PROXQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond

Antoine Bambade, Fabian Schramm, Sarah El Kazdadi, Stéphane Caron, Adrien Taylor, Justin Carpentier

PROXQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond

Antoine Bambade*,[1,2], Fabian Schramm[1], Sarah El-Kazdadi[1], Stéphane Caron[1], Adrien Taylor[1] and Justin Carpentier[1]

***Abstract*—Convex Quadratic programming (QP) has become a core component in the modern engineering toolkit, particularly in robotics, where QP problems are legions, ranging from real-time whole-body controllers to planning and estimation algorithms. Many of those QPs need to be solved at high frequency. Meeting timing requirements requires taking advantage of as many structural properties as possible for the problem at hand. For instance, it is generally crucial to resort to warm-starting to exploit the resemblance of consecutive control iterations. While a large range of off-the-shelf QP solvers is available, only a few are suited to exploit problem structure and warm-starting capacities adequately. In this work, we propose the PROXQP algorithm, a new and efficient QP solver that exploits QP structures by leveraging primal-dual augmented Lagrangian techniques. For convex QPs, PROXQP features a global convergence guarantee to the closest feasible QP, an essential property for safe closed-loop control. We illustrate its practical performance on various standard robotic and control experiments, including a real-world closed-loop model predictive control application. While originally tailored for robotics applications, we show that PROXQP also performs at the level of state of the art on generic QP problems, making PROXQP suitable for use as an off-the-shelf solver for regular applications beyond robotics.**

***Index Terms*—Optimization, Quadratic Programming, Embedded optimization**

I. INTRODUCTION

Optimization has become a key enabler to simplify and systematize the programming of complex robot motions. Nowadays, many robotic problems, ranging from simulation and control to planning and estimation, are framed as optimization problems. An important class of such optimization problems is that of convex quadratic programs (QP), which allows dealing with friction-less unilateral contact modelling [1], constrained forward dynamics [2], inverse kinematics and dynamics for task control [3]–[5], and legged locomotion [6]–[8], among others. QPs are also commonly used as subroutines for solving more complex problems, for instance, in the context of constrained optimal control problems [9]–[12].

Formally, a QP corresponds to the minimization of a convex quadratic cost under linear inequality constraints. It is mathematically described as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \left\{ f(x) \stackrel{\text{def}}{=} \frac{1}{2} x^\top H x + g^\top x \right\} \\ \text{s.t.} \quad & Cx \leqslant u, \end{aligned} \tag{QP}$$

where $H \in \mathbb{R}^{n \times n}$ is a real symmetric positive semi-definite matrix (notation $\mathbb{S}^n_+$), $g \in \mathbb{R}^n$, $C \in \mathbb{R}^{m \times n}$, and $u \in \mathbb{R}^m$. The problem dimension is n, while m corresponds to the numbers inequality constraints.

In many scenarios, QP instances have to be solved at high frequency (*e.g.*, 1 kHz is typical for inverse kinematics or dynamics), under various levels of accuracy depending on the application, and potentially in relatively large dimensions (*e.g.*, humanoid robots) particularly when it comes to model predictive control (MPC). In such contexts, it is valuable to exploit as much as possible the structure offered by the task (e.g., sparsity pattern of the underlying problem) and to make use of an optimization method that benefits from various warm-starting features. Yet, in a robotic context, warm-starting a QP with an initial guess from a previous instance requires it to lie in the feasible set of the new instance. This property, also called *recursive feasibility* [13] in the MPC literature, is challenging to enforce for closed-loop systems as the latest state estimated from sensory measurements may not be feasible, prompting practitioners to relax the initial state constraint [14], [15]. These practical reasons may limit the use of warm-starting and hot-starting to sequential problems for which recursive feasibility can be guaranteed, such as open-loop [13] rather than closed-loop [15] MPC.

This work develops a new augmented Lagrangian-based QP method and solver with the following contributions:

- ↪ The PROXQP algorithm itself, a generic method for solving convex QPs, based on a primal-dual proximal augmented Lagrangian method. In particular, we provide a convergence proof of the overall algorithm and highlight that the PROXQP algorithm, and more generally primal-dual Proximal augmented Lagrangian methods, actually solves the *closest* primal-feasible QP—in a classical ℓ_2 sense that we detail in the sequel—if the original QP appears to be primally infeasible.
- ↪ On the software side, we distribute an open-source and flexible implementation of PROXQP within the PROXSUITE library https://github.com/Simple-

*Corresponding author. **bambade.antoine@gmail.com**
[1]Inria - Département d'Informatique de l'École normale supérieure, PSL Research University.
[2]École des Ponts, Marne-la-Vallée, France.

Robotics/proxsuite. The solver is written in C++ for efficiency and comes with Julia and Python bindings.

↪ On the numerical side, we illustrate the practical performance of PROXQP on different standard sparse and dense robotics and control experiments, including real-world closed-loop problems. We notably highlight how solving the closest primal feasible QPs can be leveraged in closed-loop model predictive control. We also benchmark the performance of PROXQP against state-of-the-art QP solvers on various problem sets of generic QPs from the literature.

This article is an extension of the previously published conference paper [16]. The updated version of PROXQP utilizes a modified primal-dual augmented Lagrangian merit function, which has demonstrated improved practical efficiency in solving the sequence of proximal subproblems Additionally, we now prove that the PROXQP algorithm features a global convergence guarantee, as well as other advantageous numerical properties (see Section III-B). Moreover, we highlight a central feature of PROXQP for adequately handling primally infeasible QPs, which can be exploited in a closed-loop model predictive control scheme, as illustrated in the experimental section (see Sec. VI) or in machine learning context as leveraged in [17]. Last but not least, on the software side, PROXQP handles now sparse problems with a specific and dedicated backend. Finally, this article extends the validation benchmark to include standard control problems and a variety of robotics tasks.

The remainder of this paper is organized as follows. Section II reviews augmented Lagrangian methods for solving quadratic programs, which are at the core of our approach. Section III presents the PROXQP algorithm and its theoretical features. Section IV provides an overview of the diverse optimization methods commonly used for solving QPs, and highlights the benefits of the primal-dual augmented Lagrangian approach leveraged in PROXQP. Section V details the various features of the PROXQP implementation within the ProxSuite library. Finally, Section VI provides an extensive validation of PROXQP on different robotic and control tasks. We also benchmark the numerical robustness, accuracy, and speed of our approach against state-of-the-art solvers on more generic QPs. We also highlight with specific benchmarks the differences between PROXQP and other related Augmented-Lagrangian based approaches. Due to space limitations and to facilitate the overall reading of the paper, the annexes are collected in the companion report [18].

II. BACKGROUND

In this section, we recall some of the well-established components underlying PROXQP, namely, (i) optimality conditions related to (QP), and (ii) existing augmented Lagrangian-based strategies. The reader already familiar with this topic can skip this section and directly dive into Section III, which details the PROXQP algorithm.

A. Notation

The Lagrangian $\mathcal{L}$ associated to (QP) is defined by:

$$\mathcal{L}(x,z) \overset{\text{def}}{=} f(x) + z^\top (Cx-u),$$

where $x \in \mathbb{R}^n$ are the primal variables and $z \in \mathbb{R}^m_+$ corresponds to the dual variables. The dual function is then defined for $z \geqslant 0$ as $\delta(z) \overset{\text{def}}{=} \inf_{x\in\mathbb{R}^n} \mathcal{L}(x,z)$. The dual problem to (QP) corresponds to:

$$\max_{z\geqslant 0} \ \delta(z). \tag{Dual-QP}$$

We say that (Dual-QP) is feasible when $\delta < +\infty$.

B. Optimality conditions

For linearly constrained convex optimization problems such as (QP), strong duality holds (when either the primal or the dual problem is feasible), and the associated Karush-Kuhn-Tucker (KKT) conditions are necessary and sufficient for optimality of a primal-dual point (x,z) (see e.g. [19, Section 2.1]). For (QP), the KKT system is:

$$\begin{cases} Hx + g + C^\top z = 0, \\ z \geqslant 0, \\ Cx \leqslant u, \\ x^\top Hx + g^\top x + u^\top z = 0. \end{cases} \tag{KKT}$$

In practice, we look for a tuple (x,z) satisfying these optimality conditions (KKT) up to a certain level of predefined accuracy $\epsilon_{\text{abs}} > 0$ (whose value typically depends on the application at hand), in the following sense:

$$\begin{cases} \|Hx + g + C^\top z\|_\infty \leqslant \epsilon_{\text{abs}}, \\ \|[Cx-u]_+\|_\infty \leqslant \epsilon_{\text{abs}}, \\ |x^\top Hx + g^\top x + u^\top z| \leqslant \epsilon_{\text{abs}}, \end{cases} \tag{1}$$

where $[.]_+$ stands for the component-wise projection onto the nonnegative orthant, *i.e.* $[v]_{+,i} = \max(0, v_i)$. In this work, we prefer the ℓ_∞ norm to the ℓ_2 norm as it does not scale with the dimension d. It is also common practice to rely on relative error criteria for early-stopping, as absolute targets might be out of reach due to numerical issues [20]–[22]. Although this option is included in the software, in what follows, we will not add relative terms after absolute ones for the sake of simplicity.

C. Method of Multipliers

The method of multipliers (MM) is based on the minimization of the augmented Lagrangian (AL) penalty function, and is a precursor to the algorithm developed in this work. AL-based approaches typically feature advantageous properties such as convergence toward a solution to the closest feasible QP (see Section VI for examples in robotics).

Augmented Lagrangian penalty function. The so-called AL function [23, Section 4] relies on augmenting the cost function f with a shifted ℓ_2 penalization of the constraint (*i.e.*, $Cx-u$) with respect to a weighted dual multipliers μz:

$$\mathcal{L}_A(x,z;\mu) \overset{\text{def}}{=} f(x) + \tfrac{1}{2\mu}\left(\|[Cx-u+\mu z]_+\|_2^2 - \|\mu z\|_2^2\right),$$

where $\mu > 0$ is a positive penalty parameter. In the case where we replace the inequality constraints $Cx \leqslant u$ of (QP) by equalities $Cx = u$, this would be equivalent to augmenting the standard Lagrangian $\mathcal{L}$ with a weighted penalization:

$$\mathcal{L}_A(x, z; \mu) = \mathcal{L}(x, z) + \tfrac{1}{2\mu}\|Cx - u\|_2^2.$$

Method of multipliers. Many optimization strategies rely on alternating (i) minimizing $\mathcal{L}_A$ with respect to the primal variables, and (ii) maximizing $\mathcal{L}_A$ with respect to the dual variables. MM [24], [25] is one such strategy, where the maximization step (ii) is computed in closed-form. More precisely, considering some sequence $\{\mu^k\}_k$ bounded below by some $\mu^\infty > 0$:

$$\begin{aligned} x^{k+1} &= \operatorname*{arg\,min}_{x\in\mathbb{R}^n} \mathcal{L}_A(x, z^k; \mu^k), \\ z^{k+1} &= \operatorname*{arg\,max}_{z\in\mathbb{R}^m} \mathcal{L}_A(x^{k+1}, z; \mu^k) = \left[z^k + \tfrac{1}{\mu^k}(Cx^{k+1} - u)\right]_+. \end{aligned}$$

Yet, in the presence of inequality constraints, the exact minimization of $\mathcal{L}_A(x, z^k; \mu^k)$ with respect to x is generally hard to compute in closed-form since $x \mapsto \mathcal{L}_A(x, z^k; \mu^k)$ is piece-wise quadratic [26]. For this reason, practical MM solvers typically rely on ϵ^k-approximate solutions for computing x^{k+1}:

$$\begin{aligned} x^{k+1} &\approx_{\epsilon^k} \operatorname*{arg\,min}_{x\in\mathbb{R}^n} \mathcal{L}_A(x, z^k; \mu^k), \\ z^{k+1} &= \left[z^k + \tfrac{1}{\mu^k}(Cx^{k+1} - u)\right]_+, \end{aligned} \tag{2}$$

for some notion of "$\approx_{\epsilon^k}$". For instance, we can use:

$$\|\nabla_x \mathcal{L}_A(x^{k+1}, z^k; \mu^k)\|_\infty \leqslant \epsilon^k.$$

For ensuring convergence of the numerical scheme, one typically needs to enforce certain properties on ϵ^k and $\{\mu^k\}_k$. For instance, it is clear that one must have $\epsilon^k \to 0$; other conditions include summability of the sequence $\{\epsilon^k\}_k$, see, e.g., [23], [27], [28]. The boundedness condition on $\{\mu^k\}_k$ turns out also to be necessary for ensuring convergence [23].

Convergence properties. It turns out that MM can be seen as a proximal point method applied to (Dual-QP), see [23, Section 4]. Therefore, it is guaranteed that $\{z^k\}_k$ converges to an optimal solution to (QP) as soon as there exists one [23, Theorem 4]. Furthermore, $\{x^k\}_k$ is a minimizing sequence, *i.e.*, $f(x^k) \to \min_{x\in\mathbb{R}^n} f(x)$. Note that $\{x^k\}_k$ does not necessarily converge towards a solution to (QP) [23]. This equivalence between MM and the dual proximal point algorithm also allows MM to inherit a key property for handling primal infeasible problems. Indeed, assuming (Dual-QP) to be feasible, the dual iterates of MM actually converge linearly to a solution of the dual to the following hierarchical problem [29]:

$$\begin{aligned} s^\star = \operatorname*{arg\,min}_{s\in\mathbb{R}^m} \ & \tfrac{1}{2}\|s\|_2^2 \\ \text{s.t. } x^\star, z^\star \in \operatorname*{arg\,min}_{x\in\mathbb{R}^n} & \max_{z\in\mathbb{R}^m_+} L(x, z, s), \end{aligned} \tag{QP-H}$$

with $L(x, z, s) \overset{\text{def}}{=} f(x) + z^\top(Cx - u - s)$ (namely the Lagrangian of (QP) augmented with a shift variable s). Under strict convexity of f, it results that as soon as (QP) is primal infeasible (i.e., there is no x s.t. $Cx \leqslant u$), then MM will converge towards a solution $(x^\star, z^\star)$ of (QP-H). Such a solution satisfies the following KKT optimality conditions:

$$\begin{aligned} &Hx^\star + g + C^\top z^\star = 0, \\ &Cx^\star \leqslant u + s^\star, \\ &(x^\star)^\top H x^\star + g^\top x^\star + (u + s^\star)^\top z^\star = 0. \end{aligned}$$

In other words, s is the smallest possible (measured in ℓ_2 norm) shift on the constraints that makes (QP) feasible.

