Università di Genova

ISTITUTO ITALIANO
DI TECNOLOGIA
DYNAMIC LEGGED SYSTEMS

Scaling Model Predictive Control for High-Dimensional Robotic Systems

by

Lorenzo Amatucci

Thesis submitted for the degree of *Doctor of Philosophy* (38^{th} cycle)

February 2026

Victor Barasuol	Supervisor
Giulio Turrisi	Co-Supervisor
Claudio Semini	Co-Supervisor
Paolo Massobrio	Head of the PhD program

Reviewers:

Steve Tonneau, *University of Edinburgh*

Zachary Manchester, *Massachusetts Institute of Technology (MIT)*

Dibris

Department of Informatics, Bioengineering, Robotics and Systems Engineering

A mamma e papà

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Lorenzo Amatucci
February 2026

Acknowledgements

I would like to express my gratitude to my supervisors, Claudio, Victor, and Giulio, for their guidance, encouragement, and especially for their patience throughout these years. Even when it may have seemed that I was not listening, your advice was always valuable, and without your feedback, all the achievements of my PhD would not have been possible.
My sincere thanks go also to everyone in the DLS Lab for their assistance, discussions, and for making the lab an exceptional working environment. Special thanks to Angelo, Yle, Gio, Marco Marchitto, Illyas, and Gianlu, not only for being great colleagues but also for your deep friendship.
A special thanks also goes to all my friends in Genova, in particular Nicco, Giuli, Lollo, and Miki, for making Genova feel like my new home. I would also like to thank all the friends far away: Lo', Filli, Ema, and Isa. Even though we cannot see each other often, I know I can always count on you.
Finally, I am deeply grateful to my family for their unconditional support, encouragement, and understanding throughout all these years. In particular, I would like to thank my mom and dad, who have encouraged me since I left home many years ago to follow my passions and dreams. Thanks for making this journey possible.
Last but not least, I would like to thank Carlotta, the coauthor of the rest of my life. Thank you for making the hard times easier and the good times happier.

Abstract

Legged robots are increasingly recognized for their potential to revolutionize various sectors by performing tasks that are challenging and hazardous for humans. Unlocking new behaviors is pivotal not only for disaster response but also to expand their role in industries such as logistics and manufacturing.

Controlling such robotic systems is a challenging task. The robot should not only be able to define a strategy autonomously, but also to replan it when necessary. Model Predictive Control (MPC) promises to be a good candidate for the application, as the task can be encoded as an Optimal Control Problem (OCP) that is solved at each control loop, with the goal expressed via a cost function and the dynamic feasibility enforced through optimization constraints. However, MPC faces practical limitations. To obtain the desired behavior, the optimization should include high-fidelity models that capture the robot's dynamics, constraints that accurately reflect physical limits, and a prediction horizon long enough to be representative of the task, leading to the definition of large constrained non-linear optimization problems that must be solved in milliseconds to maintain fast re-planning rates. These conflicting requirements force trade-offs due to the computational bottleneck, limiting the deployment of MPC. In this thesis, we try to address such challenges, developing methods to solve large-scale OCPs maximizing hardware utilization, exploiting learning approaches, and modern GPU accelerators. Our goal is to retain rich models, expressive constraints, and long horizons without sacrificing the real-time performance that MPC demands.

As a first step, we developed a whole-body MPC on legged robots exploiting distributed optimization to speed up the computational time. In this first work, we decompose the robot's whole-body dynamics into smaller, parallelizable subsystems and enforce consistency via the Alternating Direction Method of Multipliers (ADMM). Each subsystem solves its own OCP while a consensus sweep maintains coherence across the full robot. This reduces computation time and enables efficient scaling to more complex morphologies; for example, adding articulated arms to a quadruped does not increase computational time despite the additional degrees of freedom. Numerical evaluations show convergence to

solutions consistent with state-of-the-art centralized methods, with up to a 75% reduction in computation time. This demonstrates a practical path to exploiting modern multi-core hardware for faster whole-body MPC on hardware.

In order to achieve fast replanning rates, we had to use predefined contact timings, since contact-sequence generation is a challenging task for optimization-based controllers due to both the combinatorial nature of contact selection and the numerical difficulties introduced by contact constraints (e.g., complementarity). We tackled this problem in a second work, casting contact planning as a sampling-based search using Monte Carlo Tree Search (MCTS) over feasible contact transitions, and bootstrapping the search with a learned value function to guide exploration. This hybrid approach preserves model-based guarantees for out-of-distribution scenarios while leveraging learning for speed. The resulting planner produces contact sequences and timings in real time, enabling reactive, non-gaited locomotion on hardware.

While the first work enables CPU parallelization, its benefits are limited by the relatively small number of available threads. By contrast, GPUs offer massive parallelism and the possibility for substantial gains. To harness accelerators' potentials, we developed a differentiable, GPU-accelerated MPC framework. We introduce a Sequential Quadratic Programming solver that exploits temporal and state-space parallelism, using parallel associative scans to solve the KKT system. This yields an overall complexity of $\mathcal{O}(\log^2(n)\log N + \log^2(m))$ with n states, m control inputs, and horizon length N instead of $\mathcal{O}(N(n+m)^3)$. Compared to the state of the art, we achieve logarithmic scaling in horizon and square-log in state dimension, enabling whole-body MPC with horizons up to 300 nodes and centralized MPC for a fleet of 16 quadrupeds, all computed in under 25 ms. Implemented in JAX, the solver is differentiable and supports large-scale batching across environments, making it feasible to train policies with MPC-in-the-loop directly on the GPU, thereby bridging model-based and learning-based methods. This opens the door to new paradigms in legged locomotion that combine the strengths of both approaches.

Overall, this thesis advanced the computational efficiency, scalability, and practicality of MPC-driven legged locomotion through numerical and learning methods.

Table of contents

List of figures

List of tables

Glossary

Notation

x, α Scalars.

$\boldsymbol{x}, \boldsymbol{\alpha}$ Vectors.

$\boldsymbol{X}, \boldsymbol{A}$ Matrices.

$\boldsymbol{A}^T$ Transpose of matrix $\boldsymbol{A}$.

$\boldsymbol{A}^{-1}$ Inverse of matrix $\boldsymbol{A}$.

$\boldsymbol{\nabla}_{\cdot}$ Gradient with respect to $\cdot$.

$\boldsymbol{\nabla}^2_{\cdot}$ Hessian with respect to $\cdot$.

$\cdot|_z$ Z axis projection.

$\|\cdot\|_*$ Weighted norm for * weights.

Symbols

$\mathcal{L}$ Lagrangian function.

$\boldsymbol{x}$ State vector.

$\boldsymbol{u}$ Control input vector.

$\boldsymbol{y}$ Decision variable.

$\hat{\boldsymbol{x}}_0$ Initial state vector.

U Constraint set control input.

X Constraint set state vector.

N Horizon length.

n State vector dimension.

m Control input dimension.

$l_k(\boldsymbol{x}_k, \boldsymbol{u}_k)$ Stage cost at time step k.

$l_N(\boldsymbol{x}_N, \boldsymbol{u}_N)$ Terminal cost.

$f(\cdot)$, $g(\cdot)$ Cost functions.

$\boldsymbol{c}(\boldsymbol{y})$ Optimization constraint.

$\boldsymbol{\lambda}$ Lagrange multipliers.

ρ Penalty ADMM.

$\boldsymbol{w}$, $\boldsymbol{z}$ ADMM splitted variables.

$\boldsymbol{q}$ Generalized coordinates.

$\boldsymbol{v}$ Generalized velocities.

$\boldsymbol{M}$ Mass matrix.

$\boldsymbol{b}$ Bias vector.

$\boldsymbol{dt}$ Integration step.

$\boldsymbol{S}$ Selection matrix.

$\boldsymbol{\tau}$ Actuation torques.

$\boldsymbol{J}$ Jacobian matrix.

$\boldsymbol{\lambda}_{grf}$ Ground reaction forces.

$\boldsymbol{\lambda}_{int}$ Interface wrench.

$\boldsymbol{p}_f$ Foot position.

$\boldsymbol{p}_{f,ref}$ Foot position reference.

$\boldsymbol{r}_f$ CoM-foot displacement vector.

r ADMM residual.

$\boldsymbol{I}_c$ Inertia matrix.

m Mass.

$\boldsymbol{g}$ Gravity vector.

$\boldsymbol{p}_c$ Center of mass position.

$\boldsymbol{v}_c$ Center of mass velocity.

$\boldsymbol{\omega}^b$ Angular velocity.

$\boldsymbol{\Phi}$ Euler angles.

δ Binary contact variable.

t_{stance} Stance time.

t_{swing} Swing time.

$\boldsymbol{c}_n$ Contact sequence.

$\boldsymbol{V}_i$ Value function.

$\boldsymbol{V}_{i \to k}$ Conditional value function.

$\boldsymbol{h}_{i \to k}$ Optimal conditional trajectory.

$\boldsymbol{h}\left(\boldsymbol{x}, \boldsymbol{u}\right)$ Dynamics constraint.

θ Dynamics constraint violation.

N_{sym} MCTS number of simulations.

K Riccati feedback gain.

k Riccati feedforward gain.

Acronyms

ADMM Alternating Direction Method of Multipliers.

AP Action Policy.

CoM Center of Mass.

CPU Central Processing Unit.

CVF Conditional Value Function.

DDP Differential Dynamic Programming.

DoF Degrees of Freedom.

EOD Extreme Out-of-Distribution.

GPU Graphics Processing Unit.

GRF Ground Reaction Forces.

ID Inside Distribution.

iLQR Iterative Linear Quadratic Regulator.

KKT Karush-Kuhn-Tucker.

LCB Lower Confidence Bound.

LICQ Linear Independence Constraint Qualification.

MCTS Monte Carlo Tree Search.

MDP Markov Decision Process.

MHE Moving Horizon Estimator.

MIP Mixed-Integer Program.

MLP Multi-Layer Perceptron.

MPC Model Predictive Control.

MPCC Mathematical Program with Complementarity Constraints.

NLP Nonlinear Programming.

OCP Optimal Control Problem.

OD Outside Distribution.

QP Quadratic Programming.

RL Reinforcement Learning.

RTI Real-Time Iteration.

SQP Sequential quadratic Programming.

SRBD Single Rigid Body Dynamics.

VF Value Function.

WB Whole Body.

Chapter 1

Introduction

1.1 Motivation

Since their first appearance in science-fiction tales, robots have captured the human imagination as tools to enhance productivity, explore new frontiers, and generally improve the quality of life. In the 1960s, robots were widely adopted in manufacturing, substituting humans in repetitive and dangerous tasks, revolutionizing production lines, and significantly boosting efficiency. Robots have also been increasingly deployed in logistics, where warehouse automation has optimized supply chains and reduced operational costs.

While industrial robots have proven incredibly effective in such structured environments, they remain limited in versatility. They are typically confined to pre-programmed tasks and often require specially designed production lines or warehouses that minimize or eliminate physical human interaction. This paradigm of rigid, environment-dependent automation stands in contrast to the early vision of general-purpose robots capable of adapting to their surroundings.

These limitations have driven research into the development of legged robots, which, thanks to their morphology, can navigate unstructured environments and interact with humans and objects in a more natural and flexible way. Legged robots can operate within spaces designed for humans and even exhibit mobility capabilities that surpass those of humans and animals. They have the potential to revolutionize a range of fields, from search-and-rescue operations in disaster zones to exploration of remote or hazardous locations such as planetary surfaces, and also transform last-mile delivery and assistive robotics for the elderly and disabled.

However, legged robots pose a significant challenge in terms of control and planning due to the high number of Degrees of Freedom (DoF), the inevitable under-actuation, and the

way they interact with the environment, making their deployment fundamentally difficult. As a starting point for our work, we sought a general framework for legged robot control that could enable a rigorous mathematical formulation of the problem and could provide a solid foundation for our contributions. In the next section, we will give a short introduction to Model Predictive Control (MPC), which will serve as the primary framework throughout this thesis.

1.2 Context

In the chase for general-purpose machines, robots must be capable of autonomously planning strategies to accomplish their tasks, but also rapidly modifying the plan in case of unforeseen disturbances or changes in the environment. MPC has emerged as a compelling candidate that could enable such capabilities. In MPC, the control policy is given by the solution of an Optimal Control Problem (OCP) that is solved online at each control loop. The cornerstone of MPC and the core of our motivation is the idea that an OCP can ideally fully describe the task, the robot, and its interaction with the environment around it. A generic OCP can be formulated as follows:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \boldsymbol{l}_{N+1}(\boldsymbol{x}_{N+1}) + \sum_{i=0}^{N} \boldsymbol{l}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ \text{s.t.} \quad & \boldsymbol{x}_{i+1} = \boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i) \\ & \boldsymbol{u}_i \in \mathsf{U}_i \\ & \boldsymbol{x}_i \in \mathsf{X}_i \\ & \boldsymbol{x}_0 = \hat{\boldsymbol{x}}_0 \end{aligned} \tag{1.1}$$

where $\boldsymbol{x}_i \in \mathbb{R}^n$ and $\boldsymbol{u}_i \in \mathbb{R}^m$ are the system state and control input at time step i, $\boldsymbol{h}(\cdot)$ represents the system dynamics, $\boldsymbol{l}_i(\cdot)$ are the cost functions encoding the desired behavior, and X_i and U_i are sets defining state and input constraints, respectively. The optimization is performed over a finite prediction horizon of length N, starting from the initial state $\hat{\boldsymbol{x}}_0$. This framework naturally accommodates general multi-objective tasks, properly handles arbitrary constraints, and provides an explicit mechanism for anticipatory decision-making over a finite prediction horizon. Despite these advantages, deploying MPC on legged robots presents substantial practical challenges. Achieving the desired behavior could require the optimization to incorporate complex nonlinear constraints that represent physical limits, terrain interactions, and task requirements, which introduces nontrivial structural and

computational difficulties in the solution process. We also need to consider meaningful prediction horizons for the robot to anticipate future contacts with the environment and plan effective and robust motion strategies. Together, these elements lead to large-scale, constrained nonlinear optimization problems that must be solved within a few milliseconds to sustain the fast replanning rates demanded, e.g. in locomotion, where the robot needs to constantly balance.

Simplifying dynamics or reducing horizon length improves computational tractability but sacrifices accuracy and robustness. Conversely, increasing model complexity or horizon length improves performance but often exceeds real-time limits. As a result, the practical use of MPC on legged robots remains constrained by computational bottlenecks, motivating our research towards more effective methods that keep the highest OCP complexity while maintaining real-time feasibility.

1.3 Background

We now take a step back to see how these OCPs are solved in literature, to pinpoint where the computational bottleneck arises from. The OCP in (1.1), as we already said, is a constrained non-linear optimization problem that can be rewritten in a more general form as:

$$\min_{\boldsymbol{y}} \; f(\boldsymbol{y}) \qquad \text{subject to} \qquad \begin{cases} \boldsymbol{c}_i(\boldsymbol{y}) = 0, & i \in \mathcal{E}, \\ \boldsymbol{c}_i(\boldsymbol{y}) \geq 0, & i \in \mathcal{I}, \end{cases} \tag{1.2}$$

Where $\boldsymbol{y}$ is the stack of all state and control variables in the horizon, $f(\boldsymbol{y})$ is the cost function to minimize, and $\boldsymbol{c}_i(\boldsymbol{y})$ are the equality and inequality constraints that include the dynamics and all other constraints in (1.1), while $\mathcal{E}$ and $\mathcal{I}$ are the sets of indices for equality and inequality constraints, respectively. For (1.2) we can define the First-Order necessary condition known also as Karush-Kuhn-Tucker (KKT) condition of optimality [Jorge and Stephen, 1999]:

$$\begin{aligned} \nabla_y \mathcal{L}(\boldsymbol{y}^*, \boldsymbol{\lambda}^*) &= \mathbf{0} \\ \boldsymbol{c}_i(\boldsymbol{y}_i^*) &= 0, \qquad i \in \mathcal{E} \\ \boldsymbol{c}_i(\boldsymbol{y}_i^*) &\geq 0, \qquad i \in \mathcal{I} \\ \boldsymbol{\lambda}_i^* &\geq 0, \qquad i \in \mathcal{I} \\ \boldsymbol{\lambda}_i^* \, \boldsymbol{c}_i(\boldsymbol{y}_i^*) &= 0, \qquad i \in \mathcal{I} \end{aligned} \tag{1.3}$$

where $\mathcal{L}(\boldsymbol{y}, \boldsymbol{\lambda}) = f(\boldsymbol{y}) + \sum_{i \in \mathcal{E}} \boldsymbol{\lambda}_i \boldsymbol{c}_i(\boldsymbol{y}) + \sum_{i \in \mathcal{I}} \boldsymbol{\lambda}_i \boldsymbol{c}_i(\boldsymbol{y})$ is the Lagrangian function associated with (1.2), and $\boldsymbol{\lambda}_i$ are the components of Lagrange multipliers associated with the constraints.

The KKT conditions provide, assuming that f and c are continuously differentiable and that the Linear Independence Constraint Qualification (LICQ) holds, a set of equations that must be satisfied at the optimal solution $(\boldsymbol{y}^*, \boldsymbol{\lambda}^*)$. We will later see in Sec.A.1.1 how LICQ plays a crucial role, as some constraints that naturally arise in MPC, such as static friction or contacts, tend to violate this assumption. For the inequality constraints, the complementary condition in the last three line ensures that either the constraint is active (i.e., $\boldsymbol{c}_i(\boldsymbol{y}^*) = 0$) with a positive multiplier ($\boldsymbol{\lambda}_i^* > 0$), or the constraint is inactive (i.e., $\boldsymbol{c}_i(\boldsymbol{y}^*) > 0$) with a zero multiplier ($\boldsymbol{\lambda}_i^* = 0$). Meaning that once the active set of constraints is defined only those constraints influence the optimal solution, acting as equality. In practice, many times the active set is not known a priori and solvers deal with inequality constraints through barrier or penalty methods to avoid the complexity of iteratively reevaluating the set. Ignoring the inequality constraints for the moment, the first two relations in (1.3) yield a system of nonlinear equations that can be solved using Newton's method. In particular, the Newton step at iteration k is obtained by solving the linearized KKT system:

$$\begin{bmatrix} \nabla^2_{\boldsymbol{yy}} \mathcal{L}^k & -\nabla \boldsymbol{c}(\boldsymbol{y}^k)^\top \\ \nabla \boldsymbol{c}(\boldsymbol{y}^k) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{y} \\ \delta \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla_{\boldsymbol{y}} \mathcal{L}^k \\ \boldsymbol{c}(\boldsymbol{y}^k) \end{bmatrix}, \tag{1.4}$$

where $\nabla^2_{\boldsymbol{yy}} \mathcal{L}^k$ is the Lagrangian Hessian,$\nabla \boldsymbol{c}(\boldsymbol{y}^k)$ is the constraint Jacobian and $\delta \boldsymbol{y}$ and $\delta \boldsymbol{\lambda}$ are the primal and dual Newton directions. The updates are then applied as:

$$\begin{aligned} \boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \alpha\, \delta \boldsymbol{y}, \\ \boldsymbol{\lambda}^{k+1} &= \boldsymbol{\lambda}^k + \alpha\, \delta \boldsymbol{\lambda}, \end{aligned} \tag{1.5}$$

where $\alpha \in [0, 1]$ is a step size typically determined through globalization techniques.

In the most general implementation, the KKT system (1.4) is factorized using dense linear algebra. Since the KKT matrix has dimension proportional to the total number of decision variables, i.e., $\boldsymbol{y} \in \mathbb{R}^{N(n+m)}$ for the OCP in (2.6), the cost of such a factorization scales cubically with the horizon length N. This makes the approach entirely impractical for MPC, where the Newton step must be computed in a few milliseconds.

Crucially, using dense algebra ignores the sparsity and block structure induced by the optimal control problem. In (1.1), the dynamics couple only consecutive stages, while other constraints (e.g. joint limits or friction cones) depend only on the variables at each stage. This yields a block-tridiagonal structure in the KKT system. Writing the KKT matrix while

highlighting its stagewise structure gives:

$$\begin{bmatrix} \nabla^2_{\boldsymbol{y}_1\boldsymbol{y}_1}\mathcal{L}_1 & -\nabla \boldsymbol{c}_1(\boldsymbol{y}_1)^T & \mathbf{0} & \cdots & \mathbf{0} \\ \nabla \boldsymbol{c}_1(\boldsymbol{y}_1) & \nabla^2_{\boldsymbol{y}_2\boldsymbol{y}_2}\mathcal{L}_2 & -\nabla \boldsymbol{c}_2(\boldsymbol{y}_2)^T & \cdots & \mathbf{0} \\ \mathbf{0} & \nabla \boldsymbol{c}_2(\boldsymbol{y}_2) & \nabla^2_{\boldsymbol{y}_3\boldsymbol{y}_3}\mathcal{L}_3 & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \nabla^2_{\boldsymbol{y}_N\boldsymbol{y}_N}\mathcal{L}_N \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{y}_1 \\ \delta \boldsymbol{\lambda}_1 \\ \delta \boldsymbol{y}_2 \\ \vdots \\ \delta \boldsymbol{y}_N \\ \delta \boldsymbol{\lambda}_N \end{bmatrix} = - \begin{bmatrix} \nabla_{\boldsymbol{y}_1}\mathcal{L}^k_1 \\ \boldsymbol{c}_1(\boldsymbol{y}^k_1) \\ \nabla_{\boldsymbol{y}_2}\mathcal{L}^k_2 \\ \vdots \\ \nabla_{\boldsymbol{y}_N}\mathcal{L}^k_N \\ \boldsymbol{c}_N(\boldsymbol{y}^k_N) \end{bmatrix} \tag{1.6}$$

where $\boldsymbol{y}_i = [\boldsymbol{u}_{i-1}, \boldsymbol{x}_i]$ is the stack of state and control for the node i. For clarity, we dropped the appendix k indicating the Newton iteration. Because this matrix is block-tridiagonal, the system can be solved using Gaussian elimination specialized for tridiagonal systems like the Thomas algorithm [Jordana et al., 2023], which yields a solution in $\mathcal{O}(N)$ time rather than $\mathcal{O}(N^3)$, while the complexity still scales cubically with the size of each block, i.e., with the number of states n and control m. This is a significant improvement, but for high-dimensional systems like legged robots, where n and m can be large, the cubic scaling remains a critical bottleneck. Before moving on, it is useful to notice that the considerations above extend to state-of-the-art solvers tailored for optimal control problems. Although these solvers differ in how they handle nonlinearities, inequality constraints, and globalization strategies, they all exploit the same sparsity structure of the OCP to achieve linear-time complexity in the horizon length. This is true regardless of whether the method is based on Differential Dynamic Programming [Mastalli et al., 2020] [Kiessling et al., 2024], Sequential Quadratic Programming [Verschueren et al., 2021] [Farshidian et al., 2023], Interior-Point methods [Vanroye et al., 2023], or Augmented Lagrangian techniques [Howell et al., 2019]. Despite their algorithmic differences, all these approaches ultimately solve the KKT system in (1.6) or a closely related variant of it at each iteration.

