the effectiveness of our method (and its implementation), we have compared it to other solvers on a set of benchmark problems which are nonlinear, difficult but realistic TO problems for robotics. We benchmark the following solvers: PROXDDP (linear and nonlinear rollouts), IPOPT [5], and the C++ version of ALTRO which we have forked from the original repository[8]. The benchmarking code can be found online[9]. All benchmarks were run on a laptop with an Intel Core i7-9750H mobile CPU.

Implementation notes:

- To get a fairer comparison (same formulations and derivatives), we have implemented wrappers for IPOPT and ALTRO to solve OCPs written with ALIGATOR's API.
- The IPOPT wrapper uses a multiple-shooting formulation (both states and controls are decision variables).

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

[8]Original repo: https://github.com/bjack205/altro. Our fork at https://github.com/ManifoldFR/altro/.

[9]https://github.com/Simple-Robotics/aligator-bench

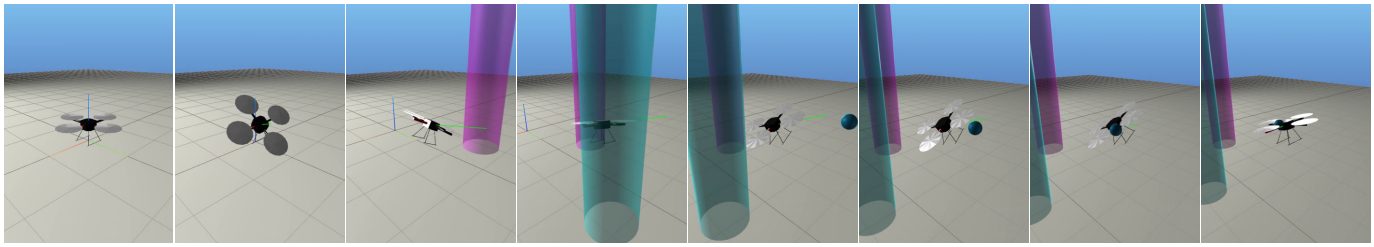


Fig. 12. Trajectory of the quadrotor avoiding obstacles, rendered using the MeshCat visualizer.

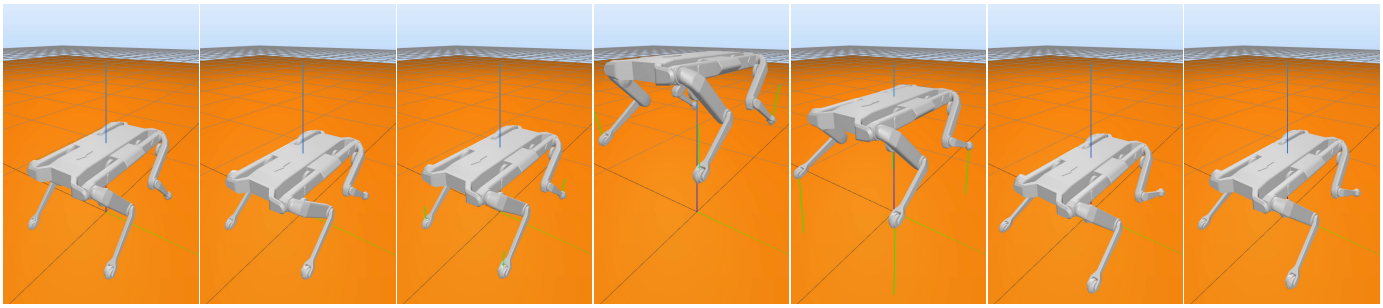


Fig. 13. 3D rendering of the Solo-12 whole-body jumping task. The time step is $\Delta t = 5\,\text{ms}$ and time horizon $T = 1.2\,\text{s}$, thus a discrete horizon of $N = 240$ time steps.