D. Proximal Method of Multipliers

The proximal method of multipliers (PMM) is an extension of MM with stronger convergence guarantees.

Proximal augmented Lagrangian function. For a certain $\rho > 0$ ($1/\rho$ corresponds to a step-size), PMM relies on an additional proximal term on the primal variables with [23, Equation 1.9], considering an additional sequence $\{\mu^k\}_k$ bounded below by some $\mu^\infty > 0$:

$$\begin{aligned} (x^{k+1}, z^{k+1}) = \operatorname*{arg\,min}_{x\in\mathbb{R}^n} \max_{z\in\mathbb{R}^m_+} & \mathcal{L}(x, z) + \tfrac{\rho}{2}\|x - x^k\|_2^2 \\ & - \tfrac{\mu^k}{2}\|z - z^k\|_2^2. \end{aligned} \tag{3}$$

Assuming that one can solve (3), one can generate a sequence $\{(x^{k+1}, z^{k+1})\}_k$ that converges to a primal-dual optimal pair for (QP). However, (3) involves a convex-concave saddle-point problem, for which numerous numerical methods exist yet closed-form solutions can seldom be found. For solving this problem, a natural strategy consists in splitting the optimization procedure in two parts:

$$\begin{cases} x^{k+1} = \arg\min_{x\in\mathbb{R}^n} \Phi^k_{\rho,\mu^k}(x), \\ z^{k+1} = \arg\max_{z\in\mathbb{R}^m_+} \mathcal{L}(x, z) - \frac{\mu^k}{2}\|z - z^k\|_2^2, \end{cases}$$

where $\Phi^k_{\rho,\mu^k}(x) \overset{\text{def}}{=} \mathcal{L}_A(x, z^k; \mu^k) + \frac{\rho}{2}\|x - x^k\|_2^2$ is often referred to as the Proximal augmented Lagrangian [30], and where the second line can be computed in closed form as $z^{k+1} = \left[z^k + \frac{1}{\mu^k}(Cx^{k+1} - u)\right]_+$. As with MM, one generally needs to resort on numerical methods for the first minimization stage, which we can only solve approximately:

$$\begin{cases} x^{k+1} \approx_{\epsilon^k} \arg\min_{x\in\mathbb{R}^n} \Phi^k_{\rho,\mu^k}(x), \\ z^{k+1} = \left[z^k + \frac{1}{\mu^k}(Cx^{k+1} - u)\right]_+, \end{cases}$$

for some notion of "$\approx_{\epsilon^k}$". For instance, we can use:

$$\left\|\nabla_x \Phi^k_{\rho,\mu^k}(x^{k+1})\right\|_\infty \leqslant \epsilon^k.$$

Again, it is clear that convergence of $\{(x^k, z^k)\}$ towards a primal-dual solution requires certain properties of the approximation error, such as $\epsilon^k \to 0$. Other conditions include summability of $\{\epsilon^k\}_k$, and $\{\mu^k\}$ being bounded from below by some $\mu^\infty > 0$, see, e.g., [23, Theorem 5].

Convergence properties. PMM enjoys the key advantage of guaranteed primal-dual convergence of its iterates under relatively weak assumptions, namely the existence of an optimal primal-dual pair with zero duality gap [23, Theorem 7]. This guarantee is stronger than that of MM, for which only the dual variables are guaranteed to converge [23, Theorem 4].

Furthermore, the proximal augmented Lagrangian function is a piece-wise quadratic strongly convex function, unlike the augmented Lagrangian $x \mapsto \mathcal{L}_A(x, z^k; \mu^k)$ which is only guaranteed to be convex. This allows accelerating the computation of the iterates, e.g., using quadratic Newton-type methods [31]. We leverage this key feature in PROXQP, as further detailed in Section III. Finally, PMM inherits from MM convergence towards the closest feasible QP solution when the problem is primal infeasible (QP-H), see Remark 1.

E. A primal-dual Proximal Method of Multipliers

Primal-dual proximal methods of multipliers (PDPMM) is an alternative formulation of PMM. Its name stems from its choice of a primal-dual penalty function for evaluating the quality of the iterates of the sequence of proximal sub-problems. When intermediary optimization subproblems are solved exactly, PMM and PDPMM produce the same iterates. Thereby, PDPMM benefits from the strong convergence properties of PMM. However, the alternate formulation of PMM is convenient and can be leveraged at computation time, and we leverage it in our algorithms.

Primal-dual proximal augmented Lagrangian function. The author of [32] shows that (3) can be equivalently reformulated as a minimization of the following primal-dual merit function:

$$\begin{aligned}\mathcal{M}^k_{\rho,\mu}(x,z) \overset{\text{def}}{=} & f(x) + \tfrac{1}{\mu}\left\|[Cx - u + \mu(z^k - \tfrac{z}{2})]_+\right\|_2^2 + \\ & \tfrac{\rho}{2}\|x - x^k\|_2^2 + \tfrac{\mu}{4}\|z\|_2^2.\end{aligned} \tag{4}$$

Hence (3) is equivalent to:

$$(x^{k+1}, z^{k+1}) = \underset{x\in\mathbb{R}^n, z\in\mathbb{R}^m}{\operatorname{arg\,min}} \mathcal{M}^k_{\rho,\mu^k}(x,z). \tag{5}$$

Of course, solving this optimization step still requires using numerical schemes, and can therefore only be approximated:

$$(x^{k+1}, z^{k+1}) \approx_{\epsilon^k} \underset{x\in\mathbb{R}^n, z\in\mathbb{R}^m}{\operatorname{arg\,min}} \mathcal{M}^k_{\rho,\mu^k}(x,z), \tag{6}$$

for some notion of "$\approx_{\epsilon^k}$". For instance, [32] uses:

$$\left\|\begin{bmatrix}\nabla_x f(x^{k+1}) + C^\top z^{k+1} + \rho(x^{k+1} - x^k) \\ [Cx^{k+1} - u + \mu^k(z^k - z^{k+1}/2)]_+ - \mu^k z^{k+1}/2\end{bmatrix}\right\|_\infty \leqslant \epsilon^k.$$

The PROXQP algorithm relies on a similar error criterion as detailed in the next sections.

Convergence properties. As for PMM, the recurrence (6) guarantees that $\{(x^k, z^k)\}_k$ converges to an optimal primal-dual pair under the relatively weak assumptions (the same as in (II-D0a): as soon as there exists a solution to (QP), and $\{\mu^k\}_k$ is bounded below by some $\mu^\infty > 0$). Furthermore, $\mathcal{M}^k_{\rho,\mu^k}$ is both strongly convex and piece-wise quadratic with respect to both the primal and the dual variables. Hence, as for $\Phi^k_{\rho,\mu}$, its minimization can be performed efficiently using semi-smooth Newton-type methods. As we show in Section III, the linear systems induced by the minimization of $\mathcal{M}^k_{\rho,\mu^k}$ are naturally better conditioned than those arising from Φ^k_{ρ,μ^k}, which is due to the fact that this formulation naturally avoids quadratic matrix multiplications (such as $C^\top C$). Finally, Remark 1 highlights that more generally to MM methods, PMM and PDPMM both converge towards the primal-dual pair of the closest feasible QP solution, which is useful when (QP) is infeasible.

Remark 1 (Convergence towards solutions of (QP-H))**.** *It can be proven following [29] that if* (QP) *is primal infeasible, its dual* (Dual-QP) *is feasible, and* $\{\mu^k\}_k$ *is a sequence satisfying* $\forall k \in \mathbb{N},\ \mu^k \geqslant \mu^\infty$ *for some* $\mu^\infty > 0$*, then, the sequence* $\{(x^k, z^k)\}$ *generated by* (3) *converge towards a solution of* (QP-H)*.*

F. Discussions

There are three practical ingredients that are key when implementing AL-based approaches.

1 – Setting ϵ^k. Appropriate choices for ϵ^k are fundamental to any AL method.

- On the one hand, picking too small values for ϵ^k leads to unnecessary good approximations of the intermediate optimization problems, even when the iterates are still far away from solutions to (QP). Practically, this scenario corresponds to requiring the internal solver to unnecessarily run for too long when solving the intermediary problems.
- On the contrary, picking ϵ^k too large results in faster computations of the intermediary subproblems at the cost of requiring much more iterations of the AL method, which might even possibly not converge.

There are many different techniques for fixing ϵ^k in practice [23], [31], [33]. For PROXQP, we chose to rely on an adaptive strategy similar in spirit to [27].

2 – Setting μ^k. Using different values for μ^k has a direct impact on the practical convergence speed (along with the value ρ for PMM and PDPMM) (see, e.g., [34, Chapter 5])

- On the one hand, lower values of μ^k lead to faster convergence (i.e., fewer iterations).
- On the other hand, larger values of μ^k render the intermediary subproblems structurally simpler to solve (by increased strong convexity).

Practically speaking, let us mention that in many solvers (including PROXQP), updating μ^k corresponds to re-factorizing some internal matrices. Hence, we must generally limit the number of updates to μ^k for best practical performances.

3 – Efficient method for computing (x^{k+1}, z^{k+1}). There are several different ways to come up with ϵ^k-approximations for (x^{k+1}, z^{k+1}). The choice of the internal routine for this approximation is thereby crucial both for timing and accuracy requirements.

The next section presents our AL-based approach for solving (QP), which we refer to as PROXQP. In particular, we discuss how to handle these three key aspects for computational efficiency.

III. PROXQP

In this section, we introduce the PROXQP algorithm for solving generic QP problems. For clarity issues, we provide the proofs of convergence of PROXQP in the companion report [18].

A. The PROXQP algorithm

Algorithm 1: PROXQP (practical implementation)

Inputs:
- initial states: x^0, z^0,
- initial parameters: ϵ^0_{bcl}, ϵ^0, ϵ_{abs}, ρ, $\mu^0 > 0$
- hyper-parameters: $\mu_f, \beta_{\text{bcl}} \in (0,1)$, $\alpha_{\text{bcl}} \in (0, 1/2)$ with $\alpha_{\text{bcl}} + \beta_{\text{bcl}} < 1$, $\mu_{\min} > 0$, $k_{\max} \in \mathbb{N} \cup \{+\infty\}$.

Initialization:
- preconditioning (see paragraph V-B0d)
- optional initialization (see paragraph V-B0b) of x^0,z^0.

while *Stopping criterion* (1) *not satisfied* **do**
  Compute $(x^{k+1}, \hat{z}^{k+1})$ satisfying (11) at accuracy ϵ^k
  (using a semi-smooth Newton method, see Section III-D);
  if $p^k \leqslant \epsilon^k_{bcl}$ *using* (8) *OR* $k \geqslant k_{\max}$ **then**
    $\mu^{k+1} = \mu^k$, $\epsilon^{k+1} = \epsilon^k \mu^{k+1}$,
    $\epsilon^{k+1}_{\text{bcl}} = \epsilon^k_{\text{bcl}} (\mu^{k+1})^{\beta_{\text{bcl}}}$
    $z^{k+1} = \hat{z}^{k+1}$
  else
    $\mu^{k+1} = \max(\mu_{\min}, \mu_f \mu^k)$
    $\epsilon^{k+1} = \epsilon^0 \mu^{k+1}$, $\epsilon^{k+1}_{\text{bcl}} = \epsilon^0_{\text{bcl}} (\mu^{k+1})^{\alpha_{bcl}}$
    $z^{k+1} = z^k$
  end
  $k \leftarrow k+1$
end

Output: A (x^k, z^k) satisfying the ϵ_{abs}-approximation criterion (1) for problem (QP).

PROXQP is detailed in Algorithm 1. It combines a proximal augmented Lagrangian technique (approximated using an internal primal-dual method) with a globalization strategy for scheduling the internal step-size and accuracy parameters. More precisely, at iteration k, PROXQP computes candidates $x^{k+1}, \hat{z}^{k+1}$ satisfying

$$(x^{k+1}, \hat{z}^{k+1}) \approx_{\epsilon^k} \underset{x \in \mathbb{R}^n, z \in \mathbb{R}^m}{\arg\min} \mathcal{M}^k_{\rho, \mu^k, \alpha}(x, z), \tag{7}$$

for a merit function $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ that we detail in Section III-D. The following steps build on the bound-constrained Lagrangian (BCL) strategy (see, e.g., [27] and [28, Algorithm 17.4]) for scheduling both ϵ^k and μ^k.

The main idea consists in a fast decrease of ϵ^k when the primal residual $\|[C\hat{x}^{k+1} - u]_+\|_\infty$ is small enough. In this case, we accept the approximation with $z^{k+1} = \hat{z}^{k+1}$ and leave μ^k unchanged with $\mu^{k+1} = \mu^k$. Otherwise, we slightly decrease ϵ^k, reject the approximation with $z^{k+1} = z^k$ and decrease μ^k. Decreasing μ^k corresponds to a heavier penalization of the feasibility constraint within the augmented Lagrangian-based function at the next step. Such a strategy has been proved to perform practically well in advanced optimization packages such as LANCELOT [35] as well as in robotics for solving constrained optimal control problems [12], [36]–[38]. We detail this globalization technique alongside its resulting global convergence properties for PROXQP in Section III-B. The method used to minimize $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ is explained in Section III-D.

B. Globalization strategy for scheduling ϵ^k and μ^k

The BCL globalization strategy in PROXQP consists in updating the dual variables z^k—obtained from minimizing a primal-dual proximal augmented Lagrangian merit function (described in Section III-D)—only when the corresponding primal inequality constraint violation (denoted by p^k hereafter) is small enough. More precisely, we use a second internal sequence of tolerances denoted by ϵ^k_{bcl} (which is also tuned within this BCL strategy) and update the dual multipliers only when $p^k \leqslant \epsilon^k_{\text{bcl}}$, where p^k denotes the ℓ_∞ primal feasibility violation:

$$p^k \overset{\text{def}}{=} \left\| [Cx^{k+1} - u]_+ \right\|_\infty . \tag{8}$$

This BCL strategy allows searching for appropriate values for the hyper-parameters ϵ^k, ϵ^k_{bcl}, μ^k. For the constraint penalization parameter μ^k, it proceeds as follows:

- **If $p^k \leqslant \epsilon^k_{\text{bcl}}$:** the primal inequality constraint violation is satisfactory, keep μ^k as is.
- **Otherwise:** the primal inequality constraint violation is too large, decrease μ^k for subsequent proximal subproblems, which has the effect of leading to heavier penalization of infeasibility, enforcing their satisfaction.

About the accuracy parameters ϵ^k and ϵ^k_{bcl}, the update rules are more technical. The motivation underlying those choices is to ensure global convergence: a geometric-decay type update when the primal inequality constraint violation is good enough, and see [27, Lemma 4.1] for when the infeasibility is too large. The detailed strategy is summarized in Algorithm 1.