1.4 Contributions

In Sec. 1.2, we discussed why we believe that MPC plays a significant role in legged-robot control, offering a general framework for simultaneous planning and control. This thesis addresses the challenges of deploying MPC for high-dimensional systems with hybrid dynamics, as legged robots. In particular, we focus on improving the solving time of the OCP in (1.1), by maximizing hardware utilization. Our contributions span algorithmic innovations, software implementations, and practical validations on real robotic platforms. The main

scientific contributions of this thesis are summarized in the works listed in this section, all published in peer-reviewed venues.

C1: Accelerating Model Predictive Control through Distributed Optimization

L. Amatucci, G. Turrisi, A. Bratta, V. Barasuol and C. Semini, "Accelerating Model Predictive Control for Legged Robots through Distributed Optimization," 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, pp. 12734-12741, doi: 10.1109/IROS58592.2024.10801676.

In **C1**, we address the challenge of deploying Whole Body (WB)-MPC by introducing a distributed optimization framework specifically designed to accelerate the solution of the OCP. Aiming for the best performance, we chose to develop a controller that uses the WB model to describe the system dynamics as it fully captures the robot's dynamics, allowing the optimization to exploit the robot's full potential. However, this model also substantially increases the computational demands of the associated OCP. In our implementation, the original problem is decomposed into smaller, independent subproblems that can be solved concurrently. Since each subproblem optimizes only a subset of the global state $\boldsymbol{x}$ and control $\boldsymbol{u}$, the computational time per subproblem is significantly reduced. Consistency between the subproblem solutions is enforced through a consensus formulation inspired by the Alternating Direction Method of Multipliers (ADMM).

We validate the convergence of the distributed approach numerically and test the overall framework both in simulation and on a real quadruped robot, demonstrating, when compared to state of the art solvers like [Verschueren et al., 2021], a two times reduction in OCP solving time for the quadruped, and up to a four times reduction for systems with a higher number of Degrees of Freedom (DoF) such as a quadruped equipped with a manipulator arm.

The code for this work is available at: https://github.com/iit-DLSLab/DWMPC.

Remark. *The underlying concept and theoretical foundations were developed by me with the supervision of G. Turrisi and V. Barasuol. The formulation and implementation were carried out by me. The experiments were conducted and analyzed by me, with support from Y. Nisticò. The manuscript was written by me with the support of A. Bratta. The entire process benefited from the guidance and supervision of all the other co-authors.*

C2: Non-Gaited Legged Locomotion With Monte-Carlo Tree Search and Supervised Learning

I. Taouil*, L. Amatucci*, K. Majid, A. Dai, V. Barasuol, G. Turrisi, C. Semini "Non-Gaited Legged Locomotion With Monte-Carlo Tree Search and Supervised Learning" in IEEE Robotics and Automation Letters, vol. 10, no. 2, pp. 1265-1272, Feb. 2025, doi: 10.1109/LRA.2024.3519908. *Equal contribution

The MPC framework presented in **C1** provides a powerful tool for legged-robot control, but it relies on predefined contact sequences, which can limit the robot's adaptability in changing environments. As noted in Sec.1.3, introducing a contact model into the OCP leads to constraints that violate LICQ, thereby breaking the presented solution method. While approximated contact models that avoid this issue do exist, they typically come at the cost of reduced model fidelity or MPC performance. On the other hand, adapting contact timings is essential for legged robots to navigate complex terrain and recover from unexpected disturbances.

We address the challenges mentioned above in **C2**. Building on our previous work on contact planning using Monte Carlo Tree Search (MCTS) [Amatucci et al., 2022], we adopt the same idea of formulating contact planning as a tree search over feasible contact transitions. To improve the efficiency of the search, we introduce a learned value function that guides exploration, allowing the planner to focus on the most promising contact sequences. This hybrid approach combines the robustness of model-based planning with the efficiency of learning-based methods, enabling real-time generation of contact sequences. We validate our approach on a quadruped robot, demonstrating its ability to perform non-gaited locomotion in real-world scenarios. To the best of our knowledge, this was the first time that a sampling-based method has been deployed for legged robot non-gaited locomotion.

Remark. *This work was carried out in collaboration with I. Taouil during his time at the DLS lab. The underlying concept and theoretical foundations were developed by me with subsequent support from I. Taouil. The formulation and implementation were carried out jointly by me and I.Taouil. The experiments were conducted and analyzed by me, with support from I. Taouil. The manuscript was written by I.Taouil with my support. The entire process benefited from the guidance and supervision of all the other co-authors.*

C3: Primal-Dual iLQR for GPU-Accelerated Learning and Control in Legged Robots

L. Amatucci*, J. Sousa-Pinto*, G. Turrisi, D. Orban, V. Barasuol and C. Semini, "Primal-Dual iLQR for GPU-Accelerated Learning and Control in Legged Robots" in IEEE Robotics and Automation Letters, 2025, doi: 10.1109/LRA.2025.3632610.
*Equal contribution

While **C1** and **C2** leverage CPU parallelism to enhance MPC performance, the scalability of CPU-based approaches is inherently limited by the number of available cores. In contrast, Graphics Processing Units (GPUs) offer massive parallelism, making them an attractive platform for accelerating MPC computations. To harness the potential of GPUs, we present in **C3** a novel SQP solver specifically designed for GPU architectures and written entirely in JAX [Bradbury et al., 2018]. The solver exploits both temporal and state-space parallelism, using parallel associative scans to efficiently solve the KKT system in $\mathcal{O}(\log N)$ time instead of the $\mathcal{O}(N)$ scaling shown in Sec. 1.3. Thanks also to the GPU-optimized matrix inversion routines in JAX, we were able to reduce the scaling with respect to the state and control dimensions n and m, from cubic to squared-logarithmic. Our implementation not only enables the handling of longer horizons and higher-dimensional state spaces, but also supports large-scale batching across multiple environments and the computation of gradients through the solver with respect to arbitrary parameters, making it feasible to train policies with MPC-in-the-loop directly on the GPU. We validate our approach numerically, demonstrating WB-MPC with horizons up to 300 nodes and centralized MPC for a fleet of 16 quadrupeds, all computed in under 25 ms. Compared to state of the art we achieved up to a 60% reduction in solving time with respect to crocoddyl [Mastalli et al., 2020] for long horizon WB-MPC and up to 600% with respect to acados [Verschueren et al., 2021] in centralized Single Rigid Body Dynamics (SRBD)-MPC. We also showcased the controller performance on a real quadruped and in simulation on humanoid robots in various scenarios.

The code for this work is available at: https://github.com/iit-DLSLab/mpx.

Remark. *This work was carried out in collaboration with J. Sousa-Pinto. The underlying concept and theoretical foundations were developed by J. Sousa-Pinto and me. The formulation and implementation were carried out jointly by myself and J. Sousa-Pinto. The experiments were conducted and analyzed by me and S. Tittel. The manuscript was written by me. The entire process benefited from the guidance and supervision of the other co-authors.*

C4: VERO: A vacuum-cleaner-equipped quadruped robot for efficient litter removal

L. Amatucci, G. Turrisi, A. Bratta, V. Barasuol, and C. Semini, "VERO: A vacuum-cleaner-equipped quadruped robot for efficient litter removal" Journal of Field Robotics, vol. 41, no. 6, pp. 1829-1842, 2024, doi: https://doi.org/10.1002/rob.22350.

This last work needs a special remark, since it is not a proper fit to the general scientific contributions of this thesis. However, we decided to include it in Appendix A since it has been carried out during the first 8 months of my PhD and showcases a real-world application of legged robots. **C4** presents VERO, a quadruped robot equipped with a vacuum cleaner designed for efficient litter removal in outdoor environments. Thanks to the leg mounted nozzle and advanced perception and navigation capabilities VERO can autonomously identify and collect litter. We validate VERO's performance through extensive field tests, testing the system in both urban and wild scenarios. Showcasing its ability to operate effectively in diverse outdoor settings.

Remark. *The initial concept was from C. Semini. The theoretical foundations were developed by me under the supervision of V. Barasuol and C. Semini. The design and implementation were carried out by me with the support of V. Barasuol. The hardware was designed by M. Villa. The experiments were conducted and analyzed by me, A. Bratta, and G. Turrisi. The manuscript was written by me with the support of G. Turrisi. The entire process benefited from the guidance and supervision of all the other co-authors.*

1.5 Outline

The rest of the thesis is organized as a collection of self-contained chapters, corresponding to the peer-reviewed publications, listed in Sec. 1.2. Chapter 2 presents the distributed optimization framework for WB-MPC presented in **C1**. Chapter 3 details the MCTS-based contact planner with learned value function published in **C2**. Chapter 4 describes the GPU-accelerated SQP solver and its application to legged robots from **C3**. Appendix A showcases the VERO quadruped robot for litter removal from **C4**. Each chapter includes an introduction, related work, methodology, experimental results, and conclusions, providing a comprehensive overview of the respective contributions. Finally, in chapter 5, we draw the thesis conclusion and highlight some possible future research directions.

Chapter 2

Accelerating Model Predictive Control through Distributed Optimization

L. Amatucci, G. Turrisi, A. Bratta, V. Barasuol and C. Semini, "Accelerating Model Predictive Control for Legged Robots through Distributed Optimization," 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, pp. 12734-12741, doi: 10.1109/IROS58592.2024.10801676.

2.1 Motivation

This chapter presents a distributed approach to speed up the computation of Whole Body (WB)-Model Predictive Control (MPC) on legged robots. Our approach splits the robot dynamics into smaller independent subsystems that can be solved separately, while ensuring consensus thanks to an Alternating Direction Method of Multipliers (ADMM) inspired communication. The proposed method was developed with the goal of creating a scalable MPC capable of running at high frequencies on onboard hardware without sacrificing model accuracy, even for robots with a high number of Degrees of Freedom (DoF). This enables the MPC to directly control the robot without requiring additional cascaded controllers, as is commonly done in the literature [Grandia et al., 2023] [Romualdi et al., 2022] [Bertsekas, 2024]. Such additional controllers often require extensive tuning and can lead to suboptimal behaviors. The idea of splitting the robot dynamics into smaller subsystems came after realizing the fact that the dominant factor driving the solving times of the Optimal Control Problem (OCP) is the size of the state and control vector, which for legged robots directly

scales with the number of legs and therefore the number of joints. Furthermore, we observed that the multi-core CPUs onboard the robot were underutilized, with most computations being executed on a single core. This meant that if we had a way to split the problem into different independent subproblems, each using a smaller subset of joints, we could achieve substantial performance improvements. With the idea of splitting the robot dynamics into smaller subsystems, we needed a method to keep the different optimizations consistent. Consensus ADMM emerged as a natural choice, providing a one-step communication mechanism between the subsystems. This means that each subproblem only has access to the most recent estimate of the variables from the other subproblems. It is worth noting that while the approximation given by such an approach may appear drastic at first glance, Newton-based solvers already linearize the system dynamics around the previous solution. Therefore, the approximation introduced by the decoupling is not as severe as it may initially seem, as we will show in Sec. 2.5.1. The outcome of our effort is a general framework that accelerates the computation of the OCP, is compatible with any Newton-based Nonlinear Programming (NLP) solver like [Mastalli et al., 2020] or [Farshidian et al., 2023], and has been successfully validated on a real robot. s

2.2 Introduction

Planning and control for legged systems present significant challenges due to their inherently nonlinear dynamics. Designing trajectories for such systems is a demanding task, as these trajectories must satisfy system constraints, including the robot's kinematic limits, actuator torque limits, and the constraints imposed by the surrounding environment, such as obstacles or gaps. MPC [Rawlings et al., 2022] has emerged as a powerful technique for generating complex motions in robotic systems, particularly in the domain of legged robots [Farshidian et al., 2017]. MPC approaches calculate the optimal control inputs based on the modeled system, resulting in complex constrained optimization problems that need to be solved rapidly enough to achieve online re-planning of the optimal trajectory. Recent advancements in onboard hardware computational power and the development of more efficient NLP solvers like [Mastalli et al., 2020], [Verschueren et al., 2021], and [Farshidian et al., 2023] have opened doors to solving , efficiently, complex OCP. However, the solution time is still a crucial factor to be considered when designing a control architecture for legged systems. To mitigate the computational complexity associated with the whole-body dynamics of the robot and speed up the solving time of the OCP, the works [Hong et al., 2020], [Romualdi et al., 2022], and [Rathod et al., 2021] opted to approximate the robot with the centroidal

model [Orin et al., 2013]. This approach reduces the robot's dynamics to its center of mass (CoM) motion, neglecting the influence of individual limbs on the base. Although this approximation may seem simplistic, it considers the key dynamic effects on the system. Additionally, it facilitates the integration of feasibility constraints, such as contact friction. The centroidal model has demonstrated effectiveness in stabilizing locomotion for both quadrupeds and bipeds, although it is necessary to carefully craft reference trajectories to track agile motions.

To reduce the drawbacks of such a model, [Sleiman et al., 2021] and [Grandia et al., 2023] developed an MPC formulation based on the Kino-Dynamic model, which takes into account joint position and velocity, neglects the effect of limb acceleration on the base dynamics while maintaining the effect of the robot joint configuration on the inertia matrix. As a result, the Kino-Dynamic model avoids the highly non-linear terms while simultaneously providing a better approximation than the centroidal model. In contrast, [Mastalli et al., 2023] and [Dantec et al., 2022] used the full-body model, entirely exploiting the robot's capabilities. The complexity of the OCPs considered in those implementations is remarkable, given the limited time budget available for solving them in the feedback MPC scheme. Indeed, it needs to be considered that the computational complexity of the NLP implementation increases with the number of state variables in the OCP, typically following a cubic $O(n^3)$ law [Nganga and Wensing, 2021]. This poses a significant limitation on the number of variables that can be feasibly included in the optimization process and restricts the length of the MPC horizon. Despite the continuous growth of the number of CPU cores of each generation of modern processors, the algorithms presented earlier face limitations in terms of parallelizability. This limits the effective utilization of the computational resources and restricts how the solver's solution time scales with the number of decision variables. A different paradigm that tries to reduce the computation burden is presented in [Ma et al., 2021a]. The authors worked towards a reduction of the computational effort by decoupling a quadruped robot into two bipeds. This brought a consistent reduction in the number of optimization variables that drastically decreased the computational time; nevertheless, the formulation they employed limits the approach to only the offline generation of periodic orbits, which is too slow to be computed online in a feedback MPC scheme. In contrast, [Kamidi et al., 2022] proposed a similar decoupling principle and applied it to quadratic programming-based nonlinear controllers. While the parallelizability of their approach leads to remarkable performances, particularly in terms of disturbance rejection properties inherited from the decoupling framework, their implementation requires the knowledge of the centralized system optimum. The approach is thus limited to tracking

pre-computed periodic orbits. Finally, [Ma et al., 2021b] developed local controllers based on the dynamic decomposition from [Ma et al., 2021a], which can stabilize the centralized system across various scenarios. However, their approach is not suitable for generating highly dynamic motions due to the intrinsic limitations of instantaneous controllers. The above-mentioned methods also lack easy adaptability to more complex systems, such as a quadruped robot with one or several articulated arms on top.

Our approach divides the robot into multiple subsystems while ensuring coherence between the solutions of the decoupled systems with a consensus formulation. In particular, we designed an easily scalable method to speed up the computation of the OCP on legged systems, exploiting a parallelized implementation of the ADMM. It is worth highlighting that ADMM has already been successfully used in legged robotics by [Budhiraja et al., 2019]. Their work primarily focused on rigorously enforcing consensus between simplified and whole-body optimizations within a two-step optimization framework, which is a common practice in locomotion controllers introduced by [Herzog et al., 2015]. In contrast, our approach focuses on decomposing the robot dynamics into reduced subsystems and using ADMM to ensure consensus among them. In other words, our approach boils down to running in parallel a separate MPC for each subsystem we have divided our robot into, using consensus ADMM to maintain consistency between the optimizations. This allows our algorithm to scale effectively with more complex robot configurations. For instance, the issue of integrating an articulated arm onto a quadruped is simplified to merely adding another subsystem in parallel to the others and adding the relative consensus. By adopting this methodology, our method is able to solve complex whole-body motions (Fig. 2.1), overcoming at the same time the curse of dimensionality that is normally associated with the use of the robot's full dynamics.

2.2.1 Contributions

The main contributions of this work are:

- the development of a novel scalable framework based on ADMM that reduces the computational time of the OCP problem, e.g. two times reduction for a quadruped robot and up to four times for a quadruped plus a 6 DoF arm on top, Fig. 2.2.
- a numerical evaluation of the scalability of the proposed approach to systems of increasing complexity, e.g. adding an arm on a quadruped robot does not affect the computational time.

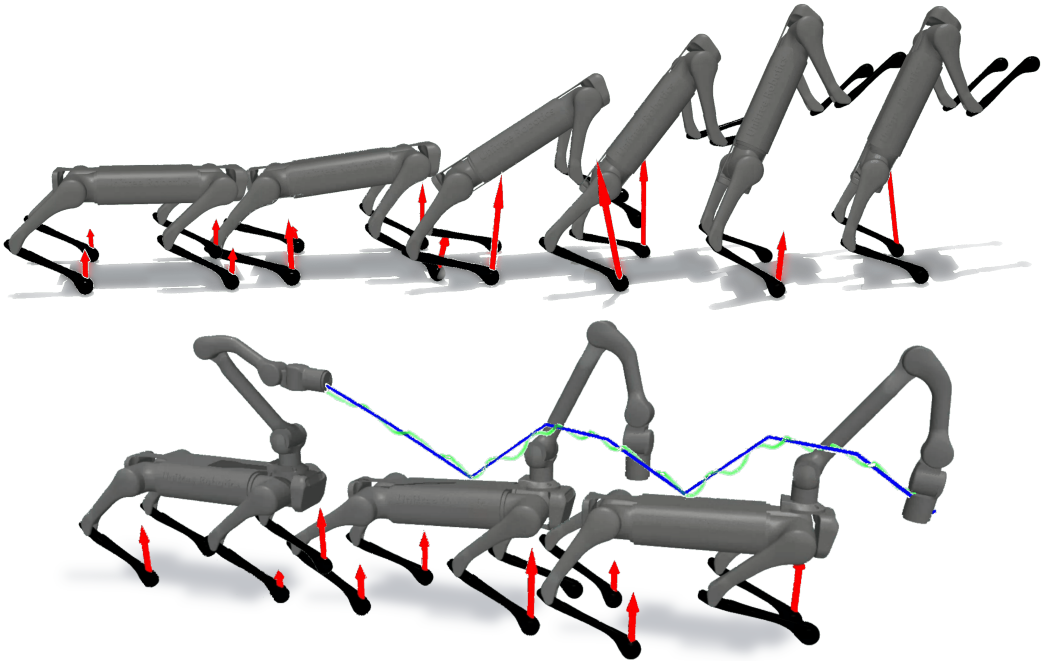

Figure 2.1 Simulation snapshots of robotic systems controlled by the proposed MPC with distributed optimization, performing different agile motions. On the top, a quadruped standing up on two feet and walking forward. On the bottom, a quadruped manipulator following a triangular spiral with the manipulator end-effector, the reference is highlighted in blue while the actual trajectory is in green.

- a quantitative evaluation, in simulation of the performance and convergence property of the proposed method against a state-of-the-art implementation of the same whole-body MPC problem.

2.2.2 Outline

This chapter is organized as follows. Section 2.3 presents the necessary background on the ADMM algorithm and MPC. Section 2.4 describes our approach in detail. Section 2.5.1 demonstrates the convergence property of our formulation, while Section 2.5.2 shows the performance in terms of computational time of our distributed whole body MPC, in a simulation environment, while Section 2.5.3 details the performed experiments. Finally, Section 2.6 and Section 2.7 draw the final considerations and conclusions.