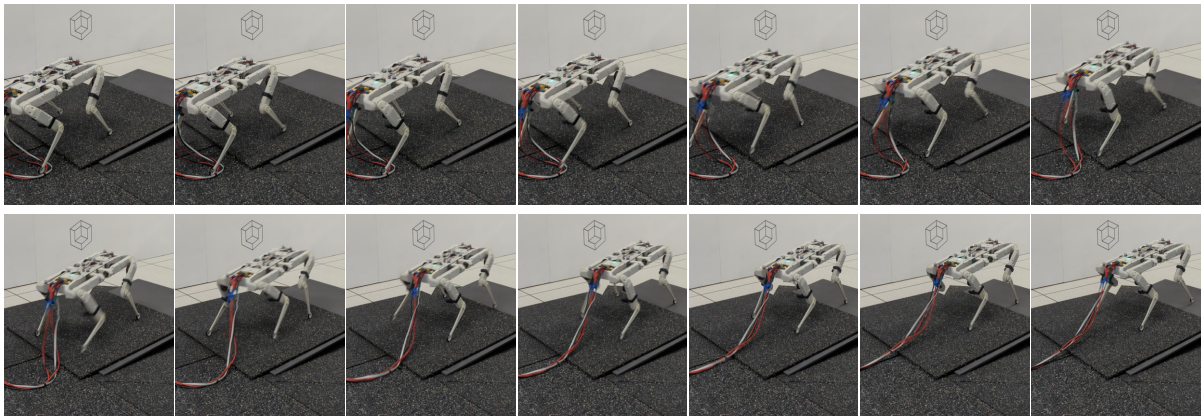


Fig. 14. Solo-12 walking task. The time step is $\Delta t = 12\,\text{ms}$ and time horizon $T = 0.480\,\text{s}$. The slope and bumps of the terrain are unmodeled and act as perturbations.

- The C++ version of ALTRO does not implement the square-root backward pass nor the solution polishing step from the original papers [35], [80].
- We have added complementary slackness (2c) to ALTRO's convergence criteria.
- Unlike PROXDDP, ALTRO solves AL subproblems (9) to a fixed tolerance which is a hyperparameter of the solver, and separate from the constraint tolerance ϵ_{tol}. For the other solvers, asking for a tolerance ϵ_{tol} pertains to both primal and dual feasibilities.
- We benchmark two configurations for ALTRO: (i) the tolerance for subproblem (9) is fixed to the 10^{-4} (ii) we set this tolerance to be the same as the desired problem tolerance ϵ_{tol}. Same for IPOPT: (i) using a Gauss-Newton Lagrangian Hessian provided by ALIGATOR, and (ii) using its implementation of L-BFGS. The configurations for both solvers are summarized in Tab. IV.

Performance metrics. To assess algorithm performance, we plot the proportion of problems solved after a given number of iterations, as well as the performance profiles. The performance profile is the proportion of problems solved as a function of how slower (in wall time) a solver is compared to the fastest one on a given problem. It is a normalized metric of solver wall time for proportion of problems solved (see [39], Sec. VI).

A. UR5 "Reaching" task

The problem is for the end-effector of an UR5 robot arm to reach a given target $p_{\text{ee}} \in \mathbb{R}^3$ by the end of a horizon $[0, t_{\text{f}}]$ discretized with a step $\Delta t = 10\,\text{ms}$. The target is implemented as a *hard terminal constraint*.

We generate random instances of the problem by sampling random initial configurations $q_0 \sim \mathcal{N}(0, 1)$ and random targets p_{ee} in a hemispherical shell around the origin. We generated $M = 80$ instances of the problem, which are solved using IPOPT, ALTRO, and four different configurations of PROXDDP. We allow for up to 400 iterations, targeting a final tolerance of

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 $\ell_{\text{imp.}}$	$\sum_j \|z^j - z^{\text{ref}}\|_{w_h} + \|v^j\|_{w_i}$	$w_h = w_i = 10^4$

TABLE IV
IPOPT AND ALTRO CONFIGURATIONS FOR THE BENCHMARKS

Ipopt config.	**Hessian approx.**
0	Gauss-Newton (provided by ALIGATOR)
1	Limited-memory BFGS
Altro config.	**Subproblem (9) tol.**
0	1e-4 (default)
1	ϵ_{tol} (desired problem tol.)

$\epsilon_{\text{tol}} = 10^{-5}$. All solvers are cold-started. The parameters for the PROXDDP configurations are in the following table.