Convergence properties. We analyze a simpler algorithm (see Algorithm 2), whose small variations are more effective in practice. More precisely, it assumes that at each iteration k of the algorithm $\rho^k = \mu^k$ compared to Algorithm 1, where ρ is fixed. Updating ρ is not efficient in practice since it would require a numerical factorization at each iteration. In this setup and under the assumption that there exists a primal-dual solution to (QP) (see Theorem 1 and Remark 2), the iterates of PROXQP are guaranteed to converge to an optimal primal-dual pair.

Assumption 1. *The problem* (QP) *is feasible.*

Assumption 2. *The iterates* $\{(x^k, z^k)\}_k$ *generated by* PROXQP *are bounded.*

Theorem 1 (Convergence of PROXQP)**.** *Under Assumption 1 (existence of a solution to QP), Assumption 2 (bounded iterates), and with all parameters set as required in the inputs of Algorithm 2, the iterates* $\{(x^k, z^k)\}$ *of* PROXQP *converge to a solution* $(x^\star, z^\star)$ *of* (QP)*.*

Proof. See (Appendix-E) □

Lemma 1. *Under Assumption 1 (existence of solution to* (QP)*) and Assumption 2 (bounded iterates), and with all parameters set as required in the inputs of Algorithm 2,* $\exists K \in \mathbb{N}$*, and* $\mu \in \mathbb{R}$ *such that* $\forall k \geqslant K \ \mu^k = \mu$*. Hence,* $\forall k \geqslant 0, p^{K+k} \leqslant \epsilon^K_{bcl} (\mu^K)^{\beta_{bcl} k}$.

Algorithm 2: PROXQP (analyzed version)

Inputs:
- initial states: x^0, z^0,
- initial parameters: ϵ_{bcl}^0, $\epsilon^0 > 0$, $\rho^0 = \mu^0 \in (0,1)$
- hyper-parameters: $\mu_f < 1$, $\alpha_{\text{bcl}} \in (0, 1/2)$,$\beta_{\text{bcl}} \in (0,1)$ with $\alpha_{\text{bcl}} + \beta_{\text{bcl}} < 1$, $k_{\max} \in \mathbb{N} \cup \{+\infty\}$.

for $k = 0, 1, \ldots$ **do**
  Compute $(x^{k+1}, \hat{z}^{k+1})$ satisfying (11) at accuracy ϵ^k with $\rho^k = \mu^k$;
  if $p^k \leqslant \epsilon_{bcl}^k$ *OR* $k \geqslant k_{\max}$ **then**
    $\mu^{k+1} = \mu^k$, $\epsilon^{k+1} = \epsilon^k \mu^{k+1}$,
    $\epsilon_{\text{bcl}}^{k+1} = \epsilon_{\text{bcl}}^k (\mu^{k+1})^{\beta_{\text{bcl}}}$,
    $z^{k+1} = \hat{z}^{k+1}$
  else
    $\mu^{k+1} = \mu_f \mu^k$,
    $\epsilon^{k+1} = \epsilon^0 \mu^{k+1}$, $\epsilon_{\text{bcl}}^{k+1} = \epsilon_{\text{bcl}}^0 (\mu^{k+1})^{\alpha_{\text{bcl}}}$,
    $z^{k+1} = z^k$.
  end
end

Remark 2 (On the necessity of Assumption 2)**.** *The BCL strategy [27, Algorithm 1 and Algorithm 2] was originally developed for solving more general optimization problems. BCL relies on an augmented Lagrangian strategy. In other words, we build on BCL for constructing an algorithm that exploits the structural properties of simpler optimization problems (namely QPs). Furthermore, rather than being based on a purely AL strategy, we rely on a proximal AL strategy. We managed to get rid of most of the strong assumptions required for ensuring global convergence of BCL [27]. Yet, Assumption* (2) *still appeared necessary in our analysis. We introduced the safeguard parameter $k_{\max} \in \mathbb{N}$ to always enforce the algorithm to ultimately always accept the candidates* (7) *if Assumption 2 does not hold. This simple trick allows inheriting some nice properties from PMM that include convergence under mild assumptions [23, Theorem 5].*

Remark 3 (Advantageous numerical properties)**.** *As a by-product of the proof of Theorem 1, the method is guaranteed to use constant penalization parameters after a finite number of iterations. This is advantageous insofar as (1) we inherit favorable properties of proximal point methods on saddle point, and (2) it limits the number of numerical matrix factorizations.*

C. Infeasible QPs

Finally, Corollary 1 guarantees that, similarly to MM, PMM and PDPMM methods, PROXQP converges towards the primal-dual pair to the closest feasible QP solution, which is useful when (QP) is infeasible.

Corollary 1. *Let* (QP) *be primal infeasible, let its dual* (Dual-QP) *be feasible, let $\mu > 0$ and $k_{\max} \in \mathbb{N}$. The sequence $\{(x^k, z^k)\}$ generated by* PROXQP *with exact sub-problem minimization converges towards a solution of* (QP-H).

Proof. This is a direct consequence of the fact that, after a finite number of iterations, the iterations of PROXQP correspond to those of PMM. □

D. Solving proximal sub-problems

We now explicit the primal-dual proximal augmented Lagrangian merit function used for generating candidates $(x^{k+1}, \hat{z}^{k+1})$ in (7). Then, we explain how to minimize it in quadratic time.

The PROXQP primal-dual proximal augmented Lagrangian merit function. We use the same procedure as in [32] for building our primal-dual proximal augmented Lagrangian merit function $\mathcal{M}^k_{\rho,\mu^k,\alpha}$. By construction, it guarantees that minimizing $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ is equivalent to solving (3). Yet, contrary to $\mathcal{M}_{\rho,\mu^k}$, we make a different choice for the parameter $\alpha \in (0,1)$[1], tuned for better practical performance,

$$\mathcal{M}^k_{\rho,\mu^k,\alpha}(x,z) \overset{\text{def}}{=} f(x) + \tfrac{1}{2\alpha\mu^k}\|[Cx - u + \mu^k(z^k + (\alpha-1)z)]_+\|_2^2 + \tfrac{\rho}{2}\|x - x^k\|_2^2 + \tfrac{(1-\alpha)\mu^k}{2}\|z\|_2^2. \tag{9}$$

This leads to the following inexact proximal scheme

$$(x^{k+1}, z^{k+1}) \approx_{\epsilon^k} \underset{x\in\mathbb{R}^n, z\in\mathbb{R}^m}{\arg\min}\ \mathcal{M}^k_{\rho,\mu^k,\alpha}(x,z), \tag{10}$$

where $\approx_{\epsilon^k}$ corresponds to

$$\left\| \begin{bmatrix} \nabla_x f(x^{k+1}) + C^\top z^{k+1} + \rho(x^{k+1} - x^k) \\ [Cx^{k+1} - u + \mu^k(z^k + (\alpha-1)z^{k+1})]_+ - \alpha\mu^k z^{k+1} \end{bmatrix} \right\|_\infty \leqslant \epsilon^k. \tag{11}$$

Since $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ is strongly convex and piece-wise quadratic, it can be minimized exactly in finite time using a semi-smooth Newton method with exact linesearch [26, Theorem 3].

Semi-smooth Newton method. For convenience, we omit the index k for the iterates of the Newton method arising at iteration k of PROXQP. A semi-smooth Newton method applied to $\mathcal{M}^k_{\rho,\mu^k,\alpha}$, and initialized at $(\hat{x}^{(0)}, \hat{z}^{(0)}) = (x^k, z^k)$ involves finding at the l^{th} inner iteration $\mathrm{d}w \overset{\text{def}}{=} (\mathrm{d}x, \mathrm{d}z)$ satisfying:

$$\nabla^2 \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)}, \hat{z}^{(l)})\mathrm{d}w + \nabla \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)}, \hat{z}^{(l)}) = 0, \tag{12}$$

where $\nabla \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)}, \hat{z}^{(l)})$ and $\nabla^2 \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)}, \hat{z}^{(l)})$ stand respectively for one element of the generalized Jacobian and the generalized Hessian of $\mathcal{M}_{\rho,\mu^k,\alpha}$ [39, Section 7.1].

Once a semi-smooth Newton step $(\mathrm{d}x, \mathrm{d}z)$ has been obtained, the exact line-search procedure consists in finding the unique t^* such that:

$$t^* = \underset{t\geqslant 0}{\arg\min}\ \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)} + t\mathrm{d}x, \hat{z}^{(l)} + t\mathrm{d}z).$$

Similarly to [16], [31], [32] the function $t \mapsto \mathcal{M}^k_{\rho,\mu^k,\alpha}(\hat{x}^{(l)} + t\mathrm{d}x, \hat{z}^{(l)} + t\mathrm{d}z)$ is continuous and piece-wise quadratic with a finite number of breaking points[2]. Hence, one can exactly compute t^*, an optimal solution to this one-dimensional problem. Finally, the semi-smooth Newton method initialized at $(\hat{x}^{(0)}, \hat{z}^{(0)}) = (x^k, z^k)$ generates a sequence $(\hat{x}^{(l)}, \hat{z}^{(l)})$ via the update rule:

$$\hat{x}^{(l+1)} = \hat{x}^{(l)} + t^*\mathrm{d}x,$$
$$\hat{z}^{(l+1)} = \hat{z}^{(l)} + t^*\mathrm{d}z.$$

[1]Notice that with $\alpha = \frac{1}{2}$ we recover $\mathcal{M}_{\rho,\mu^k}$ as in (4).

[2]This feature is very specific to linear and quadratic types of problems and does help reducing the overall computational burden.

Solving linear systems. In this section, we explicit the linear systems induced by the minimization of $\mathcal{M}^k_{\rho,\mu^k,\alpha}$. We show that they are naturally better conditioned than those arising from Φ^k_{ρ,μ^k} since they naturally avoid quadratic matrix multiplications (such as $C^\top C$).

Equation (12) reads:

$$\begin{bmatrix} H+\rho I+\frac{1}{\alpha\mu^k}C^\top(I-P^{(k,l)})C & \frac{\alpha-1}{\alpha}C^\top(I-P^{(k,l)}) \\ \frac{\alpha-1}{\alpha}(I-P^{(k,l)})C & \mu^k(\alpha-1)[-\frac{1}{\alpha}I-\frac{\alpha-1}{\alpha}P^{(k,l)}] \end{bmatrix}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}z\end{bmatrix} \\ = -\begin{bmatrix}\nabla_x f(\hat{x}^{(l)})+\rho(x^{(l)}-x^k)+\frac{1}{\alpha\mu^k}C^\top[w_k^{(l)}-u]_+ \\ \frac{\alpha-1}{\alpha}[w_k^{(l)}-u]_+ +\mu^k(1-\alpha)\hat{z}^{(l)}\end{bmatrix}, \tag{13}$$

where $P^{(k,l)}$ stands for one element of the generalized Jacobian of $[.]_+$ at $w_k^{(l)} \overset{\text{def}}{=} Cx^{(l)}+\mu^k z^k+\mu^k(\alpha-1)z^{(l)}$. $P^{(k,l)}$ consists of a diagonal matrix with entries

$$P^{(k,l)}_{jj} \overset{\text{def}}{=} \begin{cases} 1 & \text{if } C_j^\top\hat{x}^{(l)}+\mu^k z_j^k+\mu^k(\alpha-1)\hat{z}_j^{(l)} \leqslant u_j, \\ 0 & \text{otherwise.}\end{cases}$$

When $P^{(k,l)}_{jj}=1$, it implies (see [32, Section 3.2] for more details)

$$\mathrm{d}z_j = -(\hat{z}^{(l)})_j.$$

The linear system (13) can hence be equivalently formulated as:

$$\begin{cases} \begin{bmatrix} H+\rho I & C^\top_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \\ C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} & -\mu^k I_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\end{bmatrix}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}z_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\end{bmatrix} = \\ -\begin{bmatrix} H\hat{x}^{(l)}+g+\rho(\hat{x}^{(l)}-x^k)+C^\top_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\hat{z}^{(l)}_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} \\ [Cx^{(l)}+\mu^k z^k-\mu^k\hat{z}^{(l)}-u]_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}\end{bmatrix}, \\ \mathrm{d}z_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})^c} = -\hat{z}^{(l)}_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})^c}, \end{cases} \tag{14}$$

where:

$$J_k(x,z) \overset{\text{def}}{=} \left\{ j\in[1,n_i] | C_j^\top x+\mu^k z_j^k+\mu^k(\alpha-1)z_j-u_j>0\right\}$$

refers to the primal-dual active set at (x,z) for the current k^{th} sub-problem, and $J_k(x,z)^c$ stands for its complementary set (i.e., the set of inactive constraints). The linear system (14) is symmetric and non-singular. Thus it can be efficiently solved using a LDLT factorization. Furthermore, one can notice it does not explicitly contain any square matrix multiplications. We will see it is not the case when working with Φ^k_{ρ,μ^k}.

At the l^{th} iteration, a semi-smooth Newton method applied to Φ^k_{ρ,μ^k}, and initialized at $\hat{x}^{(0)}=x^k$ involves finding $\mathrm{d}x$ satisfying $\nabla^2\Phi^k_{\rho,\mu^k}(\hat{x}^{(l)})\mathrm{d}x+\Phi^k_{\rho,\mu^k}(\hat{x}^{(l)})=0$. One ends-up solving [31, Equation 3.4]:

$$\begin{bmatrix} H+\rho I & C^\top_{I_k(\hat{x}^{(l)})} \\ C_{I_k(\hat{x}^{(l)})} & -\mu^k I\end{bmatrix}\begin{bmatrix}\mathrm{d}x\\ \mathrm{d}z\end{bmatrix} = \begin{bmatrix}-\nabla_x\Phi^k_{\rho,\mu^k}(\hat{x}^{(l)})\\ 0\end{bmatrix},$$

where $I_k(\hat{x}^{(l)}) \overset{\text{def}}{=} \{i\in[1,n_i]\,|\,C_i\hat{x}^{(l)}-u_i+z_i^k\mu^k>0\}$ refers to the active set of the current proximal sub-problem, and $C_{I_k(\hat{x}^{(l)})}$ corresponds to a reduced version of the matrix C containing the active rows indexed by $\mathcal{I}_k(\hat{x}^{(l)})$. The gradient on the right-hand side of this equation equals:

$$\nabla_x\Phi^k_{\rho,\mu^k}(\hat{x}^l) = H\hat{x}^{(l)}+g+\rho(\hat{x}^{(l)}-x^k)+ C^\top\left[z^k+\tfrac{1}{\mu^k}(C\hat{x}^{(l)}-u)\right]_+.$$

For either low values of μ^k or ill-conditioning of $C_{I_k(\hat{x}^{(l)})}$, the evaluation of the gradient is numerically challenged due to the value of $\frac{1}{\mu^k}C^\top_{I_k(\hat{x}^{(l)})}C_{I_k(\hat{x}^{(l)})}$ (see, e.g., [28, Chapter 17.4] which gives examples illustrating ill-conditioning resulting from such structures).