2.3 Background

2.3.1 Alternating Direction Method of Multipliers

ADMM is an efficient approach for solving optimization problems. It has been broadly adopted in both robotics and machine learning fields [Boyd et al., 2011], [Bravo-Palacios and Wensing, 2022]. The ADMM algorithm solves problems in the form:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{z}} \quad & \boldsymbol{f}(\boldsymbol{w}) + \boldsymbol{g}(\boldsymbol{z}) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{w} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{c} \end{aligned} \tag{2.1}$$

Where $\boldsymbol{A}$,$\boldsymbol{B}$, and $\boldsymbol{c}$ are respectively the matrices and the vector that define the equality constraint. $\boldsymbol{w}$ and $\boldsymbol{z}$ are two separate decision variables. $\boldsymbol{f}(\boldsymbol{w})$ and $\boldsymbol{g}(\boldsymbol{z})$ are the two separate cost function. The augmented Lagrangian for (2.1) is:

$$\begin{aligned} \mathcal{L}_\rho(\boldsymbol{w}, \boldsymbol{z}, \boldsymbol{\lambda}) = & \boldsymbol{f}(\boldsymbol{w}) + \boldsymbol{g}(\boldsymbol{z}) + \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{w} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}) + \\ & \frac{\rho}{2}||\boldsymbol{A}\boldsymbol{w} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}||^2 \end{aligned} \tag{2.2}$$

where $\boldsymbol{\lambda}$ is the dual variable associated with the equality constraint and ρ is the penalty parameter related to the constraint violation. The ADMM exploits the separation of the cost terms between the $\boldsymbol{w}$ and $\boldsymbol{z}$ decision variables to split the problem and thus tackling smaller and simpler sub-problems at each iteration [Boyd et al., 2011]. The solution to (2.1) can be found iterating till convergence the following:

$$\begin{aligned} \boldsymbol{w}^{n+1} &= \min_{\boldsymbol{w}} \mathcal{L}_\rho(\boldsymbol{w}, \boldsymbol{z}^n, \boldsymbol{\lambda}^n) \\ \boldsymbol{z}^{n+1} &= \min_{\boldsymbol{z}} \mathcal{L}_\rho(\boldsymbol{w}^{n+1}, \boldsymbol{z}, \boldsymbol{\lambda}^n) \\ \boldsymbol{\lambda}^{n+1} &= \boldsymbol{\lambda}^n + \rho(\boldsymbol{A}\boldsymbol{w} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{c}) \end{aligned} \tag{2.3}$$

where the superfixed n indicates the iteration number. In the first two steps of (2.3), we update the two primal variables $\boldsymbol{w}$ and $\boldsymbol{z}$ sequentially, and then we calculate the dual variable; this results in a sequential implementation of the algorithm.
A widely adopted approach performs the updates of primal variables, $\boldsymbol{w^{n+1}}$ and $\boldsymbol{z}^{n+1}$ in parallel. Parallelization is achieved at the expense of both the convergence rate and convergence guarantees, prioritizing on the other hand computational speed. [Deng et al., 2017] showed that including additional regularization terms in the cost partially recovers convergence rate and guarantees. In our method, we will use a special case of ADMM

called *consensus* ADMM [Boyd et al., 2011], in which the problem is in the form:

$$\begin{aligned} &\min_{\bar{\boldsymbol{y}},\boldsymbol{y}_1,\boldsymbol{y}_2,\dots,\boldsymbol{y}_{N_{sys}}} \sum_{i=1}^{N_{sys}} \boldsymbol{f}_i(\boldsymbol{y}_i) \\ &\text{s.t. } \boldsymbol{y}_i = \bar{\boldsymbol{y}}_{\boldsymbol{i}}, \quad i = 1,\dots,N_{sys} \end{aligned} \tag{2.4}$$

where N_{sys} is the number of subsystems we split the main problem into. $\boldsymbol{y}_i$ is a local copy of a subvector of the global decision variable $\bar{\boldsymbol{y}}$. The objective in this formulation is to optimize $\bar{\boldsymbol{y}}$ through the separate optimization of the $\boldsymbol{y}_i$ variables. In this way, we transform the centralized optimization involving only $\bar{\boldsymbol{y}}$ into a network of smaller local independent problems. In our implementation, we will divide the robot model into separate independent OCP, which can be solved in parallel. The decision variables and constraints of the global problem are reduced to the ones related to the considered subsystems, making the problem much smaller. The consensus is then established between the optimizations, to avoid non-physical behaviors.

2.3.2 Model Predictive Control

In this section, we will first briefly introduce the dynamic model we are using and then we will present the OCP. The efficacy of the control input derived from an MPC strategy is directly linked with the accuracy of its predictions. Carefully modeling the dynamics of the robot is necessary to maximize its performance. Therefore, we have chosen to employ the articulated rigid body dynamics [Featherstone, 1987] with hard contacts to characterize the dynamics of a legged robot accurately. We define the state of the robot at instant k by the generalized coordinates $\boldsymbol{q}_k \in R^{n_q}$ and the generalized velocity $\boldsymbol{v}_k \in R^{n_v}$. We write the equation of motion of legged robots in discrete time following a semi-implicit Euler scheme as:

$$\begin{aligned} \boldsymbol{v}_{k+1} &= \boldsymbol{v}_k - \boldsymbol{M}^{-1}(\boldsymbol{b} - \boldsymbol{S}\boldsymbol{\tau} - \boldsymbol{J}^T\boldsymbol{\lambda}_{\boldsymbol{grf}})dt \\ \boldsymbol{q}_{k+1} &= \boldsymbol{q}_k \oplus \boldsymbol{v}_{k+1}dt \end{aligned} \tag{2.5}$$

where $\boldsymbol{M} \in R^{n_v \times n_v}$ and $\boldsymbol{b} \in R^{n_v}$ are respectively the mass matrix and the "bias" vector that includes the Coriolis, centrifugal, and gravitational terms. The dependency of $\boldsymbol{M}$ and $\boldsymbol{b}$ from $\boldsymbol{q}_k$ and $\boldsymbol{v}_k$ has been dropped in favor of readability. The variable $\boldsymbol{\tau} \in R^{n_u}$ corresponds to the joint torque, while $\boldsymbol{S} \in R^{n_v \times n_u}$ is a selector matrix. Matrix $\boldsymbol{J} \in R^{n_v \times n_c}$ is the stack of jacobian associated with each contact and $\boldsymbol{\lambda}_{\boldsymbol{grf}} \in R^{n_c}$ is the stack of ground

reaction forces. Considering the dynamics described before, we can write an MPC that solves the following OCP in a receding horizon fashion:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \boldsymbol{l}_T(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} \boldsymbol{l}(\boldsymbol{x}_k, \boldsymbol{u}_k) \tag{2.6a}$$

$$s.t. \quad \boldsymbol{v}_{k+1} = \boldsymbol{v}_k - \boldsymbol{M}^{-1}(\boldsymbol{b} - \boldsymbol{S}\boldsymbol{\tau} - \boldsymbol{J}^T\boldsymbol{\lambda}_{grf})dt \tag{2.6b}$$

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k \oplus \boldsymbol{v}_{k+1}dt \tag{2.6c}$$

$$\boldsymbol{J}\boldsymbol{v}_k = \boldsymbol{v}_{\boldsymbol{e}} \tag{2.6d}$$

$$\boldsymbol{p}_{f,k}|_z - \boldsymbol{p}^{ref}_{f,k}|_z = 0 \tag{2.6e}$$

$$\boldsymbol{u}_k \in \mathsf{U}_k \tag{2.6f}$$

$$\boldsymbol{x}_k \in \mathsf{X}_k \tag{2.6g}$$

$$\boldsymbol{x}_0 = \hat{\boldsymbol{x}}_0 \tag{2.6h}$$

where $\boldsymbol{x}_k = [\boldsymbol{v}_k, \boldsymbol{q}_k]^T \in R^{n_v+n_q}$ and $\boldsymbol{u}_k = [\boldsymbol{\tau}_k, \boldsymbol{\lambda}_k]^T \in R^{n_u+n_c}$ are respectively the state and control input. $\boldsymbol{l}(\boldsymbol{x}(\cdot))$ is a quadratic cost including a tracking term and a regularization one. $\boldsymbol{J}\boldsymbol{v}_k = \boldsymbol{v}_e$ slacks the non-slipping condition for each contact to avoid numerical instability [Baumgarte, 1972]. $\boldsymbol{p}_{f,k}|_z$ is the position of the foot along the z-axis and it is constrained to guarantee that the legs in stance are touching the ground. U_k is the set of feasible control input limits by the friction cone and torque limits at the joint. The set X_k represents the feasible states constrained by the joint kinematic limits. $\hat{\boldsymbol{x}}_0$ is the state initial condition.

The described nonlinear optimization problem poses significant challenges when used for controlling legged robots since, due to their complexity, they are difficult to solve at high frequencies. Even with the latest advancements highlighted in [Mastalli et al., 2023], the update rate of the final MPC is limited to 50Hz. Our approach aims to enhance the capabilities of modern solvers by diving the OCP into independent problems with a reduced number of decision variables that are solved in parallel. For instance, decoupling a quadruped into two parts, as illustrated in Fig. 2.2, results in a reduction with respect to the centralized approach by 25% of the number of variables in each subsystem's optimization. While the number of variables of the system incorporating also the additional arm on top is reduced by 40%. More details on the computational time gains derived from this reduction are provided in Section 2.5.2.

2.4 Distibuted Whole Body MPC

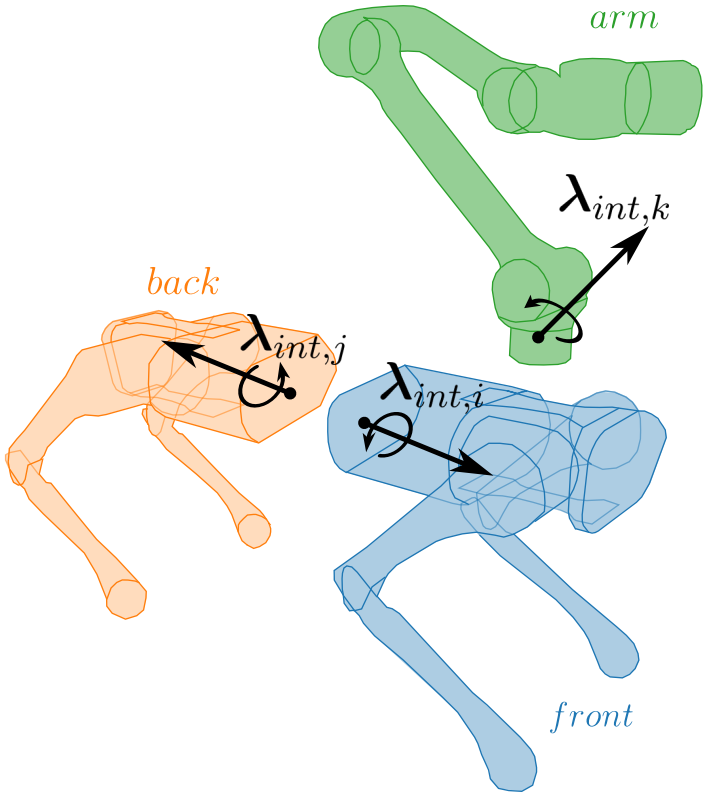

Figure 2.2 Schematic of a quadruped robot with an articulated arm on top split into three separate subsystems. Each section of the robot sees the dynamic effect of the other parts through the interaction wrench $\boldsymbol{\lambda}_{int}$.

The main contribution of this work is to leverage the efficiency and parallelizability of the consensus ADMM to speed up the solution time of the NLP in (2.6). Our method provides a framework for partitioning the original OCP into smaller, independent problems while ensuring consensus among the solutions. Through this approach, we can significantly decrease the computational time while augmenting our system's complexity, such as incorporating an articulated arm onto a quadruped robot, without significantly impacting the solution time, as reported in Section 2.5.2. We can effectively decouple the robot into distinct subsystems, as illustrated in the example in Fig. 2.2. The consensus ADMM implementation is used to enforce the consistency of the dynamic between the different subsystems and prevent them from drifting apart. Without consensus, we will only have decoupled independent systems that are no longer solving together the centralized OCP in (2.6).

2.4.1 Decomposed dynamics

The decomposition of the robot dynamics is based on the idea of considering the separated subsystems interacting with each other through a wrench $\boldsymbol{\lambda}_{int}$. For the i^{th} subsystem the wrench $\boldsymbol{\lambda}_{int,i}$ synthesizes the dynamic effect of the other j subsystems that we are splitting the robot into. The equations of motion of the i^{th} subsystem also considering the rigid body connection constraint are written as:

$$\boldsymbol{v}_{i,k+1} = \boldsymbol{v}_{i,k} - \boldsymbol{M}_i^{-1}(\boldsymbol{b}_i - \boldsymbol{S}\boldsymbol{\tau}_i - \boldsymbol{J}_i^T\boldsymbol{\lambda}_i - \boldsymbol{J}_{int,i}^T\boldsymbol{\lambda}_{int,i})dt \tag{2.7a}$$

$$\boldsymbol{J}_{int,i}\boldsymbol{v}_{i,k} = \boldsymbol{J}_{int,j}\boldsymbol{v}_{j,k} \quad \forall j \neq i \tag{2.7b}$$

for $i, j \in \mathcal{X}$ where $\mathcal{X}$ is the set of subsystems we have divided our robot into. $\boldsymbol{J}_{int,i}$ is the Jacobian that maps the generalized velocity of the subsystems into the velocity at the interface with the other systems. Manipulating (2.7) we can write $\boldsymbol{\lambda}_{int,i}$ as:

$$\boldsymbol{\lambda}_{int,i}dt = (\bar{\boldsymbol{J}}_{int}\bar{\boldsymbol{M}}^{-1}\bar{\boldsymbol{J}}_{int}^T)^{-1}\Delta v_{tot,k} \tag{2.8}$$

Here, $\bar{\boldsymbol{M}} = diag(M_0, \ldots, M_{N_{sys}})$ and $\bar{\boldsymbol{J}}_{int} = \begin{bmatrix}\boldsymbol{J}_{int,0} & \ldots & \boldsymbol{J}_{int,N_{sys}}\end{bmatrix}^T$. Δv_{tot} represent the sum of the effect of all systems and is defined as:

$$\Delta v_{tot} \equiv \sum_{*\in\mathcal{X}} \boldsymbol{J}_{int,*}\left(\boldsymbol{v}_{*,k} + \boldsymbol{M}_*^{-1}(\boldsymbol{b}_* - \boldsymbol{S}\boldsymbol{\tau}_* - \boldsymbol{J}_*^T\boldsymbol{\lambda}_{grf,*})dt\right) \tag{2.9}$$

It is worth noting that Δv_{tot} is obtained by applying the dynamic of (2.7) to each subsystem composing the considered body. The obtained coupling wrench in (2.8) depends on all subsystem's generalized velocity and coordinates. Given the parallelization employed in this work, it is not possible to have the j^{th} subsystems variables beforehand. This forced us to utilize the past information at algorithm iteration n when calculating the $n+1$ update for the i^{th} subsystem as done in the ADMM. Such an approximation, deriving from the subsystem decoupling, does not affect the optimization performances, as empirically demonstrated in section 2.5.1. Indeed, such an approximation is comparable to the one already used in centralized MPC employing Newton-base NLP solvers, which linearized at each iteration based on the previous optimal solution [Gros et al., 2020]. Finally, the subsystem dynamics in (2.7a) can be rewritten explicating the dependency of $\boldsymbol{\lambda}_{int,i}$ as:

$$\begin{aligned}\boldsymbol{v}_{i,k+1}^{n+1} =& \boldsymbol{v}_{i,k}^{n+1} - \boldsymbol{M}_i^{-1}(\boldsymbol{b}_i - \boldsymbol{S}\boldsymbol{\tau}_i^{n+1} - \boldsymbol{J}_i^T\boldsymbol{\lambda}_{grf,i}^{n+1} + \\ -\boldsymbol{J}_{int,i}^T\boldsymbol{\lambda}_{int,i}(&\boldsymbol{q}_i^{n+1}, \boldsymbol{v}_i^{n+1}, \boldsymbol{\lambda}_{grf,i}^{n+1}, \boldsymbol{\tau}_i^{n+1}, \boldsymbol{q}_j^n, \boldsymbol{v}_j^n, \boldsymbol{\lambda}_{grf,j}^n, \boldsymbol{\tau}_j^n))dt\end{aligned} \tag{2.10}$$

Equation (2.10) now only depends on the i^{th} system current decision variables, while the j^{th} subsystem's influence only appears as a parameter coming from the last n^{th} iteration. The system dynamics are reduced to only the floating base plus the joints considered in the i^{th} subsystem. We have lost the direct dependency on the other joints, but we gained a model of reduced size that can be utilized in our distributed MPC.

2.4.2 Distributed optimization

In this subsection, we will delve into the formulation of the distributed OCP. The peculiarity of such implementation lies in the fact that we are going to solve (2.6) defining N_{sys} optimization problems, one per each subsystem we are dividing the robot into. Such problems are smaller than the problem of (2.6), and can be solved in parallel, thus speeding up the computation. Each subsystem optimization problem is defined utilizing the system dynamic in (2.10) and the constraint (2.7b). Such constraint serves to ensure coherence between velocities at the interface of the subsystems, thereby preventing them from drifting apart. Since we are decoupling the subsystems to enable parallel optimization, analogously to what we did for the decomposed dynamics, (2.7b) cannot be directly used, but it can be formulated as a consensus problem. Defining the residual between the subsystems i and j at the time k as :

$$\boldsymbol{r}_{i,j} = \boldsymbol{J}_{F,i}\boldsymbol{v}_i - \boldsymbol{J}_{F,j}\boldsymbol{v}_j \tag{2.11}$$

it is possible to formulate each single subsystem optimization. We take the consensus ADMM implementation illustrated in section 2.3.1 and tailor it for the OCP in (2.6). For convenience, we utilized the scaled dual variable $\bar{y} = y/\rho$. In this way, the Lagrangian in (2.2) is reduced to its quadratic term. We defined the cost for the subsystem i as:

$$\begin{aligned} f_i = \boldsymbol{l}_T(\boldsymbol{x_i}(N)) + \sum_{k=0}^{N-1} \Big[\boldsymbol{l}(\boldsymbol{x}_{k,i}, \boldsymbol{u}_{k.i}) + \\ + \sum_{i,j \in \mathcal{X}} ||\boldsymbol{r}_{i,j} - \bar{\boldsymbol{\lambda}}^n_{i,j}||^2_\rho \Big] + ||\boldsymbol{x}_i - \boldsymbol{x}^n||^2_\sigma \end{aligned} \tag{2.12}$$

where the first two terms are the components of the original cost in (2.6) that depend on the i^{th} subsystem decision variables. ρ and σ are positive weight parameters that penalize consensus violations and regularize the solution for the whole-body. The last term of (2.12) introduced by [Deng et al., 2017] works as a regularizer towards the optimal values found at the previous iteration and it has proven to be crucial for increasing the numerical stability of the algorithm.

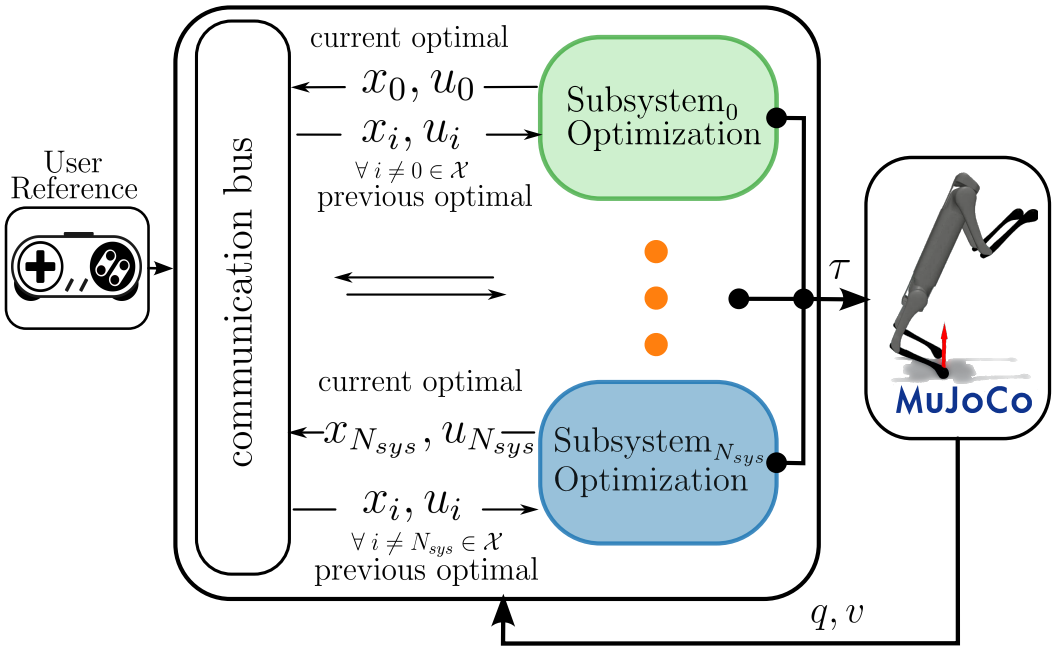

Figure 2.3 Block scheme of the proposed control framework highlighting the communication bus between the decomposed Subsystem Optimizations.

Thus, the global OCP in (2.6) is reduced to the iterative solution, executed in parallel, of the local OCPs:

$$\min_{x^i, u^i} \; f_i \quad \text{s.t. eq. (2.10)} \,, \text{eqs. (2.6c) to (2.6h)} \tag{2.13}$$

where only the states and control corresponding to the i^{th} subsystem appear as a decision variable. The dynamics constraint in (2.6b) is replaced by (2.10), the cost function is the one shown in (2.12), and the constraint form (2.6) relative to the subsystem's variables are passed to the local OCP. During each iteration of the distributed algorithm, we perform a single full Newton step for each subsystem optimization before updating the dual variable. An outline of the implementation is reported in Algorithm (1), where the **Map**() function only extracts the subsystem variables from the global one. As for [Zhou and Zhao, 2020] we defined the stopping criteria of our algorithm based on the l_2-norm of the residual:

$$||r^n||_2 < \epsilon \tag{2.14}$$

where ϵ is a positive scalar threshold on the residual. Our algorithm does not come with convergence guarantees, however, similar to other ADMM-based implementations on nonconvex problems such as those in [Zhou and Zhao, 2020] and [Budhiraja et al., 2019], the solver has empirically proven to be reliable, as shown in Section 2.5.1.