Config.	μ_0	Linesearch γ	Rollout
0	1.0	0.2	Nonlinear
1	0.1	-	Linear
2	1.0	0.85	Nonlinear
3	0.1	-	Linear

We tested two variants of this benchmark. First, without joint velocity limits, and a time horizon $t_{\text{f}} = 1.0\,\text{s}$. Second, with joint velocity limits, implemented as hard constraints, and a slightly longer horizon $t_{\text{f}} = 1.6\,\text{s}$ – some instances of the problem might be infeasible.

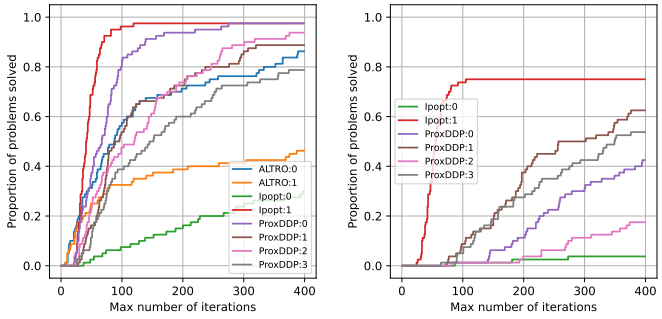
The benchmark metrics – proportion of problems solved by number of iterations, and performance profile – are shown respectively in Fig. 15 and Fig. 16.

On the variant without joint velocity limits, IPOPT with L-BFGS does better than all others in terms of number of iterations, but the performance profile shows PROXDDP is overall faster.

The variant with joint velocity limits is more difficult, with even the best solver (IPOPT L-BFGS) not being able to solve every instance, whereas ALTRO fails on every instance. For this variant, PROXDDP is clearly beat by IPOPT L-BFGS.

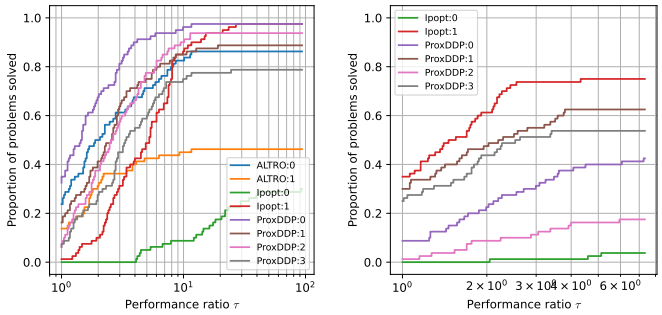
B. Solo-12 "Yoga"

This is a multiple contact-phase problem on the Solo-12 quadruped. The phases are as follow: (i) all feet are on the ground, the quadruped dips forward (spinning its body forward and towards the left or right) before returning to its initial pose, (ii) it lifts its front left foot, and sends it backwards



(a) Without joint velocity limits. (b) With joint velocity limits.

Fig. 15. **UR5 reaching benchmark.** Proportion of problems solved as a function of the number of solver iterations. For PROXDDP, the suffix $:n$ corresponds to the solver configurations. Target tolerance $\epsilon_{\text{tol}} = 10^{-5}$.



(a) Without joint velocity limits. (b) With joint velocity limits.

Fig. 16. **UR5 reaching benchmark.** Performance profiles for the two variants of the problem.

(implemented as a hard constraint $x_{\text{fl foot}} \leqslant 0.1$ on the end-effector), (iii) it lifts its hind left foot and balances solely on its front and hind right feet (iv) restores contact with the hind left foot.

The PINOCCHIO model has been modified to replace the non-Euclidean pose (using the $\text{SE}(3)$ Lie group) with a Euclidean representation (3D position and Euler angles for orientation) that all solvers can work with.

The solvers run were IPOPT and 6 different configurations

TABLE V
CONFIGURATIONS OF PROXDDP FOR THE UR10 BENCHMARK.

Config.	μ_0	Linesearch γ	Rollout
0	1.0	0.0 (monotone)	Nonlinear
1	0.1	-	Linear
2	1.0	0.85	Nonlinear
3	0.1	-	Linear

of PROXDDP. We excluded both ALTRO, and PROXDDP in nonlinear rollout mode, from this benchmark – we tried multiple configurations, but both solvers diverged after a few iterations. The PROXDDP configurations are shown in the table below:

Config.	μ_0	Linesearch γ	Rollout
0	1e-2	0.0 (monotone)	Linear
1	1e-4	-	-
2	1e-2	0.2	-
3	1e-4	-	-
4	1e-2	0.85	-
5	1e-4	-	-

This allows us to both compare with IPOPT (especially with respect to timings), and see how our solver's performance varies on this problem with different hyperparameter configurations.