E. Differences with other related AL-based methods

In this section, we highlight the key characteristics of PROXQP, focusing on its distinctions from other Augmented Lagrangian (AL)-based approaches, specifically QPALM [31] and QPDO [32]. The following differences will be further validated through experimental results in Section VI.

PROXQP is defined by several unique features:

- It employs a primal-dual augmented Lagrangian merit function $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ (see (9)). This specific choice of merit function notably affects the number of inner iterations performed.
- Both the step size, denoted as μ^k, and the accuracy parameter, ϵ^k, are jointly calibrated through a globalization technique known as the Bound-Constrained Lagrangian (BCL) strategy. It plays a critical role in determining the number of outer iterations.
- BCL functions as a filtering mechanism where multipliers z^k are selectively accepted based on predefined conditions (refer to the "if/else" decision process in Algorithm 1). This selective acceptance significantly impacts the trajectory of the iterates, differentiating PROXQP from other AL-based methods detailed hereafter.

Comparison with QPALM. QPALM, another AL-based method, leverages the Primal Augmented Lagrangian (PAL) as its merit function (see Φ^k_{ρ,μ^k} in Section II-D). In contrast to PROXQP, QPALM always accepts the multipliers in the outer loops, following the classical Primal Method of Multipliers (PMM) [23, Section 5]. Furthermore, QPALM employs a heuristic, component-wise calibration for the step size μ^k, assigning different values for each constraint. In PROXQP, however, a single μ^k is used, determined through the BCL filtration mechanism. Additionally, QPALM uses a constant geometric decay to calibrate ϵ^k, which contrasts with PROXQP's joint calibration strategy for μ^k and ϵ^k. These differences lead eventually to distinct iterative behaviors, as demonstrated by the experimental comparisons Section VI.

Comparison with QPDO. QPDO also employs an AL-based approach, using the Primal-Dual Augmented Lagrangian (PDAL) as its merit function (4). Although both methods share similar primal dual AL-based structures, PROXQP uses a merit function $\mathcal{M}^k_{\rho,\mu^k,\alpha}$ with a tunable parameter α (in QPDO, $\alpha=0.5$ is fixed, while PROXQP optimizes it for efficiency, typically choosing $\alpha=0.95$). A primary algorithmic difference lies in the outer loop: QPDO, like QPALM, always accepts the multipliers z^k, following the PMM strategy. Moreover, similarly to QPALM, QPDO uses component-wise heuristics for calibrating both μ^k, while PROXQP utilizes a unified calibration for μ^k and ϵ^k via BCL. The selective acceptance of multipliers in PROXQP, combined with its joint calibration of μ^k and ϵ^k,

leads to significantly different iterative patterns compared to QPDO.

In the next section, we present other state-of-the-art methods for solving QPs alongside their theoretical features that can be used in the context of embedded applications.

IV. Related work

Generic-constrained convex QPs are commonly solved with iterative (or indirect) methods. These methods are traditionally divided into two main families: (i) *active-set* methods, and (ii) *penalization methods*. When the problem contains no inequality constraint (i.e., unconstrained or only subject to equality constraints), direct methods can be used, as solving the corresponding QP can be done by directly solving the corresponding linear KKT system [28]. Iterative methods of both families typically aim at solving cascades of structurally simpler intermediary optimization problems, whose solutions eventually go to that of the original problem.

A. Active-set methods

This family of methods, first developed in the 50s, aims at determining the set of active constraints at an optimal point of (QP) (see, e.g., [28, Section 16.5] for an introduction). Once this set of active constraints is determined, the constrained QP can be solved directly by considering an equivalent QP subject to equality constraints only, whose solution matches that of the original QP. On the positive side, warm-start and hot-start strategies can easily be incorporated within active-set methods [40]. This is usually key for solving cascades of similar QPs, which is common in applications such as sequential quadratic programming (SQP) and model predictive control (MPC). They also usually perform well on small-sized QPs. Yet, active-set methods have limited scalability, struggle to exploit sparsity, and are hard to *early-stop* at inexact solutions satisfying (1) at accuracy ϵ_{abs} (they tend to stop only when a global solution is found, i.e., with $\epsilon_{\text{abs}} = 0$). Moreover, they may require strong assumptions, such as the QP being strongly convex or satisfying "linear independence constraint qualification" (LICQ) (see, e.g., [28, Definition 12.4]). These assumptions reduce their applicability and may cause robustness issues, such as "active-set cycling" [28, sections 13.5 and 16.5].

B. Penalization methods

This family of methods transforms the original constrained problem into a sequence of problems with either no constraints or very simple ones (such as sign constraints on some variables). The new problems typically have objectives that consist of two terms: (i) the objective of the original problem and (ii) a term for penalizing points not being feasible for the original problem. There are two very common types of penalization methods used in practice.

Interior-point methods. This family of penalization methods forces through a *barrier function* the sequence of intermediary optimization problems to have strictly feasible solutions with respect to the domain of the original problem; see, e.g., [28, Section 16.6] for an introduction. Primal-dual interior-point methods [41], [42] became popular in the 90s due to their good practical performances across a wide range of problems. On the positive side, unlike active-set methods, they benefit from early termination and can easily exploit the sparsity structure of the problem. Moreover, by construction, they control the primal-dual gap. Yet, because of the homotopy-based structure of this family of methods, one of their main drawbacks is the difficulty of using them with warm-starting procedures when solving a sequence of related QPs (e.g., within SQP solvers or MPC problems).

Augmented Lagrangian-based methods. This family of methods is primarily based on the idea of Lagrangian relaxations with an additional quadratic penalization term (possibly piecewise) for encouraging the feasibility of the iterate (see, e.g., [28, Section 17.3]). This kind of technique emerged in the 70s through the works of Hestenes and Powell [24], [25] and then later with those of Rockafellar [23], which tightly emphasized their link with the so-called "proximal-point method". Indeed, augmented Lagrangian-type methods naturally arise by applying proximal-point methods on either the dual or saddle-point formulations to (QP), thereby offering the advantage of converging under relatively weak assumptions. Augmented Lagrangian-type methods also have the advantage of exploiting the sparsity structure of the problem and can be easily warm-started via warm-start and hot-start procedures. They also benefit from early termination. Moreover, as shown in Remark 1 previously, augmented Lagrangian algorithms based on the method of multipliers inherit from the advantageous property of converging towards the closest feasible QP in ℓ_2 sense, if it happens to be primal infeasible [29]. However, AL-based methods might exhibit slow convergence behaviors similar to those of first-order optimization methods. For instance, when they are based on splitting methods such as the alternating direction method of multipliers (ADMM) [43, Section 4.4] they are relatively slow toward high accuracy levels. Furthermore, they might not necessarily have control over the primal-dual gap.

V. Software Implementation

In this section, we list a range of common QP solvers, along with their underlying optimization strategy, then detail the software implementation of ProxQP within the ProxSuite library.

A. Common QP solvers

We provide below a summary of different start-of-the-art solvers implementing iterative optimization methods previously described in Section IV, and summarized in Table I, with a short description of their practical features for embedded optimization.

Active-set solvers. Popular active set-based convex QP solvers include the open-source qpOASES [40], quadprog [44], DAQP [45], and the QPA module in the open source software GALAHAD [46].

TABLE I: Comparison of different optimization methods and software libraries for solving QPs.

Method	Solver	Backend	Warm-starting	Hot-starting	Early termination	Primal-dual gap	Infeasibility solving
ACTIVE-SET	QPOASES [40]	Dense	✓	✓	✗	✓	✗
	QUADPROG [44]	Dense	✗ [1]	✗ [1]	✗	✓	✗
	DAQP [45]	Dense	✗ [1]	✗ [1]	✗	✓	✗
INTERIOR-POINT	GUROBI [22]	Sparse	✗	✗	✓	✓	✗
	MOSEK [21]	Sparse	✗	✗	✓	✓	✗
	CVXOPT [51]	Dense	✗	✗	✓	✓	✗
	ECOS [50]	Sparse	✗	✗	✓	✓	✗
	QPSWIFT [52]	Sparse	✗	✗	✓	✓	✗
AUGMENTED LAGRANGIAN	OSQP [20]	Sparse	✓	✓	✓	✗	✗ [2]
	SCS [53]	Sparse	✓	✓	✓	✓	✗ [2]
	QPDO [32]	Sparse	✓	✗	✓	✗	✓
	QPALM [31]	Sparse	✓	✓	✓	✗	✓
	PROXQP [16]	Sparse & Dense	✓	✓	✓	✓	✓

[1] In their original implementation. [2] Not established nor implemented.

Interior-point solvers. Standard solvers using an interior-point strategy are commercial solvers GUROBI [22] and MOSEK [21], closed-source BPMPD [47], and open-source solvers OOQP [48], HPIPM [49], ECOS [50], CVXOPT [51] and QPSWIFT [52].

Augmented Lagrangian-based solvers. Common solvers based on augmented Lagrangians for solving QPs include OSQP [20] and SCS [53] (via an alternating direction method of multipliers), QPALM [31] and QPDO [32].

B. The PROXSUITE *library*

PROXQP is implemented in C++ within the PROXSUITE library and builds extensively on the Eigen library [54] for efficient linear algebra. The current implementation is tailored to both dense and sparse matrix operations and leverages recent CPU architectures equipped with advanced vectorization instructions. PROXSUITE is publicly available at https://github.com/Simple-Robotics/proxsuite under a permissive open source BSD-2-Clause license. It comes with out-of-the-box interfaces for the C++, Julia, and Python languages inspired from OSQP [20]. The documentation of PROXSUITE's API is available online at https://simple-robotics.github.io/proxsuite/. In the following, we describe some key features of PROXSUITE, list our default parameters in Table II as well as a code snippet in Listing 1.

Cholesky factorization. In PROXSUITE, we have developed dedicated dense and sparse LDLT Cholesky factorizations [55] to explicitly account for the specific calculations. In particular, this factorization implements advanced rank update routines to efficiently account for the change of active sets when solving (14) while lowering the overall memory footprint. Furthermore, when the number of constraints is too large compared to the number of (primal) variables, we add an option to factorize, using the dense backend, the matrix

$$H + \rho I + \tfrac{1}{\mu} C^\top_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})} C_{J_k(\hat{x}^{(l)},\hat{z}^{(l)})}, \tag{15}$$

instead of the one involved in (14) (the latter choice is called PRIMALDUALLDLT, whereas the first PRIMALLDLT). A heuristic (comparing "typical" algorithmic complexity for factorizing, updating, and solving (14) using one of the mentioned matrices) is also available for automatically choosing which matrix to factorize.

Moreover, a similar heuristic is available for the sparse backend when the system is large-scale. In such case, a sparse matrix-free routine then solves (15) in the same spirit as [31, Section 4.1.3]. The sparse and dense Cholesky factorizations are also freely available within the PROXSUITE library and will hopefully be integrated into the Eigen library soon.

TABLE II: Default hyperparameters for PROXQP.

Parameter	Value	Description
α	0.95	primal-dual interpolation
$\epsilon^0_{\text{bcl}} = \epsilon^0$	1	accuracy
ρ	10^{-6}	primal proximal penalization
$\mu_{\min}$	10^{-9}	minimal penalization
μ_f	0.1	penalization decay factor
α_{bcl}, β_{bcl}	0.09, 0.9	BCL hyperparameters
$k_{\max}$	10^4	safeguard
μ^0_e	10^{-3}	penalization for equality constraints
μ^0_i	10^{-1}	penalization for inequality constraints

Warm-start. Since PROXQP is built on an augmented Lagrangian-based method, the algorithm is readily warm-startable. For example, it is a useful feature for solving a cascade of QPs that change slightly (e.g., for Model Predictive Control, Inverse Kinematics, Inverse Dynamics, SQPs, etc.). Default initial guesses are also available as options. Among others, by default and motivated by the fact that equality constraints are always active at an optimal solution, we use the following initialization for primal and dual variables:

$$\begin{bmatrix} x^0 \\ z^0 \end{bmatrix} = \begin{bmatrix} H + \rho I & C^\top \\ C & -\mu^0 I \end{bmatrix}^{-1} \begin{bmatrix} -g \\ u \end{bmatrix},$$

provided C is an equality constraint matrix and u and equality constraint vector.

Hot-start. PROXQP has a dedicated feature, similar to the OSQP, SCS, and QPOASES solvers [20], [40], [53], where we can *hot-start* the solver with a factorization from a previous related problem. More precisely, if QP updates concern only linear terms (e.g., u or g), then the updates are realized

allocation-free in PROXQP. Furthermore, PROXQP has settings that combine warm-starts with a reuse of the whole previous data workspace (including the Cholesky factorization) called WARM_START_WITH_PREVIOUS_RESULT. This feature has been specially designed for solving control or SQP problems with changes concerning only linear parts.

Preconditioning. PROXQP contains several preconditioning strategies, enhancing the overall numerical stability and convergence of the optimization process. The default preconditioner used in our current implementation is often referred to as the Ruiz equilibration [56]; see, e.g., [20, Algorithm 2], also used in OSQP and QPALM.

Solving closest feasible problems. PROXQP can solve the closest feasible QP problem if it is detected to be primal infeasible (following the same procedure described as in OSQP [20, Section 3.4]). If the option is activated, then the following stopping condition will be tracked:

$$\begin{cases} \|Hx^\star + g + C^\top z^\star\|_\infty \leqslant \epsilon_{\text{abs}} \\ \|C^\top [Cx^\star - u]_+\|_\infty \leqslant \|C^\top 1\|_\infty \epsilon_{\text{abs}} \\ |(x^\star)^\top H x^\star + g^\top x^\star + (u + s^\star)^\top [z^\star]_+| \leqslant \epsilon_{\text{abs}} \end{cases}$$

Dealing with nonconvex QPs. Since PROXQP is based on a primal-dual proximal method of multipliers, it is also able to find local minima of nonconvex QPs, provided that the primal proximal step size ρ is larger than the lowest negative eigenvalue of the cost Hessian H [31, Theorem 2.6]. Following the methodology described in [31, Section 4.2], PROXQP also provides routines for evaluating this minimal eigenvalue (using a power iteration-like algorithm, or methods from the Eigen library) from the and automatically sets ρ to make the intermediary quadratic sub-problems well-defined and convex.