Algorithm 1 Distributed Whole-body Optimization

Data: x^{guess}, u^{guess}
repeat
  for subsystems **do** in *parallel*
    x_i^n, $u_i^n \longleftarrow$ **Map**(x^{guess}, u^{guess})
    x_i^{n+1}, $u_i^{n+1} \longleftarrow$ **SubSystemSolver**(x_i^n, u_i^n) (2.13)
  end for
  $\bar{\lambda}_{int}^{n+1} = \bar{\lambda}_{int}^{n} + r^{n+1}$
  x^{guess}, $u^{guess} \longleftarrow x^{n+1}, u^{n+1}$
until stopping criterion
return x^{n+1}, u^{n+1}

2.4.3 Distributed MPC

For the online execution of our framework, instead of running the described algorithm to convergence at each control loop, we implemented a receding horizon scheme. In this scheme, each local solver receives the last optimal solution from the other subsystems and the feedback from the robot. Then, we execute one Newton step for each subsystem and apply only the torque calculated at the first step of the horizon to the robot. We then shift the rest of the solution by one time-step and send it back to the communication bus to update the dual variable and provide the current local optimal to the other optimizations for the next control loop. A schematic of the control framework used in the simulation is presented in Fig. 2.3.

2.5 Result

In this section, we present the results of different analyses to demonstrate the benefits of the proposed approach. We analyze the performance of our algorithm considering two different systems: a quadruped robot and a quadruped with a manipulator on top, which we will call a *quadruped manipulator*. In the first case, we split the robot into two parts: front and back. All subsystems include a floating base and the decision variables linked with the considered joints, e.g. the front section will consider only the front legs' joint torques, positions, velocities, and the corresponding ground reaction force. This reduces the decision variables per stage from R^{61} to R^{37}, since we are discarding the 24 states and control linked with the joints included in the other half of the robot. Thanks to this reduction in the state we can solve the local OCP faster, as exemplified by the case of the quadruped manipulator, where the distributed approach can run 4 times faster than the

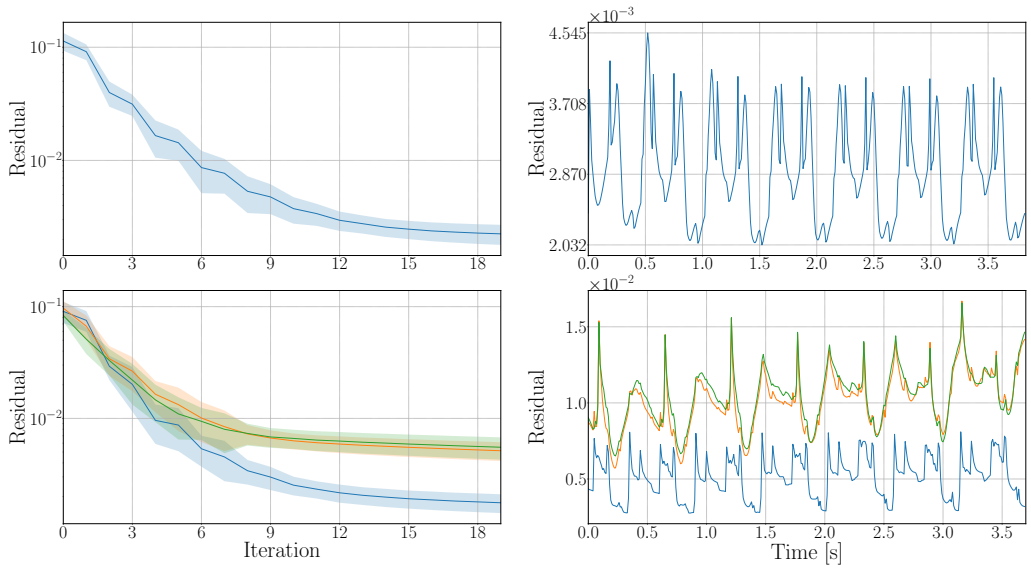

Figure 2.4 The two plots on the left show the trend of the l_2 norm of the residuals along the iteration of Algorithm 1. On top is the residual for the quadruped with no arm, while on the bottom are the residuals for the quadruped manipulator. The right side plots show the time plot of the residual norm while the robot is trotting in simulation. Again, the top plot is for the robot with no arm, and the bottom one is for the quadruped manipulator.

centralized one. Indeed, as depicted in Fig. 2.6, for the quadruped manipulator case, the solving time is on average 10 ms for the distributed case, while for the centralized one is 40 ms.

In the case of the quadruped manipulator, we include a third subsystem in the optimization consisting of the articulated arm, as shown in Fig. 2.2. In our example, the articulated arm has 6 DoF and thus it adds 18 decision variables per node (joint position, velocity, and torque for each DoF) to the centralized optimization problem. In our formulation, on the other hand, the arm is treated as an independent third optimization with 31 decision variables executed in parallel to the other two. It is worth remembering that the complexity of the optimization scales with the cube of the number of stage decision variables. Indeed, a naive centralized implementation is cursed by the increased dimension of the problem, while our implementation utilizes unexploited computational power by parallelizing the additional subsystem optimization.

In the remainder of this section, we will first focus on the convergence analysis of the proposed approach, showing the residual trend with both systems and comparing our distributed MPC approach with a centralized MPC implementation. Subsequently, we will demonstrate the benefits of our approach in terms of solving time and provide examples of agile maneuvers that are possible thanks to the whole-body implementation. The distributed

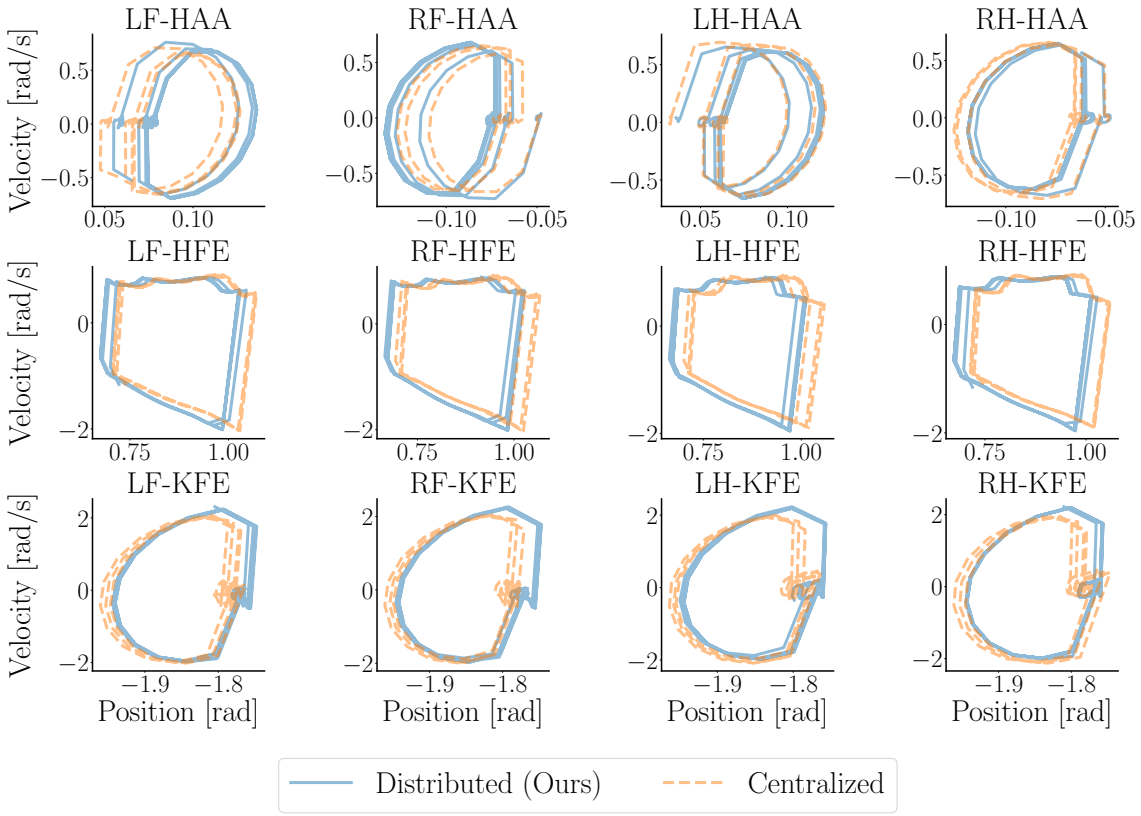

Figure 2.5 Phase plots of the robot trotting with the centralized (dashed blue line) and distributed (orange line) solutions. The plot shows a recording of the robot trotting in simulation at a desired speed of 0.3 m/s. Where RF-LF-LH-RH stands for Right Front, Left Front, Left Hind, Right Hind and, HAA, HFE, KFE stand for Hip Abduction/Adduction, Hip Flexion/Extension, and Knee Flexion/Extension.

optimization approach and the centralized one have been implemented using Adam [L'Erario et al., 2022], an open-source library for rigid-body dynamics and Acados [Verschueren et al., 2021], an open-source Sequential Quadratic Programming solver tailored for optimal control. We chose to utilize Acados for its ease of use, but our method is not tailored to any specific solver and can be reused with other NLP solvers. Both formulations utilize a prediction horizon of 50 nodes with a discretization time of 0.01 seconds, and they are both run at 50Hz for the sake of comparison. All the presented results have been obtained using an Intel Core i7 13700H laptop CPU. For the simulations, we utilized Mujoco [Todorov et al., 2012] as physics engine.

2.5.1 Convergence analysis

In this subsection, we aim to empirically analyze the convergence properties and stability of our algorithm. We begin by examining the l_2 norm of the residual, as calculated in (2.11). This metric provides insight into the quality of the consensus achieved. Fig. 2.4 illustrates the trend over the iterations of Algorithm 1. As previously explained, we consider two scenarios: one with the quadruped split in half and the other with the robot plus the arm. It should be noted that in the former case, the residual is one, while in the latter case, they become 3, indicating consensus among the three subsystems. The plots depict the average over one hundred simulations, with the system initialized in different configurations and with randomized warm starts. The variance is represented by the colored area around the mean value. As shown, the residual converges within a few iterations to small values. Notably, the value the residual reaches corresponds to an error between the subsystems at the end of the predicted trajectories in the orders of millimeters or less. In the second column of Fig. 2.4, we display the trend of the residual norm of the MPC. In this case, we reported how the residual value remains in the order of 10^{-3} when the distributed approach is utilized in a receding horizon fashion. The data was recorded while the robot was trotting in simulation following a desired speed of $0.3\frac{m}{s}$.

Finally, in Fig. 2.5 we compare, in simulation, the centralized and the distributed approach, reporting the phase plot for each joint. The figure illustrates that the steady-state solutions of both methods converge close to the same values, depicting how our solver response is the same as one from a centralized implementation.

2.5.2 Performance evaluation

In this subsection, we show the main benefits of our implementation. As displayed in Fig. 2.6 we compared the solving time of the distributed and centralized implementation for the quadruped and quadruped manipulator. We showed the occurrence of the solving times recorded in a 5 second simulation reporting in the histogram the relative frequencies. In simulation, the robot is asked to follow a desired velocity of $0.3m/s$ and is perturbed repeatedly by an external force of random direction and magnitude in the range between $\pm 30N$ for 0.1s each time. The graph shows that the solving times for the distributed controller are on average half of the ones of the centralized implementation for the quadruped robot while for the quadruped manipulator, the solving time is reduced by a factor of four.

It is worth noting that our implementation is not affected by the increased complexity of the system. Adding the arm on the quadruped robot does not increase the computational

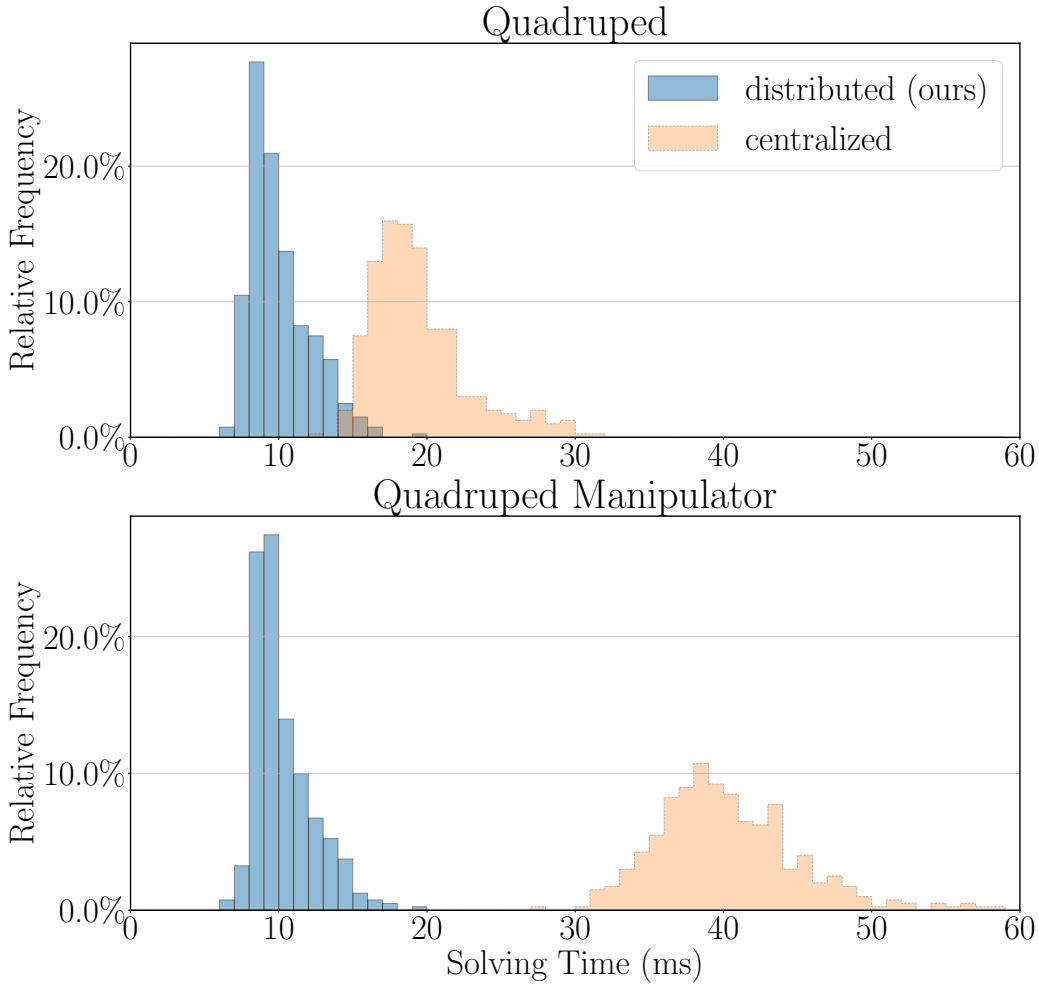

Figure 2.6 Relative frequencies for the occurrence of the solution time of the receding horizon problem in (2.6) recorded over one hundred simulations. In blue is our distributed implementation while in orange is the centralized one. On the top, the solution refers to the quadruped robot model while on the bottom plot the quadruped plus the arm.

time. It should be mentioned that the times reported for the distributed implementation are the one of the slowest subsystem optimizations. This consideration remains valid, as long as the new system we want to include can be considered as an additional subsystem that can be processed in parallel as it has been done for the arm. As already mentioned, even though the reported times refer to our implementation using Acados, changing the solver at the core of our implementation will not affect the relative gains, but only the absolute numbers in the same way for both centralized and distributed approaches.

Finally, to demonstrate the benefit of a whole-body implementation and the robustness of our formulation, we show the robot performing complex maneuvers that need whole-body coordination. In the top part of Fig. 2.1 we show the quadruped stand-up and walk on

two feet. No particular effort is required in designing the reference trajectories since the MPC can generate and stabilize agile maneuvers starting from simple unfeasible trajectories. Indeed, for the emergence of the biped walking behavior, it was sufficient a step function for the robot's desired base pitch equal to $\pi/2$ and a reference constant velocity to the base equal to $0.2m/s$.

The bottom part of Fig. 2.1 shows the quadruped manipulator trotting while its end-effector is tracking a triangular spiral defined in the world frame. In this case, the only reference we are passing to the MPC is the desired spiral trajectory for the arm's end-effector.

Both simulations can be seen in the accompanying video of [Amatucci et al., 2024a].

2.5.3 Experiments

We experimentally tested the proposed distributed MPC on an Aliengo robot from Unitree Robotics, a 23Kg platform. The entire locomotion pipeline, MPC plus state estimator [Nisticò et al., 2025], runs onboard the robot on an Intel NUC with a 10th-gen Intel i7 CPU. The distributed MPC runs at 150 Hz, while the low-level joint position controller runs at 1 KHz. The robot is commanded by an operator with a joystick that sets the desired heading, lateral, and turning velocities. The gait sequence is predefined and set by a gait timer. In particular, in the shown experiments, the robot is tasked to crawl with a step frequency of 0.6 Hz and a duty factor of 0.8. The course consists of a stair with 10 cm high steps and a rough terrain section with randomly disposed rocks that can move and tumble. Both the estimator and the controller only use proprioceptive data. The controller corrects for the early and delayed contact event thanks to a contact detection routine that adjusts the foot reference accordingly. The step height is manually set by the operator to ensure the clearance of the obstacle. As shown in Fig. 2.7, the system is able to walk on the course with ease.

2.6 Discussion & Limitation

In this section, we discuss some considerations and limitations of the approach that may not be immediately evident from the reported results and therefore, in our opinion, should be highlighted. At first glance, it may seem natural to extend the presented example by further splitting the robot dynamics (potentially down to the point where each degree of freedom is controlled by its own MPC) to maximize performance improvements. While this would

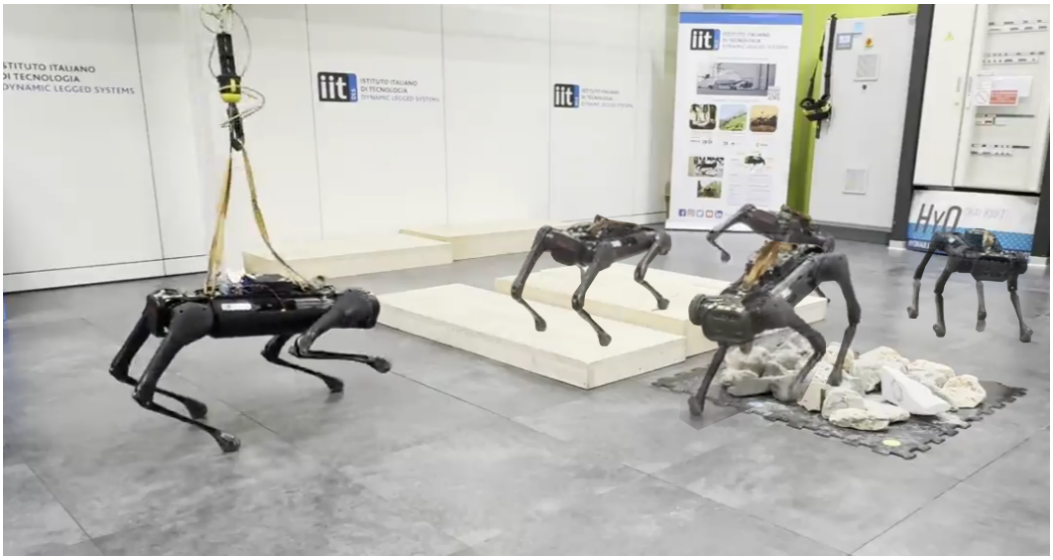

Figure 2.7 Snapshot of the real quadruped walking on stairs and rough terrain

indeed yield the best theoretical solving times, we must remember that such aggressive decomposition increases the level of approximation and places a greater burden on the consensus loop. For instance, we have assessed modeling each leg as a separate subsystem and, in practice, found the solver to be significantly more prone to breaking, especially during highly dynamic maneuvers. Another important consideration is that the consensus mechanism introduces additional elements that must be tuned in the cost function. In our experience, these parameters exhibit high sensitivity and can strongly affect the stability of the solution. For example, in the quadruped case where the robot was split into two subsystems, we observed that the range for the residual's weight ρ in (2.12) for which the solver remained stable was quite narrow, roughly between 0.5 and 10. A more aggressive partition would inevitably also narrow down the interval. Finally, although we did not explore it in this work, it would be interesting to study how different partitioning strategies affect the stability of the solution. From our empirical observations, the partition choice should be made with the specific task in mind to maximise the robustness of the approach. For example, with the front–back robot partition, we observed that during bounding (where contacts alternate between the front and rear legs), the residual tends to be smaller than in gaits where both subsystems make contact simultaneously, such as trot or pace. Our intuition is that this effect comes from an overcompensation mechanism generating a positive feedback loop between our subsystems. At the first iteration, both subsystems try to support the entire weight of the robot since the forces are initialized to zero. As a result, the interface wrenches they exchange become larger than they should be. At

the next iteration, each subsystem sees these inflated wrenches and reacts by increasing its corresponding Ground Reaction Forces (GRF)s even further. This creates a positive feedback loop: larger forces lead to even larger corrections in the next step, which may eventually lead the optimization to diverge. The only element preventing this divergence is the consensus term in the cost function, which forces the subsystems to agree. In contrast, during bounding, each subsystem GRFs are active at different times. This naturally reduces interference between the subsystems and makes the approximated dynamics closer to the true WB-dynamics of the entire robot, since each subsystem needs to account only for ballistic motion when it is out of its contact phase.