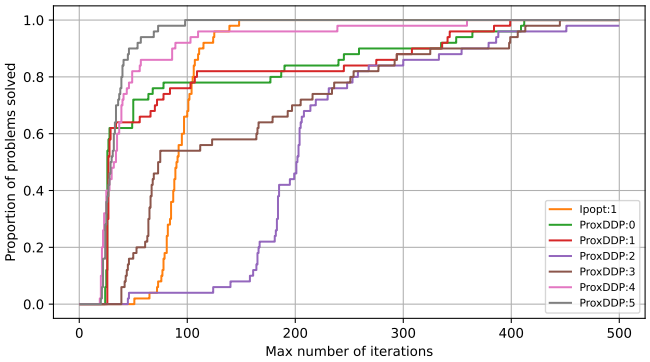
We generate $M = 50$ random instances of the problem by changing the dip and twist angles in the robot's movements. The proportion of problems solved as a function of the maximum number of iterations, and the corresponding performance profile (with respect to wall time), are shown in Fig. 17. On this example, the linesearch moving average weight γ (from Algorithm 2) has a large effect on the solver's performance. One configuration of IPOPT (the Gauss-Newton one, see again Tab. IV) fails on all instances. Furthermore, we include a plot of the distribution of average time per iteration in Fig. 18. The timings are computed by dividing the solve time of each instance by the corresponding number of iterations; thus, they include all algorithmic features, such as the function evaluations in linesearch, iterative refinement in linear solvers, IPOPT's feasibility restoration procedure, and so on.

C. Ballistics with an UR10 manipulator (benchmark)

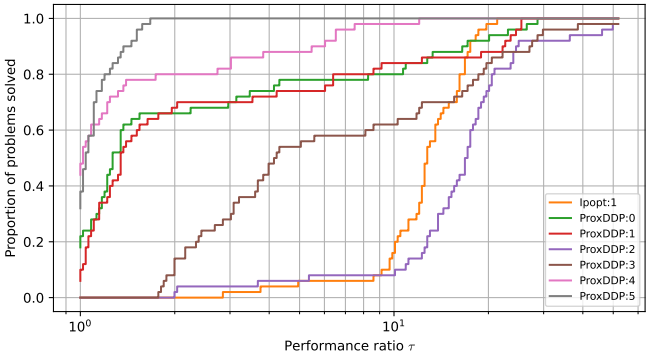
This benchmark is a variation of the task from the example in Sec. VII-B, which had torque limits on the manipulator and a hard constraint on the terminal target position of the thrown projectile. In order to make the problem Euclidean, we simplified the object to only have 3D movement (meaning no 3D rotation).

We generate $M = 50$ random instances of the problem, with randomized targets. Each random instance of the problem samples a target p for the projectile to reach, normally distributed as $p \sim \mathcal{N}((2.4, 0.0, 0.5)^\top, 5I_3)$.

The tested configurations for PROXDDP are shown in Tab. V. The benchmark metrics (proportion of problems solved, performance profile) are shown Fig. 19. We also include here a plot of average timings per solver iteration in Fig. 20.



(a) Proportion of problems solved as a function of a maximum number of iterations.



(b) Performance profile with respect to wall time.

Fig. 17. **Solo-12 benchmark.** Target tolerance set to $\epsilon_{\text{tol}} = 10^{-3}$.