Solving batches of QPs. PROXSUITE includes a dedicated data structure for parallel solving of batches of dense or sparse QPs.

```python
import proxsuite.proxqp as solver
n = 10    # primal variable
n_eq = 2  # equality constraints
n_in = 2  # generic type of ineq. constraints

# Create a qp object and change settings
qp = solver.dense.QP(n, n_eq, n_in)
qp.settings.primal_infeasibility_solving = True
qp.settings.initial_guess = solver.
    WARM_START_WITH_PREVIOUS_RESULT
qp.init(...)  # fill with QP data
qp.solve()

# Optionally set specific backend
qp1 = solver.dense.QP(n, n_eq, n_in, solver.dense.
    DenseBackend.PrimalDualLDLT)
qp2 = solver.dense.QP(n, n_eq, n_in, solver.dense.
    DenseBackend.PrimalLDLT)
qp3 = solver.dense.QP(n, n_eq, n_in, solver.dense.
    DenseBackend.Automatic)

# Solving N QPs in parallel
qp_vector = solver.dense.VectorQP()
for i in range(N):
    qp = qp_vector.init_qp_in_place(n, n_eq, n_in)
    qp.init(...)  # fill with data of i-th QP
solver.dense.solve_in_parallel(n_threads, qp_vector)
```

Listing 1: Code example for using the PROXSUITE API.

VI. EXPERIMENTAL RESULTS

In this section, we evaluate PROXQP against other state-of-the-art approaches on classic control problems and representative robotic tasks. These practical problems illustrate how PROXQP takes advantage of structural properties (sparsity structure, warm-starts, hot-starts, early stopping). We also show that PROXQP's ability to solve primal infeasible problems can be leveraged in closed-loop MPC scenarios accounting for perturbations and measurement errors that yield infeasible problems. Finally, we conclude benchmarking PROXQP against alternative solvers of the state of the art on more generic types of QPs [57], [20].

A. Benchmark setup and scenarios

Our benchmarks focus on relatively small to medium-sized QPs, typically used for embedded applications such as in robotics (less than a thousand variables and constraints), with sparse or dense problem structures.

In the first set of experiments (see Section VI-B), we consider sparse convex MPC scenarios: the benchmark proposed by [20, Section 8.4] and the one proposed by [58, Section 5.A]. We also explain with these synthetic examples how primal infeasibility solving can be used in the context of closed-loop MPC : (i) first, we initialize one instance with a primal infeasible point with respect to the dynamics bounds $x_{\min}$ and $x_{\max}$; (ii) in the latter, we add an infeasible random perturbation to the dynamics at a fourth of the simulation.

In the second set of experiments, we focus on dense problems (see Section VI-C): inverse kinematics tasks, a closed-loop dense MPC tasks for controlling the center of mass of a humanoid robot, and a real-world closed-loop MPC scenario with the Upkie wheeled-biped robot [59]. Through an extensive benchmark with the humanoid closed-loop MPC task, we also highlight that PROXQP is more robust than other state-of-the-art solvers against random perturbations that render the approximate solution infeasible.

Finally, we evaluate PROXQP on a range of standard quadratic programming (QP) problems commonly used in the optimization literature. In particular, we perform dedicated benchmarks to compare PROXQP with the QPALM and QPDO approaches. To ensure a fair comparison, we also include a variant called PROXQP-05, where the parameter α is set to 0.5, as in QPDO. This helps isolate the effect of the globalization strategy (BCL) used in PROXQP from the one used in QPDO. These benchmarks assess the impact of PROXQP's key features by comparing the total number of iterations and overall runtime.

For all experiments, PROXQP is compared against other state-of-the-art approaches: active-set methods (QPOASES, QUADPROG), interior-point methods (MOSEK, QPSWIFT) and augmented Lagrangian-based methods (SCS, OSQP, QPALM and QPDO[3]). We exploit all available features (see Table I) of each solver considering the problem structure. More precisely,

[3]The QPDO solver lacks both C++ and Python interfaces. Therefore, for benchmarking purposes, we re-implemented the algorithm in C++ within the ProxSuite framework, utilizing PROXQP's dense backend subroutines. The author of QPDO has been notified of this implementation.

TABLE III: Sparse convex MPC benchmark (total runtimes in ms for solving 100 simulation steps)

Problem type	Dimensions	PROXQP	QUADPROG	OSQP	QPOASES	SCS	QPSWIFT	MOSEK
CONVEX MPC $\epsilon_{\text{abs}} = 10^{-3}$	n=76,m=142	**5.7±0.2**	✗[2]	6.7±0.7	37.6±23.9	11.9±2.2	13.5±0.5	82.1±3.5[1]
	n=162,m=294	**16.4±1.7**	✗[2]	24.1±6.0	393.9±109.9	51.0±15.7	47.1±4.6	211.5±13.3
	n=216,m=392	**27.6±4.8**	✗[2]	48.2±29.4	1434.2±480	65.6±30.9	80.0±3.6	311.3±15.4[1]
	n=270,m=490	**43.2±8.3**	✗[2]	77.5±65.6	2345.5±608	104.7±66.0	131.0±7.4	454.2±18.6
CONVEX MPC $\epsilon_{\text{abs}} = 10^{-6}$	n=76,m=142	**8.5±1.4**	✗[2]	14.1±8.4	27.1±15.4	25.5±14.6	16.5±0.7	84.0±3.8[1]
	n=162,m=294	**21.1±2.3**	✗[2]	94.0±101.9	452.9±98.3	101.8±47.9	57.8±5.3	216.3±14.1[1]
	n=216,m=392	**36.1±4.3**	✗[2]	150.1±136.0	1626.6±650.1	162.0±95.0	99.5±4.9	314.1±17.4[1]
	n=270,m=490	**57.0±9.0**	✗[2]	346.7±362.0	2670.3±842.4	311.1±122.5	164.1±9.9	459.6±21.0[1]
CHAIN OF MASS $\epsilon_{\text{abs}} = 10^{-3}$	n=462,m=834	**45.2±0.8**	(28.3±0.6)E3	59.1±1.7	(80.2±1.6)E3	161.1±11.1	215.3±1.3	566.8±14.1
CHAIN OF MASS $\epsilon_{\text{abs}} = 10^{-6}$	n=462,m=834	**67.8±11.1**	(28.3±0.6)E3	104.9±9.6	(80.2±1.6)E3	172.6±9.8	248.9±3.1	575.5±16.8[1]

[1] The solution does not satisfy the desired absolute accuracy. [2] QUADPROG cannot solve QPs that are not strictly convex.

OSQP, QPOASES, and SCS use warm-starting and hot-starting when possible. Box constraints on the state or control variables are specified when the solver API allows to do so. Otherwise, they are listed as parts of the inequality constraints. Moreover, interior-point and augmented Lagrangian-based methods exploit their underlying early-stopping strategy if it suits the task. The experiments were conducted on a standard laptop CPU, an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. The benchmark is open-source and available at https://github.com/Bambade/proxqp_benchmark with an easy-to-use interface strongly inspired by the one proposed by OSQP [20].

B. Sparse problems

In this subsection, we consider two sets of convex MPC problems: an existing convex-MPC benchmark from [20, Section 8.4], and a sparse chain of oscillating masses from [58, Section 5A]. In both cases, we run systems over 100 simulation steps. Their initial states x_{init} are uniformly chosen between reasonable bounds. We thus measure the average of the full-time needed for solving all 100 simulation steps. For each separate randomization seed, the full simulation steps' is also averaged over 10 consecutive runs in order to mitigate the impact of loading costs. Furthermore, we consider two different scenarios: (i) one when early-stopping is relevant ($\epsilon_{\text{abs}} = 10^{-3}$); and another (ii) where higher accuracy is required ($\epsilon_{\text{abs}} = 10^{-6}$). In this test set, only the vectors terms in (QP) change in this test case, the practitioner can use warm-start and hot-start strategies if available. These features are exploited by OSQP, QPOASES, SCS and PROXQP. Furthermore, inequality constraints in these problems only consist of box constraints, a specificity that SCS leverages with a custom feature. Finally, since the problems have a sparse structure, PROXQP uses its sparse back-end.

1) Convex MPC test set

Experimental setup. In this set of experiments, we consider the MPC benchmark proposed in [20, Section 8.4]. This benchmark consists of LQR problems with additional state and input constraints, as described in [20, Appendix A3], with a horizon $T = 10$ starting from $x_0 = x_{\text{init}} \sim \mathcal{U}(-0.5\bar{x}, 0.5\bar{x})$ with $\bar{x} \sim \mathcal{U}(1, 2)$:

$$\begin{aligned} \min_{x_t \in \mathbb{R}^{n_x}, u_t \in \mathbb{R}^{n_u}} \; & x_T^\top Q_T x_T + \sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t, \\ \text{s.t. } & x_{t+1} = A x_t + B u_t, \\ & -\bar{x} \leqslant x_t \leqslant \bar{x}, \; -\bar{u} \leqslant u_t \leqslant \bar{u}, \end{aligned} \tag{16}$$

with $Q, Q_T \in \mathbb{S}_+(\mathbb{R}^{n_x})$, $R \in \mathbb{S}_{++}(\mathbb{R}^{n_u})$, and $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$. More precisely, we solve this optimal control problem in a receding horizon fashion, computing an optimal input sequence $u_0, \dots, u_{T-1}$, applying the first input u_0 to the system and propagating the state to the next time step. The whole procedure is repeated 100 times. We choose the state variable dimension n_x and control input dimension n_u for the context of small systems embedded systems, concretely $n_x = 6, 12, 16, 20$ and $n_u = n_x/4$. This benchmark generates sparse problems (i.e., the KKT matrix of the QP has about 5% of nonzeros elements) and QPs whose dimensions (number of primal variables and constraints) range from 76 to 490.

Runtime results. The first and second-row blocks of Table III show the results of this test case. We can see that in both scenarios PROXQP is the fasted method with a speed-up ranging from 1.2 to 1.8 in the early-stopping scenario (with respect to OSQP, the second fastest approach); and from 1.7 to 2.9 in the high accuracy scenario. We can also observe that sparse solvers (PROXQP, OSQP, SCS, QPSWIFT, MOSEK) obviously provide better timings than dense solver (QPOASES). Furthermore, one can notice that methods based on ADMM (i.e., OSQP and SCS) are slower to converge to high-accuracy solutions.

Solving MPC from primal infeasible points. In order to solve an MPC problem starting from $x_{\text{init}} \sim \bar{x} + \mathcal{U}(0, 0.01)$, we consider the following strategy: the solution found at the first simulation step provides a solution trajectory $\hat{x}_t, \hat{u}_t$ (over a horizon of T time steps for control inputs $\hat{u}_t$ and $T + 1$ for the state $\hat{x}_t$) for the closest feasible QP of (16) (see orange line in Figure 1). PROXQP also provides optimal shifts $\hat{s}_t^u$, $\hat{s}_t^x$ for the inequality constraints on x and u; and $\hat{s}_t^e$ for equality constraints. We make the forward updates as follows:

- $u_0 = \hat{u}_0 - \hat{s}_0^u$: we bring back the closest feasible input solution to the solution space of the initial QP (so u_0 satisfies the input constraints since $\hat{u}_0 \leqslant \bar{u} + \hat{s}_0^u$ by construction);
- $x_1 = A\hat{x}_0 + Bu_0 - \hat{s}_0^e$: we update the trajectory from a feasible control input u_0 and optimal dynamic $\hat{x}_0$ and shift $\hat{s}_0^e$. It lies on the green line drawn in Figure 1. Compared to the closest feasible optimal solution $\hat{x}_1 = A\hat{x}_0 + B\hat{u}_0 - \hat{s}_0^e$, we make a drift

$$\begin{aligned} x_1 - \hat{x}_1 &= A\hat{x}_0 + Bu_0 - \hat{s}_0^e - [A\hat{x}_0 + B\hat{u}_0 - \hat{s}_0^e], \\ &= B(u_0 - \hat{u}_0) = -B\hat{s}_0^u, \end{aligned}$$

  which tends towards zero, as soon as the optimal shift over the control input tends towards zero.
- We repeat the closed-loop process starting from x_1 to derive $\hat{x}_1, \hat{u}_1$ and then $x_2 = A\hat{x}_1 + B(\hat{u}_1 - \hat{s}_1^u) - \hat{s}_1^e$.

We illustrate the application of this update rule with an instance of the convex MPC scenario for $n_x = 6$ and $n_u = 1$ in Figure 2. Figure 2b shows the control trajectory u_t of the system. As expected, it lies in the control bounds $-\bar{u}$ and $\bar{u}$. Figure 2a shows the 6 components of the dynamics x_t. All these components are initially infeasible for a fixed number of simulation steps since they are above $\bar{x}$. Since the initial infeasible perturbation is not too large (i.e., the optimal shifts are sufficiently small), the dynamics eventually converge to the feasible steady-state position.

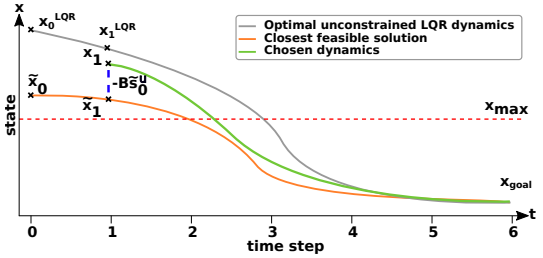
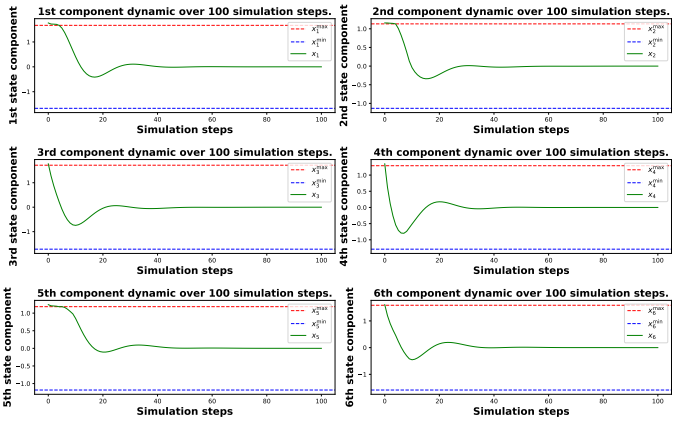


Fig. 1: A convex MPC problem is initialized with an infeasible state $x_0 = x_{\text{init}} > x_{\max}$. **Grey:** Yet, $x_0^{\text{LQR}} = x_0$ enables reaching a horizon T a feasible state x_{goal} with an unconstrained LQR scheme. **Orange:** It guarantees that a closest feasible trajectory $\hat{x}_t$ exists with optimal shifts $\hat{s}_t^x, \hat{s}_t^u, \hat{s}_t^e$. **Green:** We choose the update $x_1 \overset{\text{def}}{=} \hat{x}_1 - B\hat{s}_0^u$, which is guaranteed not moving away x_1 from x_{goal} for sufficiently small optimal shifts $\hat{s}_1^x, \hat{s}_0^u, \hat{s}_1^e$. More precisely, $\|x_1 - x_{\text{goal}}\|_2 \leqslant \|x_1^{\text{LQR}} - x_{\text{goal}}\|_2$ and $\|x_1 - x_{\max}\|_2 \leqslant \|x_1^{\text{LQR}} - x_{\text{goal}}\|_2$. (Appendix-B) provides more technical details about the properties of this update rule.