2.7 Conclusion

This chapter presented a novel approach to decomposing the robot dynamics into reduced and more tractable subsystems to accelerate MPC for legged robots. Utilizing ADMM, we ensure consensus among the parallel subsystems' optimization, demonstrating significant improvements in computational time. The parallelizability of our method facilitates the integration of additional limbs such as articulated arms without compromising the solving time, and thus making the system easily scalable. Through extensive numerical simulations, we have validated the convergence and performance of our approach across two systems of varying complexity, highlighting its effectiveness compared to state-of-the-art whole-body MPC implementations. The quantitative simulations have underscored the substantial reduction in computational time; in particular, for the quadruped model we gained a reduction of two times, while for the quadruped manipulator, the achieved reduction is four times. These gains derive from the fact that we have added a new subsystem to be solved in parallel with the others, so the computational time for our approach remains constant, while for a centralized whole-body optimization, it drastically increases due to a higher number of decision variables per node. These gains in computational time come from our more effective use of modern hardware resources.

Chapter 3

Non-gaited Legged Locomotion with Monte-Carlo Tree Search and Supervised Learning

I. Taouil*, L. Amatucci*, K. Majid, A. Dai, V. Barasuol, G. Turrisi, C. Semini "Non-Gaited Legged Locomotion With Monte-Carlo Tree Search and Supervised Learning" in IEEE Robotics and Automation Letters, vol. 10, no. 2, pp. 1265-1272, Feb. 2025, doi: 10.1109/LRA.2024.3519908. *Equal contribution

3.1 Motivation

This chapter presents a novel approach, built upon our previous work [Amatucci et al., 2022], that enables non-gaited legged locomotion by combining Monte Carlo Tree Search (MCTS) with a learned Value Function (VF) to achieve real-time gait planning on a quadrupedal robot. We decided to tackle the contact-timing problem since finding appropriate contact sequences is a crucial feature for both manipulation and locomotion. A robot must be able to decide when and where to make and break contacts, but this decision-making process is discrete in nature and therefore difficult to handle with standard optimization-based approaches. As we briefly discussed earlier, trying to properly handle contacts within the Optimal Control Problem (OCP) leads to violations of the Linear Independence Constraint Qualification (LICQ), and consequently to the loss of optimality guarantees. Our proposed solution is to remain in the discrete decision domain and cast the problem as a tree search over the space of possible contact sequences. However, the combinatorial nature of the

problem causes the search space to grow exponentially with the number of contacts and the length of the time horizon. Furthermore, evaluating the cost at each node is non-trivial, as it depends not only on the current robot state but also on the downstream effects that the current decision has on future decisions. To address this challenge, we use MCTS as a sampling-based method to explore the tree and find a solution within a limited time budget. To evaluate each node, we solve a series of Quadratic Programming (QP) approximations of the OCP, effectively simulating potential future contact sequences. However, even with a fully parallelized MCTS, the time required to obtain a solution remains too high for real-time deployment on hardware. To overcome this limitation, we employ supervised learning to train a VF that evaluates the cost at each node of the tree, limiting the number of QP to be solved at every sampled contact sequence. This approach significantly reduces the computation time required to find a solution, making it feasible to execute online on real hardware. For this work, which only focused on locomotion tasks, we demonstrated through extensive simulation analysis and experimental tests how this approach can bring improvements in motion tracking and locomotion robustness.

3.2 Introduction

Legged robots traverse the world by making and breaking contact with the environment. In doing so, they need to decide over a set of discrete and continuous optimization variables, e.g., the sequence of end-effectors establishing contact (discrete) and the contact timings and forces (continuous). Classically, the switching nature of the problem has been relaxed to cast the motion planning problem as a general Nonlinear Programming (NLP) [Tassa et al., 2012], [Mordatch et al., 2012], [Winkler et al., 2018]. However, these approaches are prone to poor local minima, and they usually need a good warm-start to yield physically plausible trajectories. Hence, they have shown little success in the real world [Koenemann et al., 2015], [Neunert et al., 2018]. Recent works have focused on handling contact as an explicit phenomenon formulating an optimization problem with a mixture of continuous and discrete optimization variables. While recent advances have enabled solving the continuous optimization for a given contact sequence in a receding horizon fashion [Meduri et al., 2023], [Mastalli et al., 2022], [Grandia et al., 2023], solving the hybrid problem online is still out of reach.

One interesting approach to solve the hybrid optimization problem is to cast it as a Mixed-Integer Program (MIP) [Deits and Tedrake, 2014] for kinematically feasible footstep planning on uneven terrain. Several extensions of this algorithm were later proposed

to solve for the gait sequence [Aceituno-Cabezas et al., 2017], or to include the robot dynamics [Ponton et al., 2021]. All of these works used a convex relaxation of the dynamics and environment geometry to make a fast resolution of the problem tractable. Later works have tapped into relaxing the integer optimization to an L1 norm minimization problem [Tonneau et al., 2020] and managed to solve the problem of contact patch selection in real-time for quadrupedal locomotion [Corbères et al., 2023]. However, in the general case of nonlinear dynamics and complex geometry of the environment, solving a non-convex MIP is an NP-hard problem and existing solvers do not scale well with the number of discrete decision variables.

In [Bratta et al., 2024], the authors demonstrated the potential of using supervised learning to determine the contact scheduling and positioning for a quadrupedal robot. Recent advancements in Reinforcement Learning (RL) offered a novel point of view on the timing problem. RL-based controllers do not explicitly tackle the problem of contact timing. Instead, during the learning process, they automatically discover behaviors that involve changes in contacts, embedding them in the network representation [Miki et al., 2022]. Nevertheless, most RL-based controllers incorporate a gait timer as an input to the network, which imposes a bias on the selected contact pattern [Hwangbo et al., 2019]. While these approaches have shown incredible performance on real hardware, heavy reward engineering is required to generate new motions and an ad-hoc domain randomization procedure is required for successful sim-to-real.

As reviewed above, MIP-based approaches efficiently use our knowledge about the dynamics and geometry of the robot, world, and corresponding constraints to generate motions in various scenarios. However, they sacrifice computational efficiency for the sake of optimality. and it is difficult to incorporate machine learning to offload some of the online computation. On the other hand, RL-based methods rely on sampling in simulation to find an optimal control policy, which ignores all our knowledge about the structure of the problem. To benefit from the power of each method, a framework that can efficiently use the structure of the problem and use machine learning to reduce the online computation is required. One approach is to use continuous RL algorithms for contact planning [Tsounis et al., 2020, Li et al., 2023]. However, these approaches ignore this structure that the contact planning problem is mostly a decision-making over *discrete* variables.

MCTS has recently emerged and shown promise as an alternative to solve the hybrid optimization problem for locomotion [Amatucci et al., 2022] and object manipulation [Zhu et al., 2023]. Both of these works have shown that MCTS can dramatically reduce the computation time in comparison with MIP, while slightly compromising the optimality of the

solution. Recently, [Dhédin et al., 2024, Akizhanov et al., 2024] used a MCTS formulation to select which surface to step onto, among all steppable surfaces. However, none of these works have managed to run MCTS in real-time on real hardware to adapt the discrete optimization variables reactively.

3.2.1 Contributions

In this work, we propose a novel approach that builds on [Amatucci et al., 2022] that leverages MCTS to optimize the gait sequence and timings for quadrupedal locomotion, combining it with offline learning [Bertsekas, 2024], [Lembono et al., 2020], [Omar et al., 2023], [Wang et al., 2024] to make it applicable in real-time. In particular, our core contributions are the following:

- We introduce a simple and effective learning-based strategy to enhance the real-time capability of an MCTS algorithm for the purpose of non-gaited locomotion.
- We carry out an extensive analysis of the MCTS parameters and their influence on the robot's performance.
- We demonstrate, to the best of our knowledge, the first-ever successful real-time implementation of a sampling-based method for non-gaited locomotion on a real quadruped robot.

The rest of the chapter is organized as follows: Sec. 3.3 introduces the vanilla MCTS algorithm, providing a foundation and context for our research. Sec. 3.4 presents a detailed and extensive analysis of the MCTS parameters analyzing their impact on the robot's performance. In Sec. 3.5, we describe how supervised learning can be used in combination with MCTS to enable real-time execution in the real world. Section 3.6 presents the simulation and experimental results, showcasing the approach on an electric quadruped robot. Finally, Sec. 3.8 concludes the chapter, summarizing our findings and suggesting directions for future research.

3.3 Gait Planning using MCTS

In this section, we present the MCTS gait planning architecture and describe how the MCTS iterative steps are adapted in the context of contact planning for controlling a multi-legged system. For the remainder of this chapter, when referring to the terms *gait*

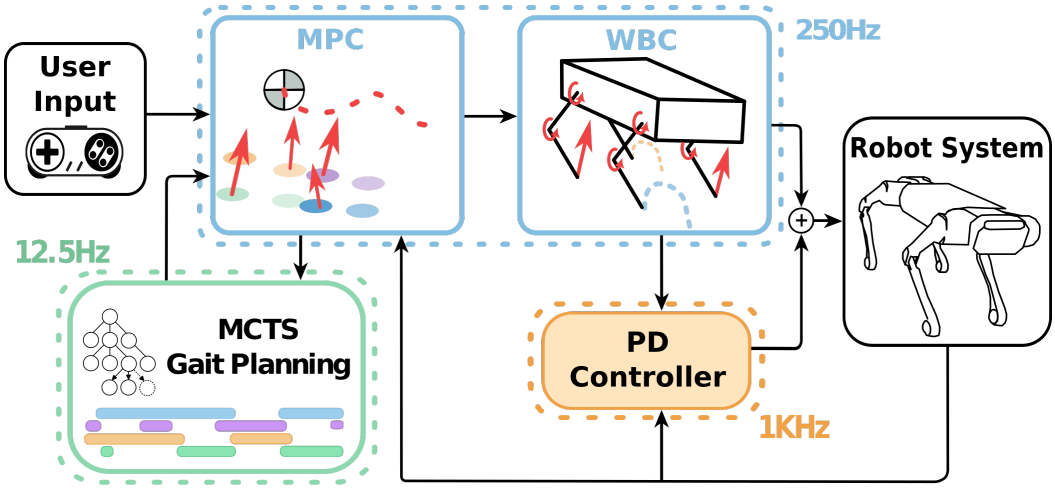

Figure 3.1 Control block diagram of the proposed approach based on the previous work [Amatucci et al., 2022]. The green block is executed at 12.5 Hz, the blue block at 250 Hz, and the orange block at 1 kHz.

planning or *contact planning* we mean in both occasions the optimization of both the contact sequence and respective contact timings of the end-effectors.

The proposed pipeline is shown in Fig. 3.1. A Model Predictive Control (MPC) framework receives, as input, a reference velocity from the user and an optimized gait sequence from the MCTS gait planner. The input velocity is used to generate the reference trajectory for the robot's base, while the gait sequence is used to generate the reference stepping locations and regulate the respective contact timings. The MPC and Whole Body (WB)-Control are then responsible for producing the desired joint torques to track the references.

The gait planning problem is formulated as a Markov Decision Process (MDP). The MDP state **s** describes the contact state of the system and it includes each end-effector contact status, swing time, and stance time. The contact status is defined as a binary value identifying if the end-effector is in contact or not. The swing and stance times are continuous variables that identify for how long the end-effector has been out-of-contact (i.e. in swing) or in contact (i.e. in stance). The action **a**, causing a state transition **s' = f(s, a)**, selects the next contact status among the feasible contacts (Sec. 3.3.2), thereby also modifying the respective swing or stance timings. Each state is assigned a prediction cost $P(s)$ specifying the expected cost to go if such a state is selected.

The MCTS algorithm is used to solve the MDP and to optimize the gait sequence. Starting from a root node, describing the current contact state of the system, a search tree is created where each child node corresponds to a contact choice. At each iteration, the tree grows and deeper nodes in the tree constitute different contact plans up to a specified

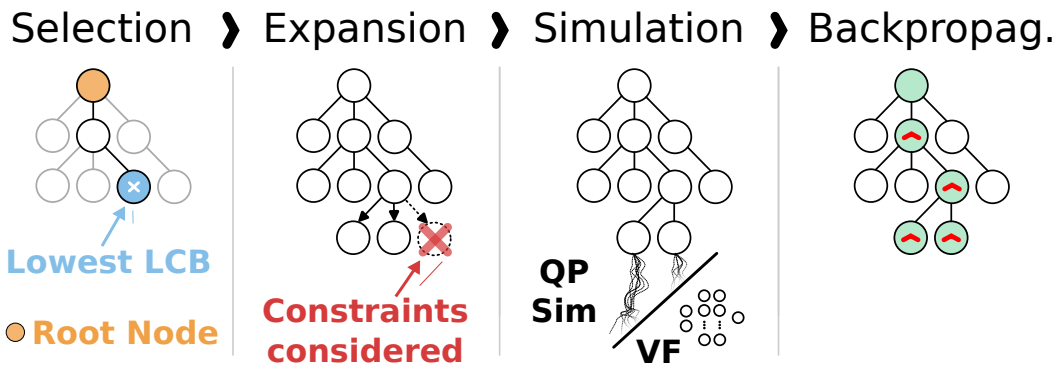

Figure 3.2 MCTS search process [Amatucci et al., 2022] augmented with learned value function evaluation. *Selection*: starting from the root node, a tree traversal is executed to find the node with the lowest cost. *Expansion*: the selected node's children that respect the MCTS constraints are added to the search tree. *Simulation*: the expanded nodes are assigned a prediction cost by solving several optimization problems and/or evaluating a learned value function network. *Backpropagation*: the assigned prediction costs are backpropagated recursively to update the node's ancestors' costs.

time horizon. The MCTS search process is shown in Fig. 3.2 and is briefly described in the remainder of this section. The MCTS algorithm terminates upon convergence, which is reached when a terminal node is selected during the MCTS *selection* process. A node n is terminal if its depth matches the MCTS planning horizon.

3.3.1 Selection

In the selection step, MCTS chooses which node to expand and explore next. This is a crucial part of the iterative process and is subject to the *exploitation-exploration trade-off*. More precisely, when deciding the next expansion direction the process should balance between favoring the most promising node for a faster convergence or exploring other possibilities. In this work, as in [Amatucci et al., 2022], we select the node to expand next based on the lowest Lower Confidence Bound (LCB) which incentivizes the exploration of nodes that are less visited in the tree. More precisely, the more a node is simulated, the smaller the difference between the LCB score and the node's average prediction cost becomes, where the latter is obtained by performing multiple MPC rollouts (Sec. 3.3.3). On the other hand, the fewer times a node is simulated, the higher the discount on the cost, incentivizing its selection for exploration.

3.3.2 Expansion

In the expansion step, the children of the selected node are added to the tree, each representing a potential contact state transition. For a legged system with m legs, the number of children is 2^m since each leg can either be in contact or not. However, because we use a simplified SRB model in the MPC (Sec. 3.3.3), the leg dynamics are ignored, potentially leading to fast swing motions or short stance phases. To prevent this, we impose swing and stance time constraints during expansion. If a leg's swing or stance time has not reached the minimum threshold, only child nodes that maintain the current swing or stance state are added, reducing the number of children. In this work, the minimum swing and stance times are set to 0.24 s and 0.16 s, respectively.

3.3.3 Simulation & Backpropagation

In the simulation step, each expanded node n is assigned a prediction cost $\bar{P}_n$, which guides the MCTS search. To compute this cost, multiple MPC rollouts are solved for different gait sequences. For a given node n_i, we retrieve its contact sequence. If incomplete, we fill the sequence by sampling contacts from a uniform distribution that meets swing and stance constraints. This sampling is repeated multiple times to obtain a more reliable average cost. Each sequence is evaluated, in parallel, by solving an OCP for a simplified SRB model, following the MPC formulation from [Turrisi et al., 2024]. Our state and control vectors are defined as follows:

$$\boldsymbol{x} = [\boldsymbol{p}_c, \dot{\boldsymbol{v}}_c, \boldsymbol{\Phi}, \boldsymbol{\omega}^b] \in R^{12},$$
$$\boldsymbol{u} = [\boldsymbol{\lambda_{grf,1}}, \boldsymbol{\lambda_{grf,2}}, \boldsymbol{\lambda_{grf,3}}, \boldsymbol{\lambda_{grf,4}}] \in R^{12}$$

where $\boldsymbol{p}_c \in R^3$ and $\dot{\boldsymbol{v}}_c \in R^3$ are respectively the position and the linear velocity of the Center of Mass (CoM); $\boldsymbol{\Phi} \in R^3$ is the base angular position (roll, pitch, and yaw); $\boldsymbol{\omega}^b \in R^3$ is the base angular velocity in the base frame, and $\boldsymbol{\lambda_{grf,i}} \in R^3$ the respective Ground Reaction Forces (GRF) for the i^{th} robot's foot. All the quantities, if not specified, are expressed in the world frame.

The cost of the OCP is defined as the combination of quadratic tracking and regularization terms, described as follows:

$$f = \sum_{k=0}^{N} \left(\|\boldsymbol{\epsilon_{x_k}}\|_{\boldsymbol{Q}_x} + \|\boldsymbol{\epsilon_{u_k}}\|_{\boldsymbol{R}_u} \right) \tag{3.1}$$

where $\boldsymbol{\epsilon}_{(.,k)}$ is the error with respect to the reference state and control at the k^{th} prediction step, and $\boldsymbol{Q}_x$ and $\boldsymbol{R}_u$ are diagonal weighting matrices.

Finally, the dynamic model is defined as

$$\begin{bmatrix} \dot{\boldsymbol{p}}_c \\ \dot{\boldsymbol{v}}_{\mathrm{c}} \\ \dot{\boldsymbol{\Phi}} \\ \dot{\boldsymbol{\omega}^b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_{\mathrm{c}} \\ 1/m \sum_{i=1}^{4} \delta_i \boldsymbol{\lambda_{grf,i}} + \boldsymbol{g} \\ \boldsymbol{E}'^{-1}(\boldsymbol{\Phi})\boldsymbol{\omega}^b \\ -\boldsymbol{I}_{\mathrm{c}}^{-1}\left(\boldsymbol{\omega}^b \times \boldsymbol{I}_{\mathrm{c}}\right)\boldsymbol{\omega}^b + \sum_{i=1}^{4} \delta_i \boldsymbol{I}_{\mathrm{c}}^{-1} \boldsymbol{r}_{f,i} \times \boldsymbol{\lambda_{grf,i}} \end{bmatrix} \tag{3.2}$$

with m characterizing the robot mass subjected to gravitational acceleration $\boldsymbol{g}$ and $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ the constant inertia tensor centered at the robot's CoM; $\boldsymbol{E}'^{-1}$ is a mapping from the SRB angular velocity to Euler rates; and the displacement vector between the CoM position $\boldsymbol{p}_c$ and the i^{th} robot's foot $\boldsymbol{p}_{f,i}$ is defined as $\boldsymbol{r}_{f,i} = \boldsymbol{p}_{f,i} - \boldsymbol{p}_c \in \mathbb{R}^3$. Binary variables $\delta_i = \{0, 1\}$ indicate whether an end-effector makes contact with the environment, and can produce interaction forces, or not. These variables are extracted from the simulated gait sequence. Finally, friction cone constraints are added to the optimization problem to limit the maximum and minimum GRF and avoid foot slippage. We define the QP problem to be solved starting from the cost function in (3.1) and imposing as equality constraint the discretized and linearized version of the dynamics in (3.2), while as inequality constraint we impose the outer pyramid approximation of the friction cones. Once f is computed by solving the QP, we update the prediction cost for each node n that is simulated as follows:

$$\bar{f}_n = \frac{\sum_{m=1}^{N_{sim}} \left(f_m + \sum_{l=1}^{L} \beta(T_{sw_r} - T_{sw,l}) \right)}{N_{sim}} \tag{3.3}$$

where N_{sim} is the number of times we solve the QP with different randomly completed contact sequences. We observed that by using only the cost f, the MCTS tends to choose in most cases the fastest allowed swing time. This behavior arises because the SRB model does not consider any component of the leg dynamics. To overcome this limitation, we included an additional discrete cost to drive each leg's swing time $T_{sw,l}$ towards a reference swing time T_{sw_r}. The constant β balances between faster and slower swing timings. Increasing the value of β makes the MCTS solution track the reference frequency making it less prone to show any contact timing adaptation in response to disturbances. As described in Sec. 3.4.1, the value N_{sim} is critical for the success of the algorithm. Due to its sampling-based nature, performing few simulations can lead to inaccurate prediction costs, whereas too many random sequences can lead to the impossibility of solving the MCTS algorithm within the replanning time budget. In the backpropagation step, the prediction cost of each simulated

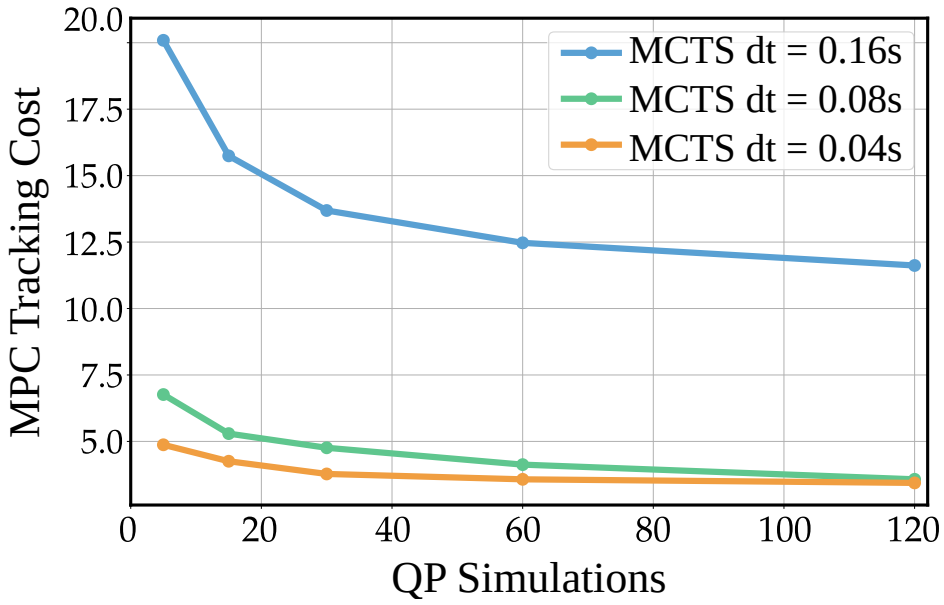

Figure 3.3 Comparison of the mean MPC tracking cost for different tree discretizations and increasing number of simulations while disturbing the system along different swing phases with a force of 150 N for 100 ms.

node is propagated back to the respective parent node in a recursive fashion until the root node is reached. Propagating the cost back is important to improve the accuracy of the estimated cost for nodes closer to the root one.