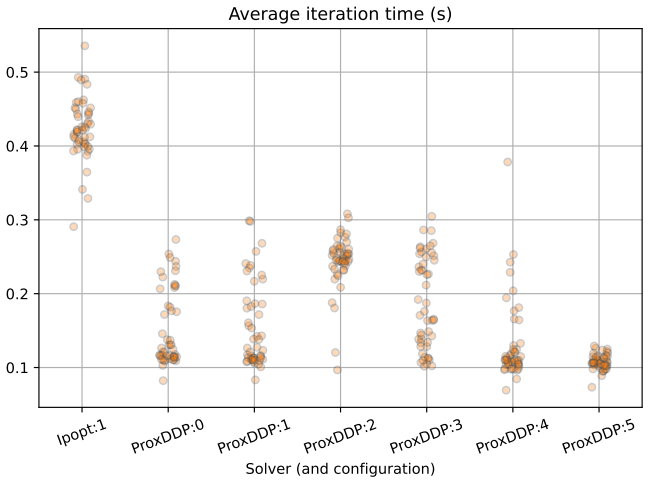
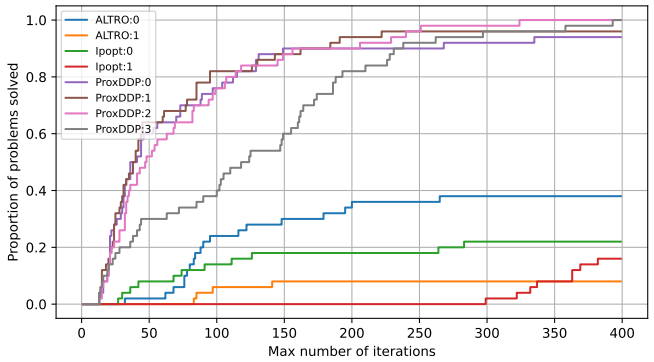


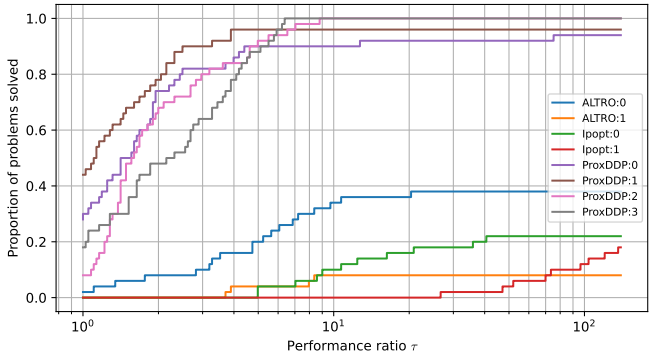
Fig. 18. **Solo-12 benchmark.** Average timings (in seconds) per iteration of each solver.

D. Discussion

Choice of initial parameters, namely μ_0, is paramount to the solver's behavior on a given problem. Our solver implements very few heuristics beyond BCL and a rule for correcting the inertia of the system matrix (37). Mature solvers such as IPOPT implement many more heuristics for initial parameter choice, initialization, dealing with too many small step sizes



(a) Proportion of problems solved as a function of the maximum number of iterations.



(b) Performance profile with respect to wall time.

Fig. 19. **UR10 ballistic benchmark.** The tolerance for the problem is set to $\epsilon_{\text{tol}} = 10^{-5}$.

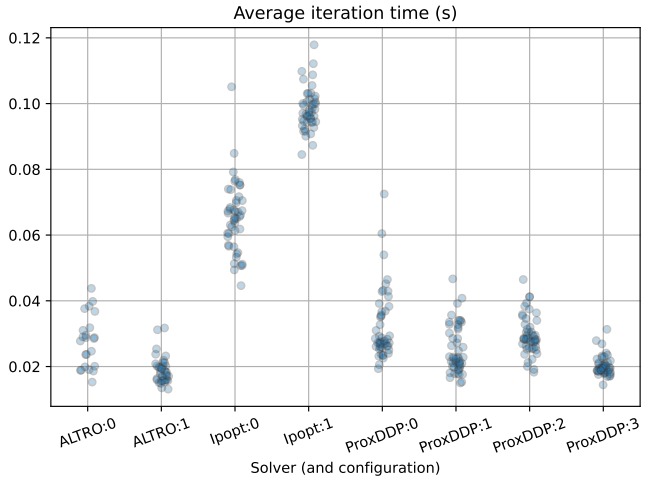


Fig. 20. **UR10 ballistic benchmark.** Average timings (in seconds) per iteration of each solver.

in linesearch, lack of progress on constraints, and so on; these heuristics were tested and tuned on large problem sets such as CUTEr [81]. This underlines the need for a comprehensive benchmark for OCPs in robotics, which could be a sandbox for testing dedicated strategies for nonlinear OCP solvers.