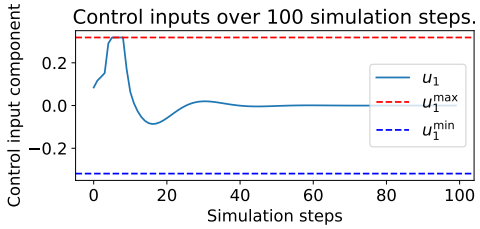
2) Sparse chain of oscillating masses

Experiment setup. We consider the following MPC problem involving a chain of six masses interconnected with spring-dampers [58, Section 5A]

$$\begin{aligned} &\min_{x_t \in \mathbb{R}^{n_x}, u_t \in \mathbb{R}^{n_u}} \frac{1}{2} x_T^\top Q_T x_T + \frac{1}{2} \sum_{t=0}^{T-1} x_t^\top Q x_t + u_t^\top R u_t, \\ &\text{s.t. } x_{t+1} = Ax_t + Bu_t,\ x_0 = x_{\text{init}} \sim \mathcal{U}(-0.5, 0.5), \\ &-4 \leqslant x_t \leqslant 4,\ -0.5 \leqslant u_t \leqslant 0.5, \end{aligned}$$



(a) 6 components of the dynamics x_t over 100 simulating steps.



(b) Control input u_t over 100 simulating steps.

Fig. 2: Closed-loop convex MPC simulation steps with $n_x = 6$ and $n_u = 1$ starting from an infeasible initial state x_0 using ProxQP. At each new simulation step t, the new control input equals $u_t = \hat{u}_t - \hat{s}_t^u$, so it is guaranteed to be feasible, while the dynamics follow $x_t = \hat{x}_t - B\hat{s}_t^u$. Since the initial infeasible perturbation is not too large, the dynamics converge after a fixed number of steps to a feasible regime.

with $R = I$ and $Q = I$. The problem dimensions are $n_x = 12, n_u = 3$ and the horizon is $T = 30$ [58, Section 5A]. It generates QPs with 462 variables and 864 constraints.

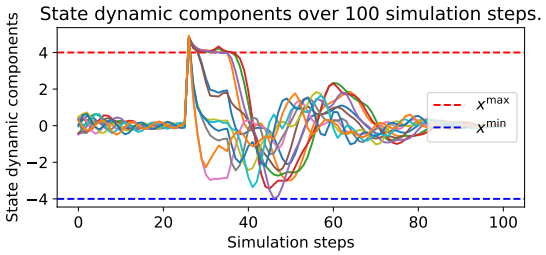
Runtime results. The last two-row blocks of Table III show the runtime results. We can see that PROXQP is the fastest method with a speed-up ranging from 1.3 (early stopping scenario) to 1.5 (high accuracy scenario).

Solving MPC with infeasible perturbations. To illustrate another possible scenario where primal infeasibility solving might be useful, we consider the following setup: at bout one fourth of the simulation (i.e., the 25$^{\text{th}}$ simulation step), a random perturbation $w_t \sim 4.8 + \mathcal{U}(0, 0.01)$ disturbs the usual dynamic update

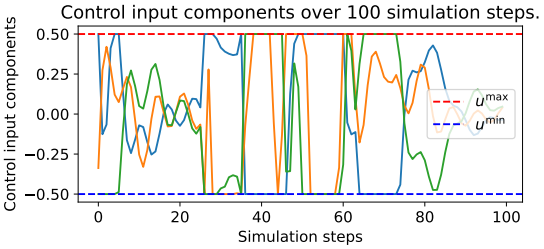
$$x_{t+1} = Ax_t + Bu_t + w_t,$$

which is thus guaranteed at the next LQR simulation step to create an infeasible problem since x_t is already close to the steady state for such a system after 25 steps. We apply the same update rule as exposed in the previous MPC problem to tackle this infeasible perturbation since the problem structure is similar. Figure 3a shows the 12 components of the chain system. It becomes infeasible after the 25$^{\text{th}}$ closed-loop update. Nevertheless, we can see that it eventually recovers to the steady state solution after a finite number of time steps. Figure 3b

shows the 3 control input components. We can see that, as expected, they all satisfy the constraint bounds thanks to the update rule $u_t = \hat{u}_t - \hat{s}_t^u$.



(a) 12 components of the chain dynamics over 100 simulation steps.



(b) 3 components of the inputs u_t over 100 simulation steps.

Fig. 3: Convex MPC problem for controlling a chain of masses with an infeasible dynamic perturbation at the fourth of the simulation. The control inputs and the dynamics are updated following paragraph VI-B1c. After a fixed number of time steps, the system manages to recover its asymptotic steady state while satisfying during the whole process feasible control bounds.

C. Dense problems

1) Inverse kinematics

In this set of experiments, we consider different inverse kinematic (IK) tasks proposed for motion control of articulated robots using Pinocchio [60] and the open-source Pink library https://github.com/stephane-caron/pink. The resulting motion controller is run for 5 s (i.e., solving $1,000$ consecutive QPs with a time-step of 5 ms). Each such run is repeated 50 times, and we measure the runtime per timestep average for achieving each IK task. Considering the structure of the problem, OSQP, SCS, QPOASES, and PROXQP benefit from warm-starting, and PROXQP uses its dense-backend. Furthermore, from a practitioner's point of view, we consider the minimal accuracy needed ϵ_{abs} for achieving these tasks (i.e., solving each IK task while maintaining sufficient smoothness in the resulting motions). We thus imposed $\epsilon_{\text{abs}} = 10^{-5}$ for each task, except when using STRETCH which required a lower $\epsilon_{\text{abs}} = 10^{-7}$ in order to satisfy to be numerically stable while enforcing joint limits.

Table IV reports runtimes in its second-row block. It shows that PROXQP is the fastest method with a speed-up ranging from 1.2 to 1.8 with respect to OSQP or QUADPROG, the next two fastest approaches. Apart from OSQP, we note how dense solvers are faster than sparse ones, which can be explained by the fact that IK problems depict a relatively high ratio of nonzero elements with respect to the problem size (the ratios of sparsity of the Hessian cost matrix H ranges from 38% to 100%, and between 4% and 16% for the constraint matrix).

2) Chain of oscillating masses

In this set of experiments, we consider a dense version of the chain of oscillating masses MPC problem proposed by [40, Section 6.3] as an SQP problem. The first test problem CHAIN80 aims at regulating a chain of nine masses connected by springs into a certain steady state. One end of the chain is fixed on a wall while the three velocity components of the other end are used as control input with fixed lower and upper bounds. The prediction horizon of 16 seconds is divided into 80 control intervals. The model equations are derived from the linearisation of the non-linear ODE model (with 57 states) at the steady state. Deviation from the steady state, the velocities of all masses, and the control action are penalized via the objective function. It forms a sequence of 101 QPs with 240 variables with only box constraints over the input variables. The second test problem CHAIN80W considers the same settings by adding also state constraints into the optimization problem in order to ensure that the chain does not hit a vertical wall close to the steady state. It results thus in a sequence of 101 QPs with 240 variables and 709 constraints. These two SQPs are run 10 times, and we measure the total runtime necessary for solving these tasks to an accuracy $\epsilon_{\text{abs}} = 10^{-6}$.

We can see the runtime results in the first-row block of Table IV. It shows that PROXQP is the fastest method with a speed-up ranging from 1.8 to 11.6 with respect to QPOASES, the second fastest approach. We can notice again that dense solvers are faster than the other sparse solvers.

3) Model predictive control for humanoid locomotion

In this last set of experiments, we consider dense convex MPC problems for controlling a walking humanoid robot in simulations, and the real robot Upkie [59] in a balancing scenario. We evaluate again the robustness of the solvers to perturbations.

Humanoid locomotion via cart-table model. The task consists in controlling a reduced model of a humanoid robot, consisting of its center-of-mass c and zero-tilting moment point (ZMP) z, using the center-of-mass jerk $u = \dddot{c}$ as control input [61], [62]. Figure 4 depicts a trajectory of this system. The goal is to keep the ZMP inside the support polygon of the feet while achieving a prescribed walking velocity. The resulting MPC problem can be cast as a quadratic program [6] of the form:

$$\begin{aligned} \min_{\substack{x_t \in \mathbb{R}^{n_x} \\ u_t \in \mathbb{R}^{n_u}}} \; & w_T \|x_T - x_{\text{goal}}\|_2^2 + \sum_{t=0}^{T-1} w_x \|x_t - x_{\text{goal}}\|_2^2 + w_u \|u_t\|_2^2, \\ \text{s.t. } & x_{t+1} = Ax_t + Bu_t, \; x_0 = x_{\text{init}}, \\ & Cx_t + Du_t \leqslant e_t, \end{aligned} \tag{17}$$

where $n_x = 3$ (for center-of-mass position c, velocity $\dot{c}$ and acceleration $\ddot{c}$), $n_u = 1$ and $T = 16$. As in [6], horizontal coordinates can be decoupled, so that we can focus on single-dimensional coordinates $c, z \in \mathbb{R}$. For the sake of this example

TABLE IV: Dense QP benchmarks (average runtime per time-step (IK) and total simulation runtimes).

Task name	PROXQP	QUADPROG	OSQP	QPOASES	SCS	QPSWIFT	MOSEK
UR3 IK	**13.2±0.1μs**	17.0±3.3μs	15.8±0.2μs	21.7±0.3μs	✗ [1]	52.2±1.2μs	323.1±74μs [2]
UR5 IK	**11.9±0.1μs**	16.8±0.2μs	16.5±0.2μs	21.4±0.2μs	✗ [1]	53.8±0.8μs	310.8±5.3μs [2]
DUAL ARMS IK	**17.2±0.1μs**	23.3±0.2μs	40.4±0.6μs	330.4±5.8μs	81.5±0.6μs	152.1±1.1μs	554.4±25.1μs [2]
KINOVAGEN2 IK	**15.8±0.1μs**	18.9±0.2μs	17.0±0.2μs	31.4±0.4μs	46.5±1.3μs	53.7±0.4μs	375.4±14.9μs [2]
SIGMABAN IK	**14.1±0.2μs**	25.2±0.4μs	45.2±0.9μs	523.6±4.8μs	68.6±0.5μs	224.9±2.0μs	452.5±8.8μs [2]
STRETCH IK	**26.9±0.3μs**	36.7±0.6μs	53.1±0.9μs	212.6±0.8μs	455.3±23.8μs [2]	152.8±1.5μs	✗ [3]
CHAIN80 SQP	**15.6±0.3ms**	355.8±0.9ms	456.5±2.6ms	182±2.9ms	837±16.0ms	2193.9±22.3ms	1554.6±19.5ms [2]
CHAIN80W SQP	**265.7±2.9ms**	610.1±5.7ms	2141.9±22.4ms	467.4±3.3ms [1]	✗ [4]	✗ [4]	3444.2±30.3ms [2]
HUMANOID MPC $\epsilon_{\text{ABS}} = 10^{-2}$	**1.6±0.01ms**	4.5±1.8ms	1.8±0.01ms	18.0±5.3ms	433.0±21.7ms[5]	✗[3]	70.7±0.7ms
HUMANOID MPC $\epsilon_{\text{ABS}} = 10^{-4}$	**2.6±0.02ms**	4.5±1.8ms	2.7±0.03ms	18.0±5.3ms	80.2±2.9ms[5]	✗[3]	68.3±0.3ms
HUMANOID MPC $\epsilon_{\text{ABS}} = 10^{-6}$	**4.4±0.05ms**	4.5±1.8ms	4.4±0.05ms	18.0±5.3ms	64.3±2.4ms[5]	✗[3]	69.7±0.8ms

[1] The solver throws a factorization error (of non-convexity). [2] The solution does not match the desired accuracy
[3] The solver does not manage to satisfy configuration limits. [4] The solver does not manage to handle upper bound constraints and outputs infeasibility errors.
[5] Low accuracy iterates provokes with SCS warm starts more difficult QPs to solve in a closed-loop strategy.

TABLE V: Humanoid locomotion MPC problems with perturbations.

Noise Level	PROXQP	QUADPROG	OSQP	QPOASES	SCS	QPSWIFT	MOSEK
10.0	**11.9±9.7%**	1.0±0.2%	1.9±0.9%	1.0±0.2%	1.0±0.2%	0±0.0%	1.0±0.2%
5.0	**58.38±36.4%**	1.1±0.3%	2.1±1.1%	1.1±0.3%	1.1±0.4%	0±0.0%	1.1±0.3%
1.0	**100±0.0%**	1.4±0.8%	3.5±2.4%	1.4±0.8%	1.5±1.1%	0±0.0%	1.4±0.9%
0.5	**100±0.0%**	1.8±1.2%	5.5±3.8%	1.9±1.5%	2.1±1.6%	0±0.0%	1.8±1.2%
0.1	**100±0.0%**	3.3±2.6%	51.6±36.7%	4.3±3.8%	4.9±4.3%	0±0.0%	3.3±2.6%
0.05	**100±0.0%**	3.5±3.2%	97.6±13.5%	5.0±6.9%	7.7±6.5%	0±0.0%	3.5±3.2
0.01	**100±0.0%**	4.4±4.4%	**100±0.0%**	7.7±9.8%	60.2±37.8%	0±0.0%	4.4±4.5%
10^{-3}	**100±0.0%**	5.0±5.2%	**100±0.0%**	11.4±12.5%	**100±0.0%**	0±0.0%	5.0±5.2%
10^{-4}	**100±0.0%**	5.0±5.2%	**100±0.0%**	15.5±16.8%	**100±0.0%**	0±0.0%	5.0±5.2%
10^{-5}	**100±0.0%**	5.0±5.2%	**100±0.0%**	83.0±36.5%	99.1±8.9%	0±0.0%	5.1±5.3%
10^{-7}	**100±0.0%**	5.0±5.2%	**100±0.0%**	**100±0.0%**	97±14.8%	0±0.0%	44.8±34.2%
10^{-9}	**100±0.0%**	5.0±5.2%	**100±0.0%**	**100±0.0%**	**100±0.0%**	0±0.0%	**100±0.0%**
0.0	**100±0.0%**	**100±0.0%**	**100±0.0%**	**100±0.0%**	**100±0.0%**	0±0.0%	**100±0.0%**

we choose lateral coordinates and a target velocity of zero corresponding to walking in place.