3.4 MCTS Parametrization

There are four main parameters that can affect the optimality of the planned gait sequence and the respective motion performance of the system. In this section, we present the quantitative evaluation of each parameter's influence on the MCTS solution used to determine our parameter choices. These are:

- *Number of Simulations*: this is the number of times an incomplete gait sequence is randomly filled and the OCP described in Sec. 3.3.3 is solved.
- *Tree Discretization*: the time resolution for each node's contact status, indicating the duration for which a specific contact status is maintained.
- *Tree Horizon*: the length of the time horizon over which the gait sequence is optimized.
- *Replanning Frequency*: the rate at which the gait sequence is updated per second.

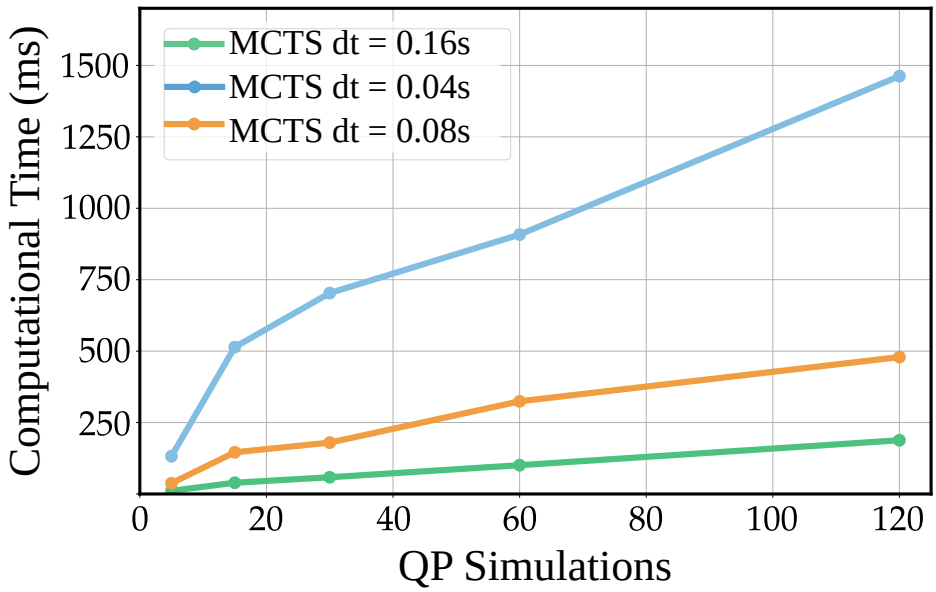

Figure 3.4 Comparison of the mean MCTS computation time for different tree discretizations and increasing number of simulations while disturbing the system along different swing phases with a force of 150 N for 100 ms.

To evaluate the parameters' importance, we compare their influence on the MPC tracking cost in simulation using the RaiSim physics engine [Hwangbo et al., 2018]. The evaluations were conducted using the Aliengo robot model, a quadrupedal robot weighing 22 kg and measuring 65cm in length. In these evaluations, the robot is tasked with tracking a reference forward velocity of 0.3 m/s while being laterally disturbed by a force of 150 N for 100 ms every 3 s. The disturbances are applied 50 times at five different instances of the swing phase (0%, 20%, 40%, 60%, 80%). Unless stated otherwise, we assume a tree horizon value of 0.64 s for the presented evaluations.

3.4.1 Number of Simulations & Tree Discretization

Figure 3.3 shows the MPC tracking cost for different MCTS tree discretization time dt (0.16 s, 0.08 s, 0.04 s) and the number of MPC rollouts (5, 15, 30, 60, 120) performed at each node expansion. The tree discretization values are chosen to be multiples of 0.04 s, which is the MPC discretization timestep, in order to ease the contact sequence conversion from the MCTS gait sequence to the MPC gait sequence.

Figure 3.3 highlights how large discretization time leads to very high MPC tracking costs, mainly due to the large model inaccuracies induced by the larger dt. Using a dt that is twice as large as the MPC's discretization performs worse than using a dt that matches

the one of the MPC when the number of simulations is low. However, as the number of simulations increases, the performance of the two discretizations becomes almost identical. Figure 3.3 also presents evidence that a high number of simulations during the MCTS gait planning leads to better performance. This is due to a better cost estimate for the nodes as they are less sensitive to cost outliers brought by the random nature of the sampling process.

The number of simulations and the tree discretization not only affect the MPC tracking cost but also the computation time required for the algorithm to converge. Figure 3.4 shows how larger computation times are associated with a higher number of simulations and smaller tree discretizations. The higher number of MPC rollouts increases the convergence time due to the increasing number of QP problems that must be solved to evaluate the final cost associated with every node. The tree discretization time directly affects the MCTS depth, since the smaller the discretization time the higher the number of nodes that must be evaluated during the search. Based on the MPC tracking cost shown in Fig. 3.3, we select 0.08 s and 120 as the best tree discretization time and number of MPC rollouts for the MCTS gait planning process, respectively.

3.4.2 Tree Horizon

Figure 3.5 shows the influence of the tree horizon parameter on the MPC tracking cost when using a tree discretization of 0.08 s and 120 MPC rollouts. We can observe that longer horizons marginally change the cost, hinting at the fact that longer horizon plans are not crucial, as shown in [Li and Wensing, 2024], as long as fast replanning is carried out (Sec. 3.4.3). This conclusion may be task-dependent. In fact, while the reduced model's approximation is dominant in the tested scenarios, for cases where the robot must execute long flight phases, such as sparse stepping stones, a longer horizon could become crucial.

3.4.3 Replanning Frequency

As previously shown in Sec. 3.4.1, a high number of simulations is important to obtain a good cost estimate for the nodes. However, the higher the number of simulations the higher the time to complete the MCTS planning process. Therefore, it is important to establish the minimum replanning frequency that should be respected to maintain the best MPC tracking cost performance established in Sec. 3.4.1. To do so, we evaluate the influence that different replanning frequencies have on the MPC tracking cost. Since we evaluated the MCTS gait planner with frequencies as low as 2 Hz, we need to make sure to have a

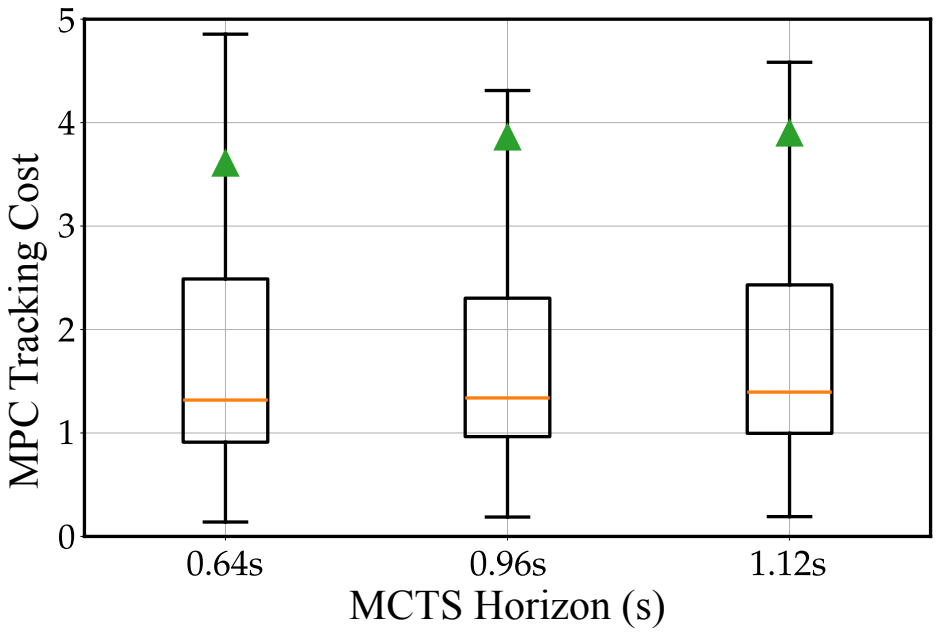

Figure 3.5 Comparison of the mean MPC tracking cost for a tree discretization dt of 0.08 s, 120 MPC rollouts, and different tree horizons while disturbing the system along different swing phases with a force of 150 N for 100 ms.

long enough contact sequence to feed into the MPC. For this reason, we use a tree horizon of 1 s.

Figure 3.6 shows the influence of the replanning frequencies on the MPC tracking cost and the MPC prediction cost. The figure shows that faster replanning improves the performance until 12.5 Hz where it hits a plateau. Hence, in our setting, 12.5 Hz is selected as the MCTS replanning frequency to be respected, which is also in line with a similar evaluation presented in [Meduri et al., 2023]. It should be noted that this update rate cannot be reached, with the currently available computational power, by the vanilla MCTS implementation described in Sec. 3.3, whose real performance is indicated with the stars in Figure 3.6. Therefore, a significant speed-up is required to reach the established replanning frequency.

3.5 MCTS Speed Up

In Sec. 3.4, we presented the results of our ablation study on the MCTS parameters. From this study, as depicted in Fig. 3.6, we identified a minimum replanning frequency of 12.5 Hz to achieve good system performance. Additionally, we concluded that a greater number of simulations leads to a better MCTS solution and robot's performance. However, as

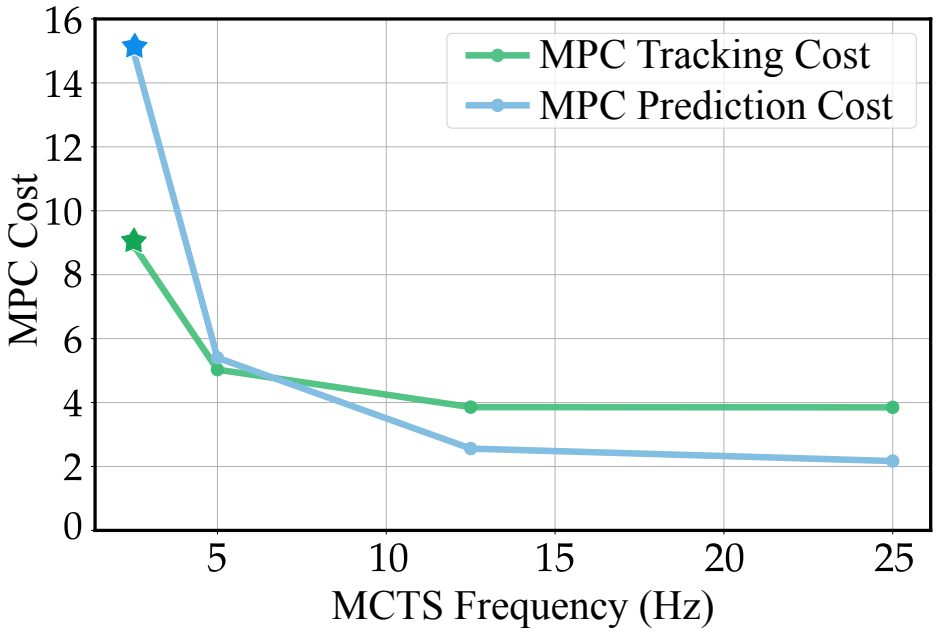

Figure 3.6 Comparison of the mean MPC tracking and prediction cost for a tree discretization dt of 0.08 s, 120 MPC rollouts, and increasing MCTS planning frequencies while disturbing the system along different swing phases with a force of 150 N for 100 ms.

shown in Fig. 3.4, increasing the number of simulations to better evaluate each expanded node also increases the computation time. This means that we can either allow the planner to perform more simulations by lowering the replanning rate, or maintain the minimum replanning frequency by limiting the MCTS to just 5 rollouts per expansion. Both of these solutions, however, limit the performance of the vanilla MCTS, making its deployment on real hardware extremely challenging.

In this section, we present a simple yet effective way to overcome this issue. We propose a learning-based method to reduce the number of MPC rollouts to be solved, with the goal of making MCTS real-time on commonly available hardware without a substantial decrease in performance. We also describe our dataset generation process and the adopted architecture for the learning-based method.

3.5.1 Dataset & Architecture

The dataset for the training process is generated in simulation using the Raisim physics engine. We run the vanilla MCTS method with the best-identified parameters in Sec. 3.4, which correspond to a dt of 0.08 s and 120 simulations. We trained on flat and rough terrains while randomizing the velocity commands, the step height, the robot's mass, the

swing trajectory controller gains, and the robot's height. Additionally, we also perturb the system with randomly generated forces in terms of magnitude, duration, and frequency. We log each MCTS iteration input and output and use the nodes' cost estimates for training. This way, we collect a total of 26563 MCTS trees from the simulations.

A Multi-Layer Perceptron (MLP) with three hidden layers of 512 neurons each, batch normalization layers, and dropout regularization are used as the underlying architecture. We use ReLU activation functions for the intermediate layers and a linear activation function for the output layer. The Adam optimizer [Diederik, 2015] with a learning rate scheduler is used to optimize the network's weights.

3.5.2 Value Function Network

Given the computationally demanding cost of estimating expanded nodes by solving several MPC rollouts, we propose to learn a VF that approximates the cost-to-go $\bar{P}_n$ (Sec. 3.3.3) similarly to [Wang et al., 2024]. The network's input is defined as follows:

$$\boldsymbol{x}_{vf} = [\boldsymbol{z}_{c,e}, \boldsymbol{v}_{c,e}, \boldsymbol{\Phi}_e, \boldsymbol{\omega}_e^b, \boldsymbol{r}_f, t_{swing}, t_{stance}, \boldsymbol{c}_n] \in R^{78}$$

where $\boldsymbol{z}_{c,e}$ is the error between the reference and actual CoM height, $\boldsymbol{v}_{c,e}$ is the error between the reference and actual CoM linear velocity, $\boldsymbol{\Phi}_e$ is the error between the reference and actual base orientation, and $\boldsymbol{\omega}_e^b$ is the error between the reference and actual base angular velocity. $\boldsymbol{r}_f$ comprises the actual foot positions with respect to the CoM, t_{swing} and t_{stance} are the actual swing and stance time of each leg in seconds, and $\boldsymbol{c}_n$ is the contact sequence that leads to the node n. If the node is not terminal, we fill the missing part of the contact sequence with -1 values, in order to make sure the input size to the network remains the same.

Estimating a node's cost with the proposed VF takes, on average, less than 1 ms. This results in a substantial speed-up over the QP-based evaluation, since a comparable performance is only achieved with 120 simulations, which would take approximately 20 ms if running 10 processes in parallel.

3.5.3 Combined Approach

Only relying on the learned value function to estimate $\bar{P}_n$ (3.3), can be detrimental when the states are out of the distribution of the training data. This is a well-known problem of imitation learning [Ross et al., 2011] from offline data, and it is likely to happen during the

deployment of such networks in the real world. In this work, we seek to obtain generalization through the combination of model-based MPC rollouts solutions with the bootstrapping obtained by employing the VF network.

Inspired by [Ross et al., 2011], we perform a simple update rule for the node cost, such as

$$P_n = \alpha \bar{P}_n + (1 - \alpha)\bar{P}_{vf}$$

with α being a heuristically chosen parameter, 0.75 in our case. Note that, contrary to [Ross et al., 2011], we keep α fixed to allow the robot to react in new situations. As we will see in the result presented in Sec. 3.6, α makes a trade-off between trusting the learned VF and online MPC rollouts.

3.6 Results

In this section, we present the evaluation of our proposed method in simulation using the RaiSim physics engine and on a real electric quadruped.

We perform several evaluations of our proposed method in simulation. We first compare a vanilla MCTS gait planner against two purely learning-based methods and our proposed hybrid method, using the same evaluation method described in Sec. 3.4. Then, we present a quantitative analysis on the impact of MPC rollouts on the proposed hybrid approach in an Extreme Out-of-Distribution (EOD) situation. Finally, we conduct a comparison between our proposed hybrid method against trotting gait sequences that assume periodic contact timings with different step frequencies.

On hardware, using a real electric quadruped, we demonstrate the pipeline's disturbance rejection performance in comparison to a fixed periodic gait. As highlighted in the accompanying video of [Taouil et al., 2025], this is the first successful real-time implementation of a sampling-based method for non-gaited locomotion. Moreover, we perform a qualitative comparison between simulation and hardware results.

For both simulation and hardware evaluations, we set a maximum allowed time budget of 80 ms for the MCTS gait planner in order to meet the 12.5 Hz replanning frequency identified in Sec. 3.4.

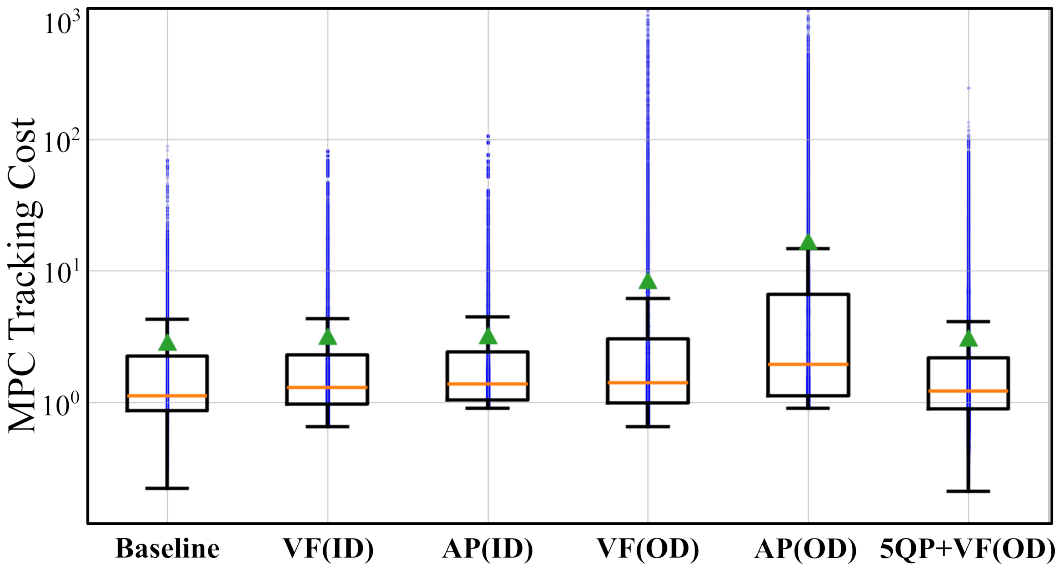

Figure 3.7 Comparison of how the MPC tracking cost distribution varies, Inside Distribution (ID) and Outside Distribution (OD) the learning distribution for different approaches. With a green triangle, we showcase the mean MPC tracking cost. From left to right we show the comparison between the MCTS Baseline against four purely learning-based approaches based on VF and Action Policy (AP) approaches for both ID and OD scenarios, and a hybrid approach performing additional model-based MPC rollouts with VF bootstrapping in an OD scenario. The blue points represent the MPC tracking error samples for each approach. All proposed approaches see an increase in the mean MPC tracking cost compared to the baseline of 11%, 12%, 195%, 484%, and 7% respectively.

3.6.1 Simulation results

Baseline Comparison

we evaluate the following three approaches against a vanilla MCTS gait planner baseline:

- VF
- AP
- Combined Approach (5 QP + VF)

The first two methods are purely learning-based while the third one is a hybrid approach that combines the learned VF network with 5 model-based MPC rollouts. The AP outputs the next optimal contact directly, thereby obtaining the full sequence by successively querying the network. We add such a method in the analysis to provide a complete insight into the benefit of our proposed method. Both VF and AP share the same network architecture, a MLP with three hidden layers of 512 neurons each.

We carry out a comparison, always in terms of MPC tracking cost, on two different scenarios: ID and OD. In the ID case, we train two different models for various target speeds while including, in the training data, external disturbances to the robot base in the form of pushes. On the other hand, in the OD case, we only train the models with different target speeds without taking into consideration any disturbance force. Figure 3.7 shows the results for both scenarios.

In the ID scenario, both the VF and AP learning-based models perform on par with the baseline in terms of mean and variance of the MPC tracking cost. The mean MPC tracking cost is almost identical between the three approaches, showing a good overall approximation by the learning-based methods. The MPC tracking cost distribution for both VF and AP tends to reach higher peaks compared to the baseline, although being within close range. Overall, for both purely learning-based approaches, if they are trained on a diverse enough dataset that covers the state space of the system, they show similar performance as the baseline. In the OD scenario, the learning-based models undergo a substantial increase in the mean MPC tracking cost compared to the baseline, with the VF showing a better performance compared to AP. This is primarily because, in the case of the VF, constraints are imposed during the expansion process, whereas for the AP, they are integrated as part of the learned behavior.

Observing the results for the third comparison, we see that the combination of the VF with only 5 MPC rollouts significantly lowers the mean MPC tracking cost and brings back its distribution to a range similar to the baseline. This demonstrates the benefit of our proposed hybrid approach in OD cases, where a few QP model-based simulations can help maintain a certain level of robustness and performance while being capable of running at the desired replanning frequency.

MPC Rollouts Impact on the Hybrid Approach

Figure 3.8 shows the effect of the numbers of MPC rollouts on our proposed hybrid approach in an EOD scenario. We compare the hybrid approaches with an MCTS vanilla baseline and a pure VF learning-based approach. The EOD scenario is set such that the VF is trained only on data containing forward velocities up to 0.5 m/s on a flat terrain, but we test on a sloped terrain with a commanded forward velocity of 0.6 m/s without exerting external disturbance forces.