Finding the right heuristics for automatic tuning of a nonlinear solver's parameters is a problem in itself. In the realm of augmented Lagrangian methods, the ALGENCAN package [82] implement a few heuristics for choosing initial AL penalties, multipliers; a book from the same authors [83] focuses on these aspects of designing practical algorithms.

Overall, the benchmark results show that PROXDDP is a competitive option for quickly solving nonlinear OCPs compared to IPOPT and ALTRO. IPOPT has shown, as expected, to be a very robust solver, but not suitable for real-time applications. The results also show that asking ALTRO for a tighter stationarity tolerance – which is both a tolerance and a hyperparameter for this algorithm – leads to poor performance. In contrast, PROXDDP exhibits timings which are more compatible with real-time applications than IPOPT while being competitive with it, and more reliable than ALTRO.

IX. CONCLUSION

In this paper, we have introduced 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 for constrained trajectory optimization. Our solver is implemented in a new open-source C++ framework for numerical OC called ALIGATOR, an additional software contribution introduced in this paper which we hope the wider robotics community can build upon. We have demonstrated our solver working on several real-size OCPs for various classes of robots requiring hard constraint satisfaction, and showed its capabilities for whole-body MPC on a quadruped. Additionally, we have compared our solver to two other solvers: the generic NLP solver IPOPT and the C++ implementation of the TO solver ALTRO. We hope this work will entice future developments in robust, real-time constrained trajectory optimization solvers for model-predictive control and other applications such as learning policies [84], [85].

In future work, we consider going deeper into a couple of subjects. We plan to continue work on the contributed 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 effort akin to what is being done with both PINOCCHIO and CROCODDYL. Second is further work on faster structure-exploiting linear algebra solvers, both for the subproblems (38) and for the backward-pass/forward-pass algorithm well as the overall algorithm, e.g. parallelization. We have already moved on this second point, with a publication on parallelizing our Riccati algorithm [86]. We expect this to enable more complex formulations for MPC on larger systems such as humanoids and dexterous hands, or facilitating the deployment of MPC solutions on 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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.*, Oct. 2012, pp. 4906–4913.

[15] 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.

[16] 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."

[17] 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.

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

[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," *Optimal Control Applications and Methods*, vol. 45, no. 1, pp. 436–454, 2024.

[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. Sola, N. Mansard, and S. Vijayakumar, "A feasibility-driven approach to control-limited ddp," Aug. 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. 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] A. Conn, N. I. M. Gould, and P. Toint, *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, ser. Springer Series in Computational Mathematics. Heidelberg: Springer Verlag, 1992.

[35] 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.

[36] 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.

[37] A. Conn, N. I. M. 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.

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

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

[44] 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.

[45] R. A. Poliquin and R. T. Rockafellar, "Generalized hessian properties of regularized nonsmooth functions," *SIAM Journal on Optimization*, vol. 6, no. 4, pp. 1121–1137, Nov. 1996.

[46] R. T. Rockafellar, "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.