The problem is solved with respect to the stacked vector $U = [u_0 \dots u_{T-1}]$ of control inputs, deriving states x_t linearly from U by recursive expansion of the dynamics $x_{t+1} = Ax_t + Bu_t$ [63]. This leads to a series of dense QPs with 16 variables and inequality constraints. The experiment is run for 100 simulation steps with a duration of $0.1s$ each, and the results are averaged over 100 trials. We measure the average total runtime for solving the task. Since the system is time-invariant, OSQP, SCS, QPOASES, and PROXQP benefit from warm-starting and hot-starting. In our experiments, setting $\epsilon_{\text{abs}} = 10^{-2}$ is sufficient to solve the task for all solvers based on IP and AL-based methods. Yet, they do not benefit the same way from this early-stopping feature. For this reason, we execute the runtime benchmark for $\epsilon_{\text{abs}} = 10^{-2}$, $\epsilon_{\text{abs}} = 10^{-4}$ and $\epsilon_{\text{abs}} = 10^{-6}$.

The runtime results are displayed in the last row block of Table IV. PROXQP is slightly faster than OSQP or QUADPROG, the best performing state-of-the-art solvers on this test set, where we note again how, as expected, dense solvers are generally faster than sparse solvers. Finally, we can see that SCS benefits less from early stopping here. Timings for SCS improve when the iterates are more accurate.

Robustness to perturbations. We apply random (Gaussian) noise at each simulation step to the center-of-mass acceleration with an amplitude σ modeling state observation inaccuracies

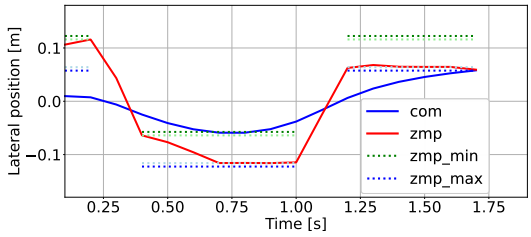


Fig. 4: Controlling the lateral center-of-mass trajectory (blue) to maintain the ZMP (red) within the real support polygon (dotted dark green and blue). Light dotted lines are more conservative ZMP bounds used during solving QP. Random perturbation of the current center-of-mass acceleration can lead to a ZMP outside the support polygon and a primal infeasible QP.

or unmodeled dynamics. When the ZMP is close to the edge of its support polygon, a small perturbation in center-of-mass acceleration can cause it to cross the edge, which would break the foot contact. For this reason, it is common practice to add safety margins to ZMP support areas in real systems, sometimes reducing them to a square well within the full area [64]. We include such margins in our QP formulation, as depicted by the light-dotted lines in Figure 4. In order to quantify the robustness of the different solvers to various noise amplitudes, we measure the percentage of the full trajectory (100 simulation steps) each solver manages to solve and average over 100 seeds. This closed-loop task is interrupted if the primal feasibility

violation with respect to the original support polygon (dark dotted lines) exceeds the accuracy of $\epsilon_{\text{abs}} = 10^{-2}$, or if the solver does not return any solution. For all solvers, we set the primal infeasibility tolerance to the minimum possible value in order to solve as many problems as possible. In Table V, PROXQP solves the entire trajectory for noise amplitudes lower than 1.0 m/s^2, while the second best solver OSQP reaches this level of performance for a noise amplitude of 10^{-2} m/s^2. All other solvers that are based on interior-point or active-set methods exhibit a less robust behavior on this task.

4) Closed-loop MPC on a real robot

We test PROXQP on real hardware with closed-loop model predictive control experiments on an Upkie wheeled biped [59]. The controller runs entirely on the robot's embedded computer, a Raspberry Pi 4 Model B equipped with a quad-core ARM Cortex-A72 64-bit SoC @ 1.8 GHz. Note that this CPU has fewer cores and a lower frequency than the standard laptop CPU used in other test cases. We formulate the MPC problem as a quadratic program using the same strategy as in (17), for a discretization of the linearized cart-pole model (for wheels) rather than cart-table (for flat feet). States observed from IMU Kalman filtering and wheel odometry are fed directly as an initial state x_{init} to the MPC problem. We set $T = 50$ timesteps with a time discretization of 20 ms, resulting in a receding horizon duration of 1 s. Task weights are chosen as $w_T = 1$, $w_x = 10^{-3}$ and $w_u = 10^{-3}$. The resulting QPs have dimension $n = 50$ with $m = 100$ inequality constraints. They are solved to precision $\epsilon_{\text{abs}} = 10^{-5}$ by PROXQP with hot-starting in 0.8 ± 0.02 ms, fitting well within the budget of a 200 Hz control loop. Figure 5 reports the resulting computation times alongside observed states for a sample run of the balancing controller on the robot.

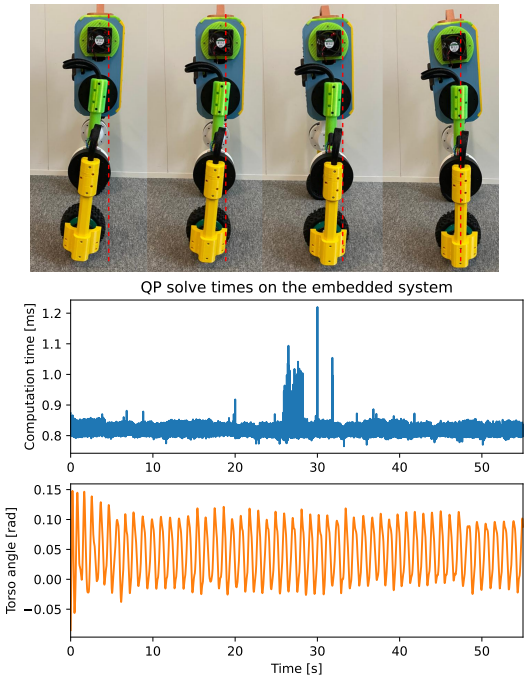


Fig. 5: Closed-loop model predictive control with PROXQP on the Upkie wheeled biped. The top plot shows the time taken by the hot-started solver to solve MPC problems at each control cycle on the robot's embedded computer. The bottom plot shows the angle between the robot's torso and the vertical at the corresponding times, illustrating how it balances upright.

Comparison to HPIPM and QPALM. For a relative comparison, we also evaluate the performance of solving these closed-loop MPC problems with HPIPM, PROXQP and QPALM. All three solvers are installed on a laptop computer (CPU: 12th Gen Intel(R) Core(TM) i7-12800H) and given the same absolute tolerance $\epsilon_{\text{abs}} = 10^{-5}$. All solvers successfully balance the robot in simulation. Computation times ranged from 0.13 ± 0.03 ms for PROXQP and 0.14 ± 0.03 ms for HPIPM[4] (no statistically-significant difference between the two) to 0.33 ± 0.06 ms for QPALM, where timing statistics were computed over $12,000$ solver calls corresponding to one minute of motion.

The source code to reproduce this experiment, in simulation or on a real Upkie robot, is available in the following repository: https://github.com/stephane-caron/proxqp_balancer.

D. Generic QPs

1) Random QPs

We generated random QPs with 15% of sparsity for both matrices H and C in the spirit of the benchmark API from OSQP [16], [20], [31]. The dimension d of those problems is in the range from 10 to 1000. For each dimension, we generated 1000 problem instances with different random seeds and averaged the runtime of each algorithm on 10 consecutive runs to limit the impact of loading costs. We then report the failure rate and the *shifted geometric means* [16, Section VI-Db] in Table VI. PROXQP is about 2.6 times faster across all the problems than OSQP, the second fastest approach. Furthermore, PROXQP, OSQP, and QPOASES exhibit no failure rate.

Comparison with QPDO, QPALM and variant ProxQP-05. Figure 6a shows that PROXQP requires fewer iterations than both QPALM and QPDO, especially as problem size increases—achieving up to a two- to fourfold reduction compared to the other methods. Compared to PROXQP-05 ($\alpha = 0.5$), PROXQP performs slightly better in terms of iterations. These gains carry over to execution times as well: as shown in Figure 6b, PROXQP is nearly ten times faster than QPALM and QPDO on problems of size 1000.

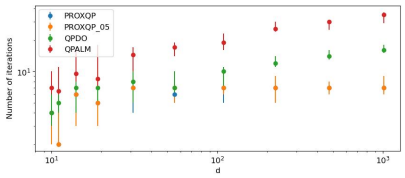
2) Maros-Mészáros problems

The Maros-Mészáros test set [57] is composed of 138 "hard" QPs. Most of them are sparse and ill-conditioned problems, and they contain up to 90597 variables and 180895 constraints. About 83% of the problems have a sparsity level lower than 10% for the Hessians H, as well as for the constraint matrices. We restrict the benchmark to problems whose dimensions

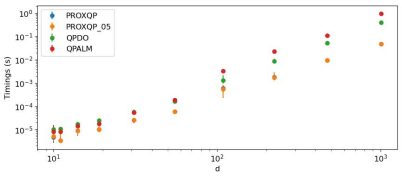
[4] HPIPM implements four "modes" detailed in section 4.2 of [49]. We only report `speed_abs` as it performed the fastest on this test.

TABLE VI: Shifted geometric means (SGM) and failure rates (FR) for solving sparse QPs. The lower the better.

	PROXQP	QUADPROG	OSQP	QPOASES	SCS	QPSWIFT	MOSEK
SGM	**1**	18.8	2.6	36.7	9.0	261.2	3279.8
FR	0.0%	0.02%	0.0%	0.0%	0.02%	3.31%	25.38%



(a) Iterations comparison.



(b) Solving time comparison.

Fig. 6: Sparse inequality-constrained QPs with increasing dimensions. The total number of iterations for both QPALM and QPDO grows similarly and at a faster rate compared to PROXQP and PROXQP-05 ($\alpha = 0.5$), leading to a corresponding increase in total runtime.

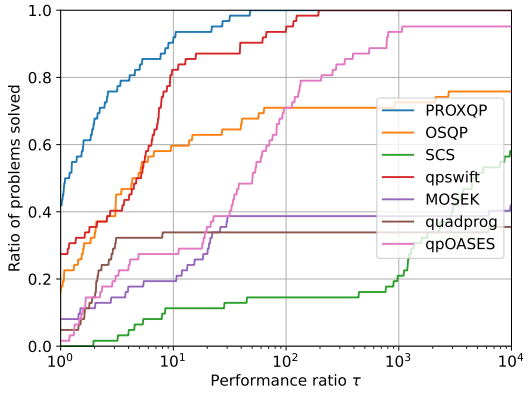


Fig. 7: Performance profiles on small to medium-sized Maros-Mészáros problems, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. Target accuracy $\epsilon_{abs} = 10^{-6}$ and time limit set to 1000 seconds. The higher the better.

TABLE VII: Shifted geometric means (SGM) based on iteration and runtimes, and failure rates (FR) for solving Maros-Mészáros problems for different target accuracy levels. The lower the better.

	ϵ_{abs}	PROXQP	PROXQP-0.5	QPALM	QPDO
SGM (iter)	10^{-9}	**1**	1.03	9.04	2.32
SGM (time)	10^{-9}	**1**	1.01	13.24	4.13
FR	10^{-9}	4.84%	4.84%	32.26%	14.52%
SGM (iter)	10^{-8}	**1**	1.03	3.85	2.72
SGM (time)	10^{-8}	**1**	1.02	12.45	12.20
FR	10^{-8}	1.61%	1.61%	16.13%	14.52%
SGM (iter)	10^{-7}	**1**	1.04	2.61	3.03
SGM (time)	10^{-7}	**1**	1.11	53.0	90.11
FR	10^{-7}	0.0%	0.0%	9.7%	12.90%
SGM (iter)	10^{-6}	**1**	1.04	3.22	2.91
SGM (time)	10^{-6}	**1**	1.03	74.08	83.26
FR	10^{-6}	0.0%	0.0%	1.61%	12.90%
SGM (iter)	10^{-5}	**1**	1.03	1.46	2.84
SGM (time)	10^{-5}	**1**	1.13	7.88	89.01
FR	10^{-5}	0.0%	0.0%	1.61%	12.90%

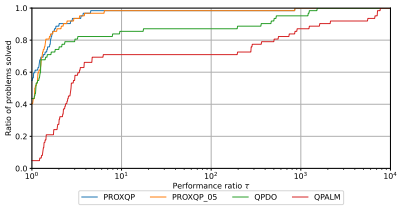
(constraints and variables) are below or equal to 10^3. This results in a subset of 62 problems, about 45% of the Maros-Mészáros test set, for which two-third of the problems have a sparsity level no larger than 20%.

We report performance profiles [16], [20], [31] on Figure 7 comparing PROXQP with its dense backend against the other solvers of the testbed. Performance profiles correspond to the fraction of problems solved as a function of a certain runtime (measured in terms of a multiple of the runtime of the fastest solver for that problem). Figure 7 depicts that even-though the problems are sparse, PROXQP always depicts the best performance profile. Hence it is the fastest approach for solving the problems of this testbed.

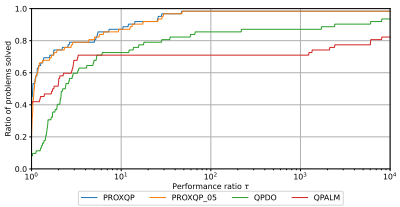
Comparison with QPDO, QPALM and variant ProxQP-05. We test for the total number of iterations and runtime for different levels of absolute tolerance from 10^{-5} to 10^{-9}. Indeed, choosing a low ϵ_{abs} when dealing with ill-conditionned problems has a few non-negligible advantages when comparing these AL-based approaches. Indeed, the fact a solver might not reach every desired level of accuracy (within the available finite precision limits) can reveal here either "inappropriate" (i) calibration strategy, or (ii) or underlying subroutines (including linear algebra ones) with a limited working precision/stability range, worsening the effect of finite precision. Table VII demonstrates that PROXQP consistently achieves significantly lower shifted geometric means for iterations at all target accuracy levels compared to both QPALM and QPDO. We can also see that PROXQP-05 (for which $\alpha = 0.5$) performs slightly worse than PROXQP, suggesting that the parameter α has a significantly less important impact than the BCL globalization strategy. Furthermore, both PROXQP and PROXQP-05 exhibit lower failure rates, leading to better shifted geometric means when evaluated based on runtime. The results for a target accuracy of 10^{-9} are illustrated in Figure 8a and Figure 8b.