The results, similar to Sec. 3.6.1, show how a pure VF approach performs worse in OD scenarios compared to a hybrid approach, where 5 MPC rollouts are performed. However, we also note that increasing the number of MPC rollouts performed in the hybrid approach

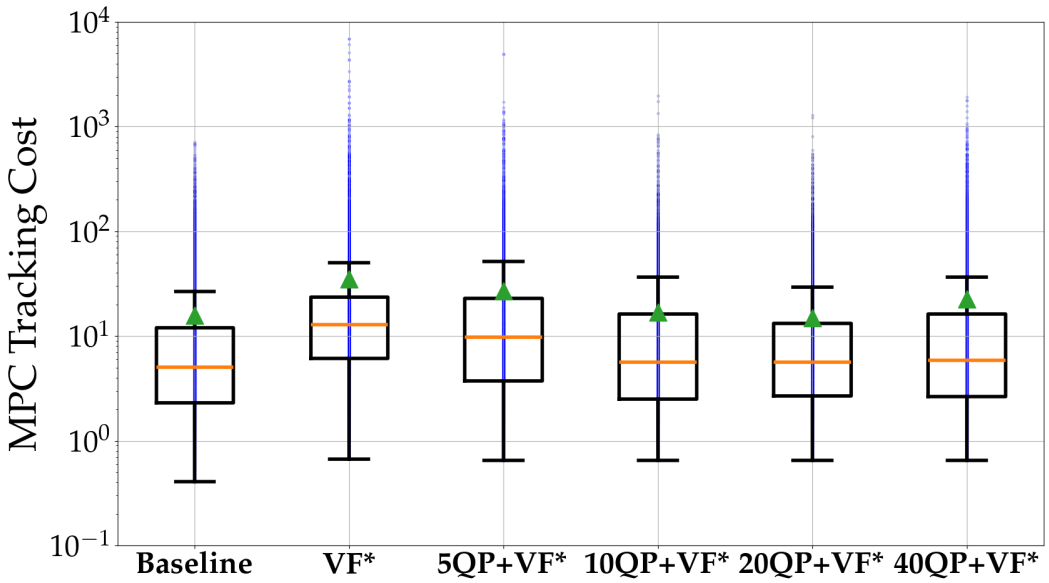

Figure 3.8 Comparison of how the MPC tracking cost distribution varies for an EOD scenario. From left to right we show the comparison between the MCTS Baseline, using a tree discretization of 0.08 s and 120 simulations, against a pure VF learning-based approach, and four hybrid approaches performing respectively 5, 10, 20, and 40 additional model-based MPC rollouts with VF bootstrapping. The blue points represent the MPC tracking error samples for each approach. With the * we mean a VF trained according to the description in Sec. 3.6.1. All proposed approaches see an increase in the mean MPC tracking cost compared to the baseline of 123%, 73%, 7%, 5%, and 44% respectively.

does not necessarily lead to better system performance. Since we enforce an optimal replanning frequency of 12.5 Hz for all configurations, as found in Sec.3.4.3, the hybrid MCTS has limited time available to return a solution. This means that by increasing the number of simulations per node expansion, the MCTS spends the majority of its time estimating fewer node costs with MPC rollouts, leaving the rest to be estimated using VF alone. Given our computational resources, the average percentage of nodes evaluated with MPC rollouts decreases from 39% with 5 rollouts per node to 27% for 40 rollouts. Consequently, the mean MPC tracking cost improvement stagnates, showing only minor gains when increasing the number of rollouts up to 20, while system performance actually degrades with 40 rollouts.

In conclusion, increasing the number of MPC rollouts in the hybrid MCTS is not always advantageous. A balance must be found between using more rollouts per node to obtain fewer but more accurate QP-based cost estimates or reducing the number of rollouts to estimate less accurate costs across a larger number of nodes.

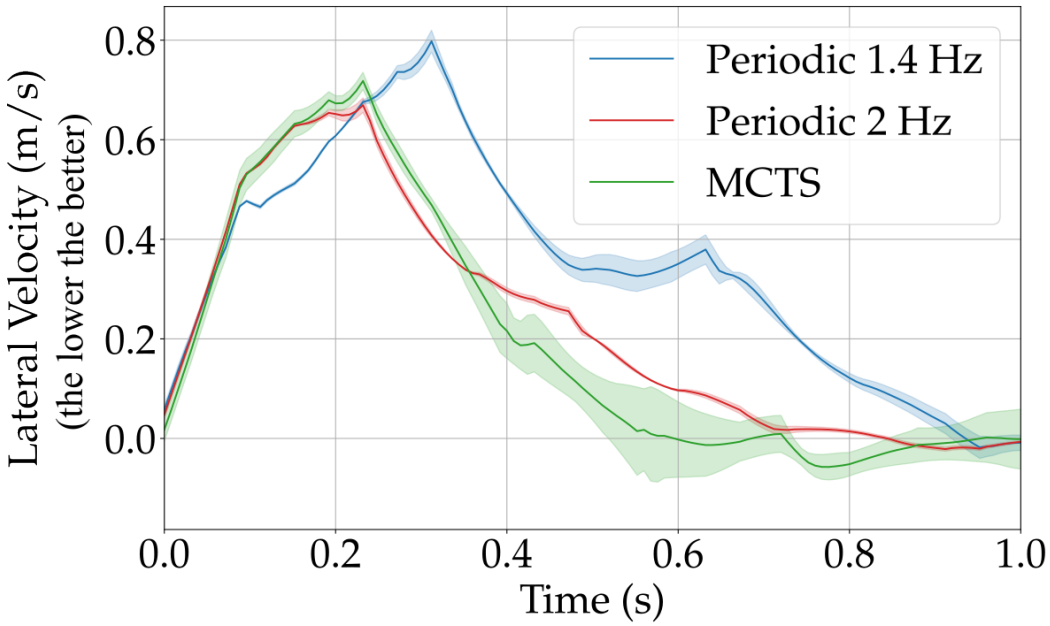

Figure 3.9 Comparison of the lateral velocity (the solid colors represent the mean value and the shaded areas the confidence band) between two periodic trot gait sequences with a respective step frequency of 1.4 Hz and 2 Hz against our proposed MCTS gait planner. During walking, we apply a disturbance at $t = 0$ s with a positive force of 150 N for 100 ms in the robot's lateral direction on a sloped terrain. We repeat the disturbance process 50 times for each method. As shown in the plot, the MCTS results in an overall lower lateral velocity after disturbance.

Comparison with Periodic Contact Scheduling

Figure 3.9 presents a comparison of the mean lateral velocity and respective confidence band between a non-gaited sequence, obtained using MCTS, and two periodic trotting gaits: one with a relatively low stepping frequency of 1.4 Hz and the other with a higher stepping frequency of 2 Hz. For both frequencies, we consider a step duty factor of 0.6 (ratio between the stance period and the whole step cycle period). The values shown in Fig. 3.9 are obtained by pushing the system 50 times at time 0 s with a force of 150 N in the robot's lateral direction for 100 ms while tracking a forward velocity of 0.3 m/s.

Our method and the 2 Hz trotting gait show a similar overall performance. They both peak in terms of recorded lateral velocity after 0.2 s from the moment of the push, with our method having a slightly higher peak lateral velocity of 0.72 m/s compared to the 0.63 m/s for the 2 Hz trotting gait. On the other hand, the 1.4 Hz trotting gait peaks only at 0.3 s with a peak lateral velocity of 0.8 m/s.

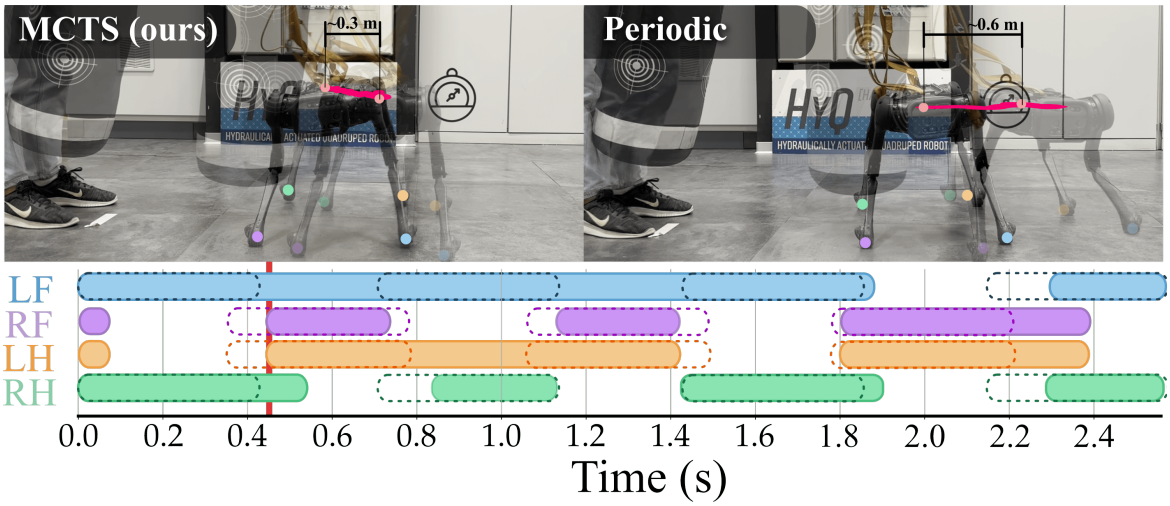

Figure 3.10 On top, snapshots of the experiments on the Aliengo platform. On the left, the robot used the presented MCTS gait planner. On the right, a periodic trot with the same reference step frequency used by MCTS was used. A punching bag let go from the same height pushes the robots in a repeatable manner in both cases. In pink is the trace of the robot trunk, the smaller the traveled distance, the better the disturbance rejection. On the bottom, are the plots of the contact sequence chosen by the MCTS (solid color) and the periodic gait (dashed line) for each leg, where we highlight in red the moment of contact between the robot and the punching bag.

In general, we also note how our proposed method is able to bring the system back to zero lateral velocity at around 0.6 s which is significantly faster than the 0.72 s for the 2 Hz trotting gait and the 0.96 s for the slower 1.4 Hz trotting gait. The superior performance of our method comes from its ability to optimize both timing and gait sequence, enabling it to outperform conventional periodic gaits.

3.6.2 Hardware Experiments

To validate the proposed method, we also performed tests on real hardware. Thanks to the speed-up achieved by our hybrid approach, we are able to reach the necessary replanning rates for deployment on a real quadruped. We test the pipeline on Aliengo, an electric quadruped robot made by Unitree Robotics [unitree, 2025] that weighs around 22 kg. The entire pipeline runs externally on a 12th-generation Intel i7 processor. In Fig. 3.10, we present some snapshots of the experiments while also highlighting the contact sequence generated by the proposed MCTS gait planner. To ensure the repeatability of the scenario, the robot is pushed by a 27 kg punching bag hanging from a crane. Our method, exploiting

the MCTS gait adaptation, is compared with a periodic approach where the robot trots at a fixed frequency of 1.4 Hz and duty factor 0.6.

As shown in the diagram, the MCTS gait planner adapts the contact sequence by keeping both left feet on the ground after the impact to better counteract the push before returning to the more efficient trotting frequency incentivized by the cost in (3.3). The improved disturbance rejection can be observed by the trajectory of the trunk's position (illustrated by a pink line). These results and further tests on the robot can be seen in the accompanying video of [Taouil et al., 2025].

3.7 Discussion & Limitation

This work tries to overcome most of the technical limitations of our previous approach [Amatucci et al., 2022]. Thanks to a full C++ implementation and a thread-safe QP-based simulation step, we were able to increase the algorithm's speed compared to the earlier MATLAB version in [Amatucci et al., 2022] by a factor of 4. The in-depth analysis we conducted on the parameter set enabled an optimized deployment of the MCTS planner, while the incorporation of the learned VF provided the remaining performance boost needed to achieve real-time execution. Our work stresses the improved locomotion robustness given by our formulation, but a critical limitation, crucial for deploying legged robots in real scenarios, remains. Our current implementation lacks a proper way to handle unstructured, non-flat terrains. In fact, surface selection further expands the already huge search space by increasing the number of possible contact combinations. Moreover, on more extreme terrains, where hardware limits play a key role, the simplified model used in the simulation may no longer be sufficiently accurate to find feasible solutions. However, with the current QP solver, increasing the model accuracy would drastically slow down the simulation. Although the distributed approach presented in Chapter 2 could mitigate this issue, the number of simulations required at each MCTS iteration will still be prohibitively high, making not only the real-time deployment a distant dream but also the collection of data for the VF training prohibitively long. These considerations motivated the need for a solver capable of massively parallelizing OCP solutions in large batches, a direction that constitutes one of the core motivations in Chapter 4.

3.8 Conclusions

In this chapter, we presented a novel approach for non-gaited legged locomotion that extended the work in [Amatucci et al., 2022] by bringing to the framework real-time capability and the first-ever successful implementation on hardware of such a sampling approach. We offered an extensive analysis of the parametrization of the MCTS formulation for non-gaited locomotion and compared it to standard control approaches that assume periodicity in the gait sequence, ultimately showcasing the benefit of our approach over such methods.

Chapter 4

Primal-Dual iLQR for GPU-Accelerated Learning and Control in Legged Robots

L. Amatucci*, J. Sousa-Pinto*, G. Turrisi, D. Orban, V. Barasuol and C. Semini, "Primal-Dual iLQR for GPU-Accelerated Learning and Control in Legged Robots" in IEEE Robotics and Automation Letters, 2025, doi: 10.1109/LRA.2025.3632610.
*Equal contribution

4.1 Motivation

In this final work, we present a novel Graphics Processing Unit (GPU)-accelerated solver that exploits hardware acceleration to enhance the scalability of the Optimal Control Problem (OCP) with respect to both the state dimension and the prediction horizon, enabling Model Predictive Control (MPC) formulation of long horizons and systems with a high number of Degrees of Freedom (DoF). The idea for this work originated from the desire to develop a differentiable and parallelizable solver suitable for massively parallel training. These reasons guide our decision to focus on a fully GPU-friendly implementation. GPUs are by design optimized for high-throughput linear algebra, and while they typically have slower single-core performance compared to Central Processing Unit (CPU)s, their large number of specialized cores allows them to outperform traditional processors as the dimensions of the involved matrices grow. This makes hardware acceleration particularly advantageous for large-scale problems. During development, several interesting additional benefits emerged placing our

implementation on par with, and in some scenarios surpassing, state-of-the-art CPU-based solvers. The result, enabled by GPU optimized matrix operations, is a solver capable of handling tens of robots coordinated by the same centralized MPC to perform collaborative tasks. In addition, the solver supports horizons on the order of hundreds of nodes, which is essential in scenarios where long-term prediction is crucial. Moreover, it can be used in a massively parallel simulation pipeline to accelerate the training of Reinforcement Learning (RL) policies.

4.2 Introduction

Among the many well-known control approaches available, MPC has proven to be highly effective in generating and controlling complex dynamic behaviors in robotic systems, especially legged robots, as shown by [Farshidian et al., 2017] and [Grandia et al., 2023]. At the core of an MPC is the transcription of a task we want the robot to perform into an OCP and the ability to solve it fast enough to be used in a closed-loop controller.

One of the most widely used methods in robotics for solving such OCP is Differential Dynamic Programming (DDP), which gained renewed attention through the work of [Todorov and Li, 2005], who introduced a variant known as Iterative Linear Quadratic Regulator (iLQR). iLQR discards the second-order terms of the dynamics in the Hessian, sacrificing the local quadratic convergence of DDP in favor of faster update rates. However, a key limitation of this method is its single-shooting nature, which requires a feasible initial guess as a starting point and often exhibits critical numerical issues. [Giftthaler et al., 2018] overcame such limitation presenting a multiple shooting variant of the iLQR algorithm. More recently, [Mastalli et al., 2020] proposed a feasibility-driven multiple shooting approach to DDP with notable results also on real hardware [Dantec et al., 2022]. Another common approach to the solution of the OCP is Sequential quadratic Programming (SQP). Compared to DDP-like algorithms, SQP has been more extensively developed for general-purpose solvers due to its flexibility in handling a broader range of constraints and objectives, as well as its robustness in dealing with infeasible iterations. Solvers like OCS2 [Farshidian et al., 2023] and acados [Verschueren et al., 2021] also showed relevant results on real hardware as demonstrated in [Jenelten et al., 2022] and [Rathod et al., 2021], thanks to the tailoring of the solver to the structure of OCP. Recent work like [Jordana et al., 2023] highlights the connection between SQP and DDP, describing how the sparsity structure of the OCP can be exploited not only in DDP-like solvers but also in other solver types, such as SQP or interior point-based solvers.

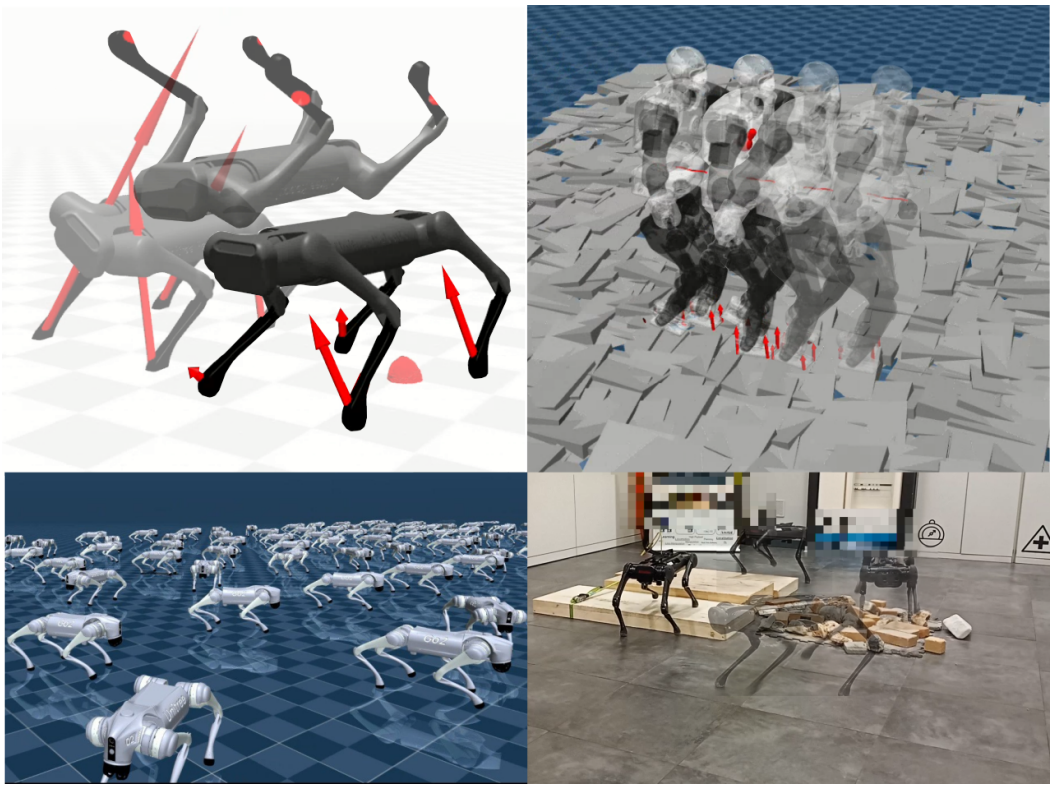

Figure 4.1 Top left: the quadruped robot Aliengo performing a barrel roll. Top right: the humanoid robot Talos walking blindly on uneven terrain. Bottom left: an example of the massive parallelization achievable with our JAX formulation. Bottom right: Aliengo robot walking blindly on uneven terrain in a lab experiment.

While the above methods have been shown to exploit the structure of the OCP, they have struggled to exploit new hardware accelerators like GPUs. Compared to CPUs, graphics cards can massively parallelize computation and are specifically designed for high-throughput linear algebra operations. [Jeon et al., 2025] tried to bridge the gap by developing Cusadi, a tool to convert expressions written in the well-known framework CasADI [Andersson et al., 2019] to GPU. Cusadi only translates closed-form expressions into GPU-compatible code, lacking the branching and looping capabilities necessary for most solvers. [Bishop et al., 2024] introduced a GPU accelerated Quadratic Programming (QP) solver, Relu-QP, leveraging the Alternating Direction Method of Multipliers (ADMM) algorithm, achieving performance superior to state-of-the-art CPU-based solutions. However, its applicability to Nonlinear Programming (NLP) is constrained by the need to pre-compute offline part of the algorithm. [Lee et al., 2022] implements an iLQR controller on GPU but only partially leverages its capabilities, limiting parallelization to the line search and gradient computation. In contrast, [Lee et al., 2024] focuses only on batching, solving

multiple OCPs in parallel without exploiting state or temporal parallelism. [Plancher and Kuindersma, 2020] extends the approach of [Lee et al., 2022] by incorporating the temporal parallelization strategy introduced in [Farshidian et al., 2017]. However, [Farshidian et al., 2017] requires introducing approximations in the backward pass and performing a consensus sweep to maintain consistency. [Frasch et al., 2015], proposed a method to iteratively solve linear quadratic control exploiting parallel computation, while [Wright, 1991] showed how to partition dynamic programming for parallel computation. However, none of the mentioned methods achieved at the same time logarithmic time complexity and an exact formulation. On the other hand, [Särkkä and García-Fernández, 2023] showed how parallel associative scan operations can be utilized to improve the computational complexity of DDP. When parallelized on GPU, associative scans can compute the optimal control policy in $\mathcal{O}(\log^2(n)\log N + \log^2(m))$ instead of the $\mathcal{O}(N(n+m)^3)$ of the classical Riccati Recursion found in DDP, where n and m are the dimensions of the system state and control input, and N is the prediction horizon length.