[47] 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.
[48] R. T. Rockafellar, "Monotone operators and the proximal point algorithm," *SIAM Journal on Control and Optimization*, vol. 14, no. 5, pp. 877–898, Aug. 1976.
[49] 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.
[50] 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.
[51] S. Kazdadi, J. Carpentier, and J. Ponce, "Equality constrained differential dynamic programming," in *ICRA 2021 - IEEE International Conference on Robotics and Automation*, May 2021.
[52] 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.
[53] H. Zhang and W. W. Hager, "A nonmonotone line search technique and its application to unconstrained optimization," *SIAM Journal on Optimization*, vol. 14, no. 4, pp. 1043–1056, Jan. 2004.
[54] 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.
[55] 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.
[56] F. Farshidian *et al.*, "OCS2: An open source library for optimal control of switched systems," [Online]. Available: https://github.com/leggedrobotics/ocs2.
[57] F. H. Clarke, *Optimization and nonsmooth analysis*. SIAM, 1990.
[58] M. Hintermuller, "Semismooth newton methods and applications," Humboldt-Universität zu Berlin, Tech. Rep., 2010.
[59] L.-Z. Liao and C. Shoemaker, "Advantages of differential dynamic programming over newton's method for discrete-time optimal control problems," Cornell University, Tech. Rep., Feb. 1993.
[60] 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.
[61] J. Hauser, "A projection operator approach to the optimization of trajectory functionals," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 377–382, Jan. 2002.
[62] F. Bayer, G. Notarstefano, and F. Allgöwer, "A projected sqp method for nonlinear optimal control with quadratic convergence," in *Proceedings of the IEEE Conference on Decision and Control*, Firenze, Italy, Dec. 2013, p. 6468.
[63] 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.
[64] L. Sforni, S. Spedicato, I. Notarnicola, and G. Notarstefano, "Gopronto: A feedback-based framework for nonlinear optimal control," Aug. 2021.
[65] 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.
[66] 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.
[67] L. Vanroye, A. Sathya, J. De Schutter, and W. Decré, "Fatrop : A fast constrained optimal control problem solver for robot trajectory optimization and control," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Detroit, MI, USA: IEEE, Oct. 2023.
[68] 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.
[69] 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.
[70] M. Toussaint, "A novel augmented lagrangian approach for inequalities and convergent any-time non-central updates," Dec. 2014.
[71] 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.
[72] 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.
[73] 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.
[74] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.
[75] 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.
[76] 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.
[77] 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.
[78] 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.
[79] 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.
[80] 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.
[81] N. I. M. Gould, D. Orban, and P. Toint, "Cuter and sifdec: A constrained and unconstrained testing environment, revisited," *ACM Trans. Math. Softw.*, vol. 29, no. 4, pp. 373–394, Dec. 2003.
[82] E. G. Birgin, R. A. Castillo, and J. M. Martínez, "Numerical comparison of augmented lagrangian algorithms for nonconvex problems," *Computational Optimization and Applications*, vol. 31, no. 1, pp. 31–55, May 2005.
[83] E. G. Birgin and J. M. Martínez, *Practical Augmented Lagrangian Methods for Constrained Optimization*, ser. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, May 2014.
[84] Q. Le Lidec, W. Jallet, I. Laptev, C. Schmid, and J. Carpentier, "Enforcing the consensus between trajectory optimization and policy learning for precise robot control," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, May 2023, pp. 946–952.
[85] F. Liu, Z. Gu, Y. Cai, Z. Zhou, S. Zhao, H. Jung, S. Ha, Y. Chen, D. Xu, and Y. Zhao, "Opt2skill: Imitating dynamically-feasible whole-body trajectories for versatile humanoid loco-manipulation," Oct. 2024.
[86] W. Jallet, E. Dantec, E. Arlaud, N. Mansard, and J. Carpentier, "Parallel and proximal constrained linear-quadratic methods for real-time nonlinear mpc," in *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, Jul. 2024.

APPENDIX

A. Relationship between PROXDDP and multiple shooting

Similarly to [51], we can solve multiple shooting formulations with an equality constraint-capable DDP by using slack variables (s_t) for the dynamics:

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

and add the constraint that $s_t = 0$. The TO problem becomes:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u},\boldsymbol{s}}\ & J(\boldsymbol{x},\boldsymbol{u}) \\ \text{s.t. } & x_{t+1} = f(x_t, u_t) - s_t, \\ & s_t = 0. \end{aligned} \tag{50}$$

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

$$Q_t(x,u,s,\lambda) = \ell_t(x,u) + V_{t+1}(f(x,u) - s) + \lambda^\top s. \quad (51)$$

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

$$\begin{aligned} 0 &= Q_u = \ell_u + f_u^\top V'_x \\ 0 &= Q_s = -V'_x + \lambda^\star \\ 0 &= s^\star. \end{aligned} \quad (52)$$

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 [51], 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 (where $\lambda_{k+1} = \lambda^\star$):

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

B. More generic initial conditions in the PROXDDP *algorithm*

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:

$$\begin{bmatrix} W_0 & \omega_x^\top \\ \omega_x & -\mu_k I \end{bmatrix} \begin{bmatrix} \delta x_0 \\ \delta \lambda_0 \end{bmatrix} = - \begin{bmatrix} V_{x0} + \omega_x^\top \lambda_0 \\ \mu_k(\lambda_{k,0} - \lambda_0) + \omega(x_0) \end{bmatrix}, \quad (53)$$

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