VII. CONCLUSION

In this work, we have introduced a new algorithm for solving generic QPs, motivated by robotic and control applications. Among others, we combine the bound constraint Lagrangian (BCL) globalization strategy [27] for automatically scheduling key parameters of a primal-dual proximal augmented Lagrangian algorithm. The intermediary proximal subproblems are solved via a semi-smooth Newton method. The convergence



(a) Iterations comparison.



(b) Solving time comparison.

Fig. 8: Performance profiles on small to medium-sized Maros-Mészáros problems, using a Core i5 - 5300U - 5th Generation @ 2,3 GHz processor. The higher the better. Target accuracy $\epsilon_{\text{abs}} = 10^{-9}$. The total number of iterations is far less important for PROXQP than for both QPALM and QPDO, resulting also in better running time performance. The variant PROXQP-05 ($\alpha = 0.5$) performs slightly worse than PROXQP in number of iterations, impacting its timing performance.

guarantees come together with an efficient software implementation of PROXQP in the open-source PROXSUITE library, which contains several practical features for embedded optimization. The performance of PROXQP is evaluated on different standard sparse and dense robotic and control experiments, including a real-world closed-loop controller application. In particular, we have highlighted how solving the *closest* feasible QPs can be efficiently leveraged in the context of closed-loop convex MPC. Finally, through various benchmarks of the optimization literature [20], [57], [58], we have shown that PROXQP performs at the level of state-of-the-art solvers on a large set of generic QP problems.

As future works, we plan to further study and extend the primal-dual proximal augmented Lagrangian approach developed in this work to more general contexts beyond quadratic costs and linear constraints, as initiated through recent works [12], [65]. Another appealing research direction concerns the extension of the approach for proper differentiation of QP problems in the context of differentiable optimization [66], notably in the context of infeasible QPs [17]. From a software perspective, we plan to provide long-term support to help disseminate the approach for research, industrial, and commercial contexts.

ACKNOWLEDGMENTS

The authors would like to thank Viviane Ledoux for participating in the real-robot experiments. This work has received support from the French government, managed by the National Research Agency, through the INEXACT project (ANR-22-CE33-0007-01) and NIMBLE project (ANR-22-CE33-0008) and under the France 2030 program with the references Organic Robotics Program (PEPR O2R), the PIQ program under the management of Agence de Programme du Numérique, and "PR[AI]RIE-PSAI" (ANR-23-IACL-0008). The European Union also supported this work through the AGIMUS project (GA no.101070165). The Paris Île-de-France Région also supported this work in the frame of the DIM AI4IDF. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the funding agencies.

REFERENCES

[1] S. Redon, A. Kheddar, and S. Coquillart, "Gauss least constraints principle and rigid body simulations," in *2002 International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2002, pp. 517–522.

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

[3] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[4] A. Herzog, N. Rotella, S. Mason, F. Grimminger, S. Schaal, and L. Righetti, "Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid," *Autonomous Robots*, vol. 40, no. 3, pp. 473–491, 2016.

[5] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, no. 3, pp. 429–455, 2016.

[6] P.-B. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *2006 International Conference on Humanoid Robots*. IEEE, 2006, pp. 137–142.

[7] J. Carpentier and P.-B. Wieber, "Recent progress in legged robots locomotion control," *Current Robotics Reports*, vol. 2, no. 3, pp. 231–238, 2021.

[8] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. Del Prete, "Optimization-based control for dynamic legged robots," *arXiv preprint arXiv:2211.11644*, 2022.

[9] D. B. Leineweber, I. Bauer, H. G. Bock, and J. P. Schlöder, "An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization. part 1: theoretical aspects," *Computers & Chemical Engineering*, vol. 27, no. 2, pp. 157–166, 2003.

[10] B. Houska, H. J. Ferreau, and M. Diehl, "Acado toolkit—an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

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

[12] 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 (IROS)*. IEEE, 2022, pp. 13 371–13 378.

[13] M. Ciocca, P.-B. Wieber, and T. Fraichard, "Strong recursive feasibility in model predictive control of biped walking," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 730–735.

[14] D. Q. Mayne, M. M. Seron, and S. Raković, "Robust model predictive control of constrained linear systems with bounded disturbances," *Automatica*, vol. 41, no. 2, pp. 219–224, 2005.

[15] N. A. Villa and P.-B. Wieber, "Model predictive control of biped walking with bounded uncertainties," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 836–841.

[16] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond," in *RSS 2022 - Robotics: Science and Systems*, New York, United States, Jun. 2022.

[17] A. Bambade, F. Schramm, A. Taylor, and J. Carpentier, "QPLayer: efficient differentiation of convex quadratic optimization," Jun. 2023, working paper or preprint.

[18] A. Bambade, F. Schramm, S. E. Kazdadi, S. Caron, A. Taylor, and J. Carpentier, "Companion Report of PROXQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond," INRIA, Tech. Rep., Sep. 2023. [Online]. Available: https://inria.hal.science/hal-04196897

[19] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, June 2016.

[20] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[21] Mosek, "MOSEK optserver documentation," *Mosek*, 2022.

[22] G. Optimization, "GUROBI optimizer reference manual," *Gurobi Optimization*, 2020.

[23] 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.

[24] M. R. Hestenes, "Multiplier and gradient methods," *Journal of Optimization Theory and Applications*, vol. 4(5), p. 303–320, 1969.

[25] M. J. D. Powell, "A method for nonlinear constraints in minimization problems," *Optimization*, pp. 283–298, 1969.

[26] J. Sun, "On piecewise quadratic Newton and trust region problems," *Mathematical Programming*, vol. 76, pp. 451–467, 1997.

[27] A. R. 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, no. 2, pp. 545–572, 1991.

[28] J. Nocedal and S. Wright, *Numerical optimization*, 2nd ed., ser. Springer series in operations research and financial engineering. Springer, 2006.

[29] A. Chiche and J. C. Gilbert, "How the augmented Lagrangian algorithm can deal with an infeasible convex quadratic optimization problem," *Journal of Convex Analysis*, vol. 23, no. 2, 2016.

[30] 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)*, 2019.

[31] ——, "QPALM: a proximal augmented lagrangian method for nonconvex quadratic programs," *Mathematical Programming Computation*, vol. 14, no. 3, pp. 497–541, 2022.

[32] D. Marchi., "On a primal-dual newton proximal method for convex quadratic programs," *Computational Optimization and Applications*, vol. 76, pp. 451—-467, 2021.

[33] E. G. Birgin and J. M. Martínez, *Practical Augmented Lagrangian Methods for Constrained Optimization*. SIAM, 2014.

[34] A. d'Aspremont, D. Scieur, and A. Taylor, "Acceleration methods," *Foundations and Trends® in Optimization*, vol. 5, no. 1-2, pp. 1–245, 2021.

[35] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Lancelot a Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992.

[36] B. Plancher, Z. Manchester, and S. Kuindersma, "Constrained unscented dynamic programming," in *2017 International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5674–5680.

[37] S. El Kazdadi, J. Carpentier, and J. Ponce, "Equality Constrained Differential Dynamic Programming," in *2021 International Conference on Robotics and Automation (ICRA)*, 2021.

[38] W. Jallet, N. Mansard, and J. Carpentier, "Implicit differential dynamic programming," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[39] F. Facchinei and J.-S. Pang, *Finite-dimensional variational inequalities and complementarity problems*. Springer, 2003.

[40] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[41] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[42] S. J. Wright, "Primal-dual interior-point methods," *SIAM*, 1997.

[43] N. Parikh and S. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, p. 127–239, jan 2014.

[44] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming*, vol. 27, pp. 1–33, 1983.

[45] D. Arnström, A. Bemporad, and D. Axehill, "A dual active-set solver for embedded quadratic programming using recursive LDL^T updates," *IEEE Transactions on Automatic Control*, vol. 67, no. 8, pp. 4362–4369, 2022.

[46] N. I. M. Gould, D. Orban, and P. L. Toint, "GALAHAD user documentation," *GALAHAD*, 2017.

[47] C. Mészáros, "The bpmpd interior point solver for convex quadratic problems," *Optimization Methods and Software*, vol. 11, no. 1-4, pp. 431–449, 1999.

[48] E. M. Gertz and S. J. Wright, "Object-oriented software for quadratic programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 1, pp. 58–81, 2003.

[49] 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, 2020.

[50] A. Domahidi, E. Chu, and S. P. Boyd, "ECOS: an SOCP solver for embedded systems," in *ECC*. IEEE, 2013, pp. 3071–3076.

[51] M. S. Andersen, J. Dahl, and L. Vandenberghe, "Cvxopt: A python package for convex optimization," *version 1.1.6.*, 2013.

[52] A. G. Pandala, Y. Ding, and H.-W. Park, "qpSWIFT: A Real-Time Sparse Quadratic Program Solver for Robotic Applications," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3355–3362, 2019.

[53] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, June 2016.

[54] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," 2010.

[55] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[56] D. Ruiz, "A scaling algorithm to equilibrate both rows and columns norms in matrices," Rutherford Appleton Laboratory, Tech. Rep., 2001.

[57] I. Maros and C. Mészaros, "A repository of convex quadratic programming problems," *Optimization Methods and Software*, vol. 11(1-4), pp. 671–681, 1999.

[58] 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, 2009.

[59] "Upkie wheeled biped robots," Build instructions and open source software at https://github.com/upkie, last accessed on 2024-07-23, 2024.

[60] 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 *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019, pp. 614–619.

[61] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, "The 3d linear inverted pendulum mode: a simple modeling for a biped walking pattern generation," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, vol. 1, 2001, pp. 239–246 vol.1.

[62] S. Caron, A. Kheddar, and O. Tempier, "Stair climbing stabilization of the hrp-4 humanoid robot using whole-body admittance control," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 277–283.

[63] H. Audren, J. Vaillant, A. Kheddar, A. Escande, K. Kaneko, and E. Yoshida, "Model preview control in multi-contact motion-application to a humanoid robot," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 4030–4035.

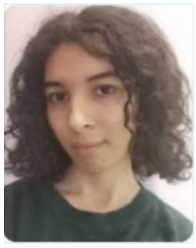
[64] N. Scianca, D. De Simone, L. Lanari, and G. Oriolo, "Mpc for humanoid gait generation: Stability and feasibility," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1171–1188, 2020.

[65] W. Jallet, A. Bambade, N. Mansard, and J. Carpentier, "Proxnlp: a primal-dual augmented lagrangian solver for nonlinear programming in robotics and beyond," *arXiv preprint arXiv:2210.02109*, 2022.

[66] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," *Advances in neural information processing systems*, vol. 32, 2019.

Antoine Bambade received the Engineering degree in applied and computational mathematics from the École Polytechnique, Palaiseau, France, the M.Sc. degree in mathematical statistics from the University of Cambridge (tripos part III), Cambridge, U.K., both in 2019, and the Ph.D. degree in mathematics from the University ENS PSL, Paris, France, in 2024. He is currently a Senior Civil Servant of the "Corps des Ponts, des Eaux et Forêts" and also a Research Scientist with EDF Lab, Gif-sur-Yvette, France. He is the main author and developer of the ProxSuite optimization toolkit (ProxQP, QPLayer). His research interests cover mixed-integer, continuous, stochastic, distributed optimization, and machine learning methods for real-time applications.

Fabian Schramm Fabian Schramm is a third-year Ph.D. student jointly affiliated with the Willow team at Inria Paris and the Institute for Intelligent Systems and Robotics (ISIR) at Sorbonne University, under the supervision of Justin Carpentier and Nicolas Perrin-Gilbert. His research focuses on enhancing robotic agility and dexterity through the integration of optimal control and reinforcement learning. Before starting his Ph.D., he worked as a research engineer at Wandercraft and Inria, contributing to control algorithms for exoskeletons and the development of open-source robotics libraries. He holds a master's degree in Robotics, Cognition and Intelligence from the Technical University of Munich.



Sarah El-Kazdadi received the Engineering degree in computational mathematics from Ecole Polytechnique, Palaiseau, France, in 2019. She was a Research Engineer with the Willow research team at Inria, Paris, France, focusing on high-performance numerical optimization and linear algebra. She is currently a Research Engineer with NVIDIA, Santa Clara, CA, USA. She is one of the core developers of ProxSuite, and is involved in the open-source community, with a focus on modern systems programming languages and high-performance scientific computing.

Stéphane Caron is a researcher at Inria. He received his M.Sc. in computer science from the École Normale Supérieure (2012) and his Ph.D. in robotics from the University of Tokyo (2016). Stéphane has worked at CNRS as a tenured researcher and at ANYbotics AG as locomotion team lead before joining Inria, where his research now revolves around visuomotor control. Stéphane is a proponent of open source robotics and contributes to projects like robot description and inverse-kinematics libraries, the benchmarking of quadratic programming solvers, and Upkie open-source wheeled biped robots.

Adrien Taylor is a research scientist at the Inria Paris research center, within the Sierra team (a joint project between Inria, the École normale supérieure in Paris, and the CNRS). He previously studied and obtained his PhD at UCLouvain. His work focuses on numerical and convex optimization with a particular emphasis on understanding and analyzing the dynamics of optimization algorithms. His research has been recognized with several distinctions, including the IBM-FNRS Innovation Award and a nomination as a finalist for the Tucker Prize from the Mathematical Optimization Society. He has also received best paper awards (Optimization Letters, 2017; NeurIPS, 2021), reviewer awards (NeurIPS, ICML, Mathematical Programming), and, more recently, a European Research Council (ERC) Starting Grant to support the development of his research program.

Justin Carpentier (Member, IEEE) is a researcher at Inria and École Normale Supérieure, heading the Willow research team since 2023. He graduated from École Normale Supérieure Paris-Saclay in 2014 and received a Ph.D in Robotics in 2017 from the University of Toulouse. He did his Ph.D. in the Gepetto team at LAAS-CNRS in Toulouse, working on the computational foundations of legged locomotion. In 2024, he received an ERC Starting Grant focusing on laying the algorithmic and computational foundations of Artificial Motion Intelligence. His research interests lie at the interface of optimization, machine learning, computer vision, simulation, and control for robotics, with applications ranging from agile locomotion to dexterous manipulation. He is the main developer and manager of widely used open-source robotics software, among them Pinocchio, ProxSuite, COAL, Aligator and Simple. He is currently an Associate Editor of *IEEE Transactions on Robotics* and *IEEE Robotics and Automation Letters*.