In this work, we designed an SQP method that uses an associative scan-based LQR solver to solve the primal-dual Karush-Kuhn-Tucker (KKT) system efficiently. The solver avoids the offline precalculation necessary for Relu-QP [Bishop et al., 2024]. Our method benefits from a multiple shooting implementation, instead of the single shooting approach presented in [Särkkä and García-Fernández, 2023]. Our algorithm fully exploits the parallelization capabilities of GPUs both in the temporal and the state space dimensions, while maintaining, in contrast to [Plancher and Kuindersma, 2020], an exact backward pass. We tailored the formulation for the deployment as a receding horizon controller for legged robot locomotion and analyzed its impact on learning and control.

Control: At first, the reduction in the computational complexity of the solver may appear to have limited significance in a practical application. Some of the already described methods, like OCS2 [Farshidian et al., 2023] and crocoddyl [Mastalli et al., 2020], have been successfully used for whole-body MPC even for experiments, as shown in [Mastalli et al., 2023] and [Grandia et al., 2023]. To ensure that such an MPC formulation can run online on real hardware, the prediction horizon length and the number of robots that can be controlled simultaneously are constrained by the time complexity of the underlying algorithm. In particular, for solvers like OCS2, acados, and crocoddyl, the complexity scales linearly with the horizon length and cubically with the state dimension. In contrast, our approach scales with the square-log of the horizon length and linearly with the state dimension. This improved scaling enables the use of more nodes at the same update rate, allowing for finer integration steps for a better approximation of the robot dynamic or, more critically, the

inclusion of a Moving Horizon Estimator (MHE), such as those presented by [Yoon et al., 2024] or [Martinez et al., 2024], within the same optimization loop, opening the pace for "end-to-end" MPCs capable of adapting and reacting to system changes directly using the sensor data. The reduction in complexity with respect to the state dimension enables better scalability across different robot morphologies. While other methods, such as the one presented by [Amatucci et al., 2024a], use an approximate approach in the form of consensus ADMM to parallelize over the system state, our implementation leverages the inherent parallelism of matrix operations provided by the GPU. State space scaling becomes crucial when a centralized controller for collaborative tasks is considered. As demonstrated by [Vincenti and Coros, 2023] and [Kim et al., 2023], centralized controllers can handle complex tasks involving multiple collaborating agents. However, while these approaches achieve notable performances, the complexity of the task and the number of agents involved are constrained by their capability to solve the increasingly large OCPs at a reasonable control frequency. Our framework overcomes such barriers, as shown in Sec. 4.5.3.

Learning: In recent years, RL has demonstrated remarkable performance, enhancing the robustness and capabilities of legged robots. RL controllers have surpassed model-based controllers in robustness against model mismatches and sensor noise. [Cheng et al., 2023] also demonstrated agile movements involving jumps on unstructured terrains with a real robot. However, RL policies have shown limitations in scenarios that involve crossing gaps and stepping stones, since the learning process efficiency is significantly affected due to the sparse reward signals typical of such tasks. [Jenelten et al., 2024] proposed a solution to bridge the gap between learning-based and model-based controllers by incorporating MPC as a bias for the RL-policy. In contrast, works such as [Omar et al., 2023] and [Pandala et al., 2022] integrated learned behaviors directly into the optimization process. Despite the noticeable results, all of the aforementioned methods that mix model-based controllers and RL suffered from a slower training process due to the dependency on CPU-based solvers. In contrast, our implementation is entirely developed in JAX [Bradbury et al., 2018], and it can be coupled with GPU-based simulators such as IsaacLab [Mittal et al., 2023] or Mujoco XLA [Todorov et al., 2012] to achieve a significant speed-up in simulation throughput.

4.2.1 Contribution

To summarize, the major contributions of this work are:

- a novel GPU-accelerated MPC for legged robots that achieves logarithmic scaling in computation complexity on the horizon length, square-log scaling in the state space

and control vector dimensions, and can easily be parallelized for use with data-driven approaches.

- a detailed analysis of the performance benefits and limitations of the proposed algorithm with respect to state-of-the-art solvers.
- an open-source code repository for the rapid prototyping of MPC for legged robots and loco-manipulation. Written in JAX, the code provides tools for controlling legged robots and also supports large-scale parallelization.

4.2.2 Outline

This chapter is organized as follows. Sec. 4.3 describes the details of the proposed solver. Sec. 4.4 presents the dynamical models used in our formulation. Sec. 4.5 shows the benefits of our approach on comparative simulations against state-of-the-art solutions. Finally, Sec. 4.7 draws the final conclusions.

4.3 Primal-Dual LQR

4.3.1 Optimal Control Problem

We start describing the OCP at the base of the MPC formulation we are presenting. In a OCP, we transcribe not only the task we want to be performed but also the physical limits the system needs to respect. In this work, we consider the OCP in the following form:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \quad \boldsymbol{l}_{N+1}(\boldsymbol{x}_{N+1}) + \sum_{i=0}^{N} \boldsymbol{l}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{4.1a}$$

$$s.t. \quad \boldsymbol{x}_{i+1} = \boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \tag{4.1b}$$

$$\boldsymbol{u}_i \in \mathsf{U}_i \tag{4.1c}$$

$$\boldsymbol{x}_i \in \mathsf{X}_i \tag{4.1d}$$

$$\boldsymbol{x}_0 = \hat{\boldsymbol{x}}_0 \tag{4.1e}$$

where $\boldsymbol{x}_i$ is the system state and $\boldsymbol{u}_i$ is the control input. $\boldsymbol{l}(\boldsymbol{x_i}, \boldsymbol{u_i})$ is the stage cost made of a quadratic tracking and regularization terms, while $\boldsymbol{l}(\boldsymbol{x}_N)$ is the terminal cost, and $h(\boldsymbol{x}_i, \boldsymbol{u}_i)$ is the system dynamics. Finally, X_i and U_i are, respectively, the set of feasible system states and control inputs. The remainder of this section will describe the methods

we use to solve the optimization problem posed by the OCP formulation presented in (4.1). By leveraging both the knowledge of the problem structure and exploiting modern hardware accelerators, we are able to obtain logarithmic time complexity in horizon length N as well as square-log for the state x and input u dimensions. For simplicity, in this section, we avoid treating the inequality constraint in (4.1d) and (4.1c). These details will be treated separately in Sec. 4.3.6. We now focus on the solution of the equality-constrained optimization problem, which only presents the initial condition (4.1e) and dynamics (4.1b) as constraints. In particular, we are going to derive a multiple-shooting approach, which, in contrast to the single-shooting one, keeps both the state and control as optimization variables. We first derive an efficient SQP algorithm that exploits the Riccati recursion to solve the equality constrained QP Sec. 4.3.3, and then we focus on the temporal parallelization of the solver through the use of parallel associative scan, Sec. 4.3.4.

4.3.2 Sequential Quadratic Programming

We start defining the Lagrangian relative to the problem in (4.1) as:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) =& \boldsymbol{l}_{N+1}(\boldsymbol{x}_{N+1}) + \boldsymbol{\lambda}_0^T(\hat{\boldsymbol{x}}_0 - \boldsymbol{x}_0) + \\ & \sum_{i=0}^{N} \boldsymbol{l}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + \boldsymbol{\lambda}_{i+1}^T(\boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) - \boldsymbol{x}_{i+1}),\end{aligned} \tag{4.2}$$

where $\boldsymbol{\lambda}_i$ for $i = 1 \dots N+1$ are the Lagrangian multipliers related to the dynamics. In an SQP method, we iteratively solve the QP derived from the quadratic approximation of the Lagrangian and the linearization of the constraints at the current guess, $\boldsymbol{x}^k$, $\boldsymbol{u}^k$, where k represents the last iterate. The QP is then written as:

$$\begin{aligned}\min_{\boldsymbol{\delta x}, \boldsymbol{\delta u}} & \sum_{i=1}^{N} \begin{bmatrix} \boldsymbol{q}_i \\ \boldsymbol{r}_i \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{\delta x_i} \\ \boldsymbol{\delta u_i} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \boldsymbol{\delta x_i} \\ \boldsymbol{\delta u_i} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{Q_i} & \boldsymbol{S_i}^\top \\ \boldsymbol{S_i} & \boldsymbol{R_i} \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta x_i} \\ \boldsymbol{\delta u_i} \end{bmatrix} \\ & + \boldsymbol{p}_{N+1}^\top \boldsymbol{\delta x}_{N+1} + \frac{1}{2} \boldsymbol{\delta x}_{N+1}^\top \boldsymbol{P}_{N+1} \boldsymbol{\delta x}_{N+1} \\ & \boldsymbol{\delta x}_0 = 0 \\ & \boldsymbol{\delta x}_{i+1} = \boldsymbol{A}_i \boldsymbol{\delta x}_i + \boldsymbol{B}_i \boldsymbol{\delta u}_i + \boldsymbol{b}_i \quad \text{for} \quad i = 0, \dots, N\end{aligned}$$

where the search directions $\boldsymbol{\delta x}$ and $\boldsymbol{\delta u}$ are defined as:

$$\begin{aligned}\delta \boldsymbol{x}_i &= \boldsymbol{x}_i^{k+1} - \boldsymbol{x}_i^k \qquad \text{for} \quad i = 0, \dots, N+1 \\ \boldsymbol{\delta u}_i &= \boldsymbol{u}_i^{k+1} - \boldsymbol{u}_i^k \qquad \text{for} \quad i = 0, \dots, N.\end{aligned} \tag{4.3}$$

The linearized dynamics is defined as:

$$\boldsymbol{A}_i = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}_i},\ \boldsymbol{B}_i = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{u}_i},\ \boldsymbol{b}_i = \boldsymbol{h}_i - \boldsymbol{x}_{i+1}^k$$

The linear terms in the objective function are defined as:

$$\boldsymbol{q}_i = \nabla_{\boldsymbol{x_i}} \mathcal{L}, \quad \boldsymbol{r}_i = \nabla_{\boldsymbol{u_i}} \mathcal{L}, \quad \boldsymbol{p}_{N+1} = \nabla_{\boldsymbol{x}_{N+1}} \mathcal{L}.$$

Finally, the Quadratic terms are defined as:

$$\boldsymbol{P}_{N+1} = \nabla^2 \mathcal{L}_{N+1}, \quad \begin{bmatrix} \boldsymbol{Q}_i & \boldsymbol{S}_i^\top \\ \boldsymbol{S}_i & \boldsymbol{R}_i \end{bmatrix} = \nabla^2 \mathcal{L}_i$$

where we removed the dependency of $\boldsymbol{x}_i^k$ and $\boldsymbol{u}_i^k$ from the dynamics $\boldsymbol{h}$ and the cost function $\boldsymbol{l}$.

4.3.3 Primal & dual problem solution

As shown by [Kiessling et al., 2024], such QP can be efficiently solved by exploiting the problem structure. By applying the Riccati Recursion, starting from the N last node in the horizon and going backward in time we obtain:

$$\begin{aligned}
\boldsymbol{G}_i &= \boldsymbol{R}_i + \boldsymbol{B}_i^\top \boldsymbol{P}_{i+1} \boldsymbol{B}_i, \\
\boldsymbol{H}_i &= \boldsymbol{S}_i + \boldsymbol{B}_i^\top \boldsymbol{P}_{i+1} \boldsymbol{A}_i, \\
\boldsymbol{h}_i &= \boldsymbol{B}_i^\top (\boldsymbol{p}_{i+1} + \boldsymbol{P}_{i+1} \boldsymbol{b}_i) + \boldsymbol{r}_i, \\
\boldsymbol{K}_i &= -\boldsymbol{G}_i^{-1} \boldsymbol{H}_i, \\
\boldsymbol{k}_i &= -\boldsymbol{G}_i^{-1} \boldsymbol{h}_i, \\
\boldsymbol{P}_i &= \boldsymbol{Q}_i + \boldsymbol{A}_i^\top \boldsymbol{P}_{i+1} \boldsymbol{A}_i + \boldsymbol{K}_i^\top \boldsymbol{H}_i, \\
\boldsymbol{p}_i &= \boldsymbol{q}_i + \boldsymbol{A}_i^\top (\boldsymbol{p}_{i+1} + \boldsymbol{P}_{i+1} \boldsymbol{b}_i) + \boldsymbol{K}_i^\top \boldsymbol{h}_i.
\end{aligned} \tag{4.4}$$

where $\boldsymbol{K}_i$ and $\boldsymbol{k}_i$ are, respectively the feedback and feed-forward term of the optimal control policy. We can now retrieve the control and state vector applying recursively the forward pass starting from the initial condition as follows:

$$\boldsymbol{\delta u}_i = \boldsymbol{K}_i \boldsymbol{\delta x_i} + \boldsymbol{k}_i \quad \boldsymbol{\delta x}_{i+1} = \boldsymbol{A}_i \boldsymbol{\delta x}_i + \boldsymbol{B} \boldsymbol{\delta u}_i + \boldsymbol{b}_i \tag{4.5}$$

Once retrieved the values of $\boldsymbol{P}_i$ and $\boldsymbol{p}_i$ for $i = 0, \dots, N+1$ we can compute the update of the Lagrangian multiplier $\boldsymbol{\delta\lambda}$ in parallel. This is achieved by applying, at each node, the following update:

$$\boldsymbol{\delta\lambda}_i = \boldsymbol{P}_i \boldsymbol{\delta x}_i + \boldsymbol{p}_i \tag{4.6}$$

4.3.4 Parallel Associative Scan

In this section, we will briefly recap how parallel associative scans can be used to solve the primal problem as shown for the first time by [Särkkä and García-Fernández, 2023]. Given a sequence of elements $a_1, \dots, a_N$ and an associative operator $\otimes$ defined on them, a parallel associative scan can be used to evaluate the values $s_1, \dots, s_N$ as:

$$(s_1,\ s_2,\ \dots,\ s_N) = (a_1,\ a_1 \otimes a_2,\ \dots,\ a_1 \otimes a_2 \otimes \cdots \otimes a_N) \tag{4.7}$$

in $\mathcal{O}(\log N)$ as shown by [Blelloch, 1989]. This is possible since $\otimes$ is an associative operation, allowing the computation to be reorganized into smaller interdependent subproblems. To make use of this property, we first define the Conditional Value Function (CVF) $V_{i\to j}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as the minimal cost related to the optimal trajectory that goes from state $\boldsymbol{x}_i$ at node i in the horizon to state $\boldsymbol{x}_j$ at node j, with $i < j$. Given the connection between the value function of (4.3) and its CVF, $V_i(\boldsymbol{x}_i) = V_{i\to N+1}(\boldsymbol{x}_i, 0)$, we will use the scan to evaluate $V_i(\boldsymbol{x}_i)$ for $i = 0, \dots N+1$. We begin by initializing the elements a_i with the CVF $V_{i\to i+1}$ defined between two adjacent states $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$. Next, we define the operator $\otimes$ for dynamic programming as:

$$V_{i\to k} \otimes V_{k\to j} = \min_{\boldsymbol{x}_k} \{V_{i\to k} + V_{k\to j}\} \tag{4.8}$$

where $\boldsymbol{x}_k$ is a third intermediate state between i and j. Note that this operator is associative since the $\min$ operator is associative. We can now perform an associative scan on the CVF to obtain $V_{i\to N+1}$ in a single pass, thus $V_i(\boldsymbol{x}_i)$ for all i in the horizon. To derive the combination rule for (4.8), we start by noting that the CVF of (4.3) is also a quadratic program with affine equality constraint (4.1b), which can be represented in its dual form [Boyd and Vandenberghe, 2004] by introducing the Lagrangian multiplier $\boldsymbol{\eta}$:

$$\begin{aligned} V_{i\to j}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max_{\boldsymbol{\eta}} \frac{1}{2}\boldsymbol{x}_i^\top \tilde{\boldsymbol{P}}_{i,j}\boldsymbol{x}_i + \tilde{\boldsymbol{p}}_{i,j}^\top \boldsymbol{x}_i - \frac{1}{2}\boldsymbol{\eta}^\top \tilde{\boldsymbol{C}}_{i,j}\boldsymbol{\eta} + \\ - \boldsymbol{\eta}^\top \left(\boldsymbol{x}_j - \tilde{\boldsymbol{A}}_{i,j}\boldsymbol{x}_i - \tilde{\boldsymbol{b}}_{i,j}\right) \end{aligned} \tag{4.9}$$

where the tilde $\tilde{\cdot}$ is used to distinguish the values related to the conditional value function from the one in (4.4), while the subscripts are used to specify the initial and final state considered. Can be proved that given $V_{i\to k}$ and $V_{k\to j}$ that are in the same form of (4.9), $V_{i\to j}$ will still be in the form of (4.9) and is characterized by such combination rule:

$$\begin{aligned}
\tilde{\boldsymbol{P}}_{i,j} &= \tilde{\boldsymbol{A}}_{i,k}{}^{\top}\left(\boldsymbol{I}+\tilde{\boldsymbol{P}}_{k,j}\tilde{\boldsymbol{C}}_{i,k}\right)^{-1}\tilde{\boldsymbol{P}}_{k,j}\tilde{\boldsymbol{A}}_{i,k}+\tilde{\boldsymbol{P}}_{i,k},\\
\tilde{\boldsymbol{p}}_{i,j} &= \tilde{\boldsymbol{A}}_{k,j}{}^{\top}\left(\boldsymbol{I}+\tilde{\boldsymbol{P}}_{k,j}\tilde{\boldsymbol{C}}_{i,k}\right)^{-1}\left(\tilde{\boldsymbol{p}}_{k,j}-\tilde{\boldsymbol{P}}_{k,j}\boldsymbol{b}_{i,j}\right)+\tilde{\boldsymbol{p}}_{i,k},\\
\tilde{\boldsymbol{A}}_{i,j} &= \tilde{\boldsymbol{A}}_{k,j}\left(\boldsymbol{I}+\tilde{\boldsymbol{C}}_{i,k}\tilde{\boldsymbol{P}}_{k,j}\right)^{-1}\tilde{\boldsymbol{A}}_{i,k},\\
\tilde{\boldsymbol{C}}_{i,j} &= \tilde{\boldsymbol{A}}_{k,j}\left(\boldsymbol{I}+\tilde{\boldsymbol{C}}_{i,k}\tilde{\boldsymbol{P}}_{k,j}\right)^{-1}\tilde{\boldsymbol{C}}_{i,k}\tilde{\boldsymbol{A}}_{k,j}{}^{\top}+\tilde{\boldsymbol{C}}_{k,j},\\
\tilde{\boldsymbol{b}}_{i,j} &= \tilde{\boldsymbol{A}}_{k,j}\left(\boldsymbol{I}+\tilde{\boldsymbol{C}}_{i,k}\tilde{\boldsymbol{P}}_{k,j}\right)^{-1}\left(\tilde{\boldsymbol{b}}_{i,k}-\tilde{\boldsymbol{C}}_{i,k}\boldsymbol{p}_{k,j}\right)+\tilde{\boldsymbol{b}}_{j,k}.
\end{aligned} \tag{4.10}$$

We now initialize the values a_i of a reverse associative scan, using the conditional value function $V_{i\to i+1}$ characterized by:

$$\begin{aligned}
\tilde{\boldsymbol{A}}_{i,i+1} &= \boldsymbol{A}_i-\boldsymbol{B}_i\boldsymbol{R}_i^{-1}\boldsymbol{S}_i^{\top}, \quad \tilde{\boldsymbol{P}}_{i,i+1}=\boldsymbol{Q}_i-\boldsymbol{S}_i\boldsymbol{R}_i^{-1}\boldsymbol{S}_i^{\top}\\
\tilde{\boldsymbol{C}}_{i,i+1} &= \boldsymbol{B}_i{\boldsymbol{R}_i}^{-1}\boldsymbol{B}_i^{\top}, \qquad \tilde{\boldsymbol{p}}_{i,i+1}=\boldsymbol{q}_i-\boldsymbol{S}_i\boldsymbol{R}_i^{-1}\boldsymbol{r}_i\\
\tilde{\boldsymbol{b}}_{i,i+1} &= \boldsymbol{b}_i-\boldsymbol{B}_i\boldsymbol{R}_i^{-1}\boldsymbol{r}_i,
\end{aligned} \tag{4.11}$$

while $V_{N,N+1}$ by :

$$\begin{aligned}
\tilde{\boldsymbol{P}}_{N,N+1} &= \boldsymbol{Q}_{N+1}, \quad \tilde{\boldsymbol{A}}_{N,N+1}=0, \quad \tilde{\boldsymbol{b}}_{N,N+1}=0\\
\tilde{\boldsymbol{p}}_{N,N+1} &= \boldsymbol{q}_{N+1} \qquad \tilde{\boldsymbol{C}}_{N,N+1}=0
\end{aligned} \tag{4.12}$$

From the solution of the parallel associative scan using the combination rule in (4.10), we obtain $V_{i\to N+1}(\boldsymbol{x}_i,\boldsymbol{x}_{N+1})$ for $i=0,\dots,N+1$. Finally, recalling that $V_{i\to N+1}(\boldsymbol{x}_i,0)$ is equal to the value function $V_i(x_i)$ of the problem in (4.3) and that the value function is also in the same form of (4.9) we get that $\boldsymbol{P}_i=\tilde{\boldsymbol{P}}_{i,N+1}$ and $\tilde{\boldsymbol{p}}_i=\boldsymbol{p}_{i,N+1}$. This means that we obtained all $\boldsymbol{P}_i$ and $\boldsymbol{p}_i$ for $i=0,\dots,N+1$ that can then be used in (4.3.3) to calculate $\boldsymbol{K}$ and $\boldsymbol{k}$ in on pass in parallel. Thanks to a GPU-accelerated scan implementation, we can obtain the control policy over the entire horizon with an overall computational complexity of $\mathcal{O}(\log^2(n)\log N+\log^2(m))$. This follows from the fact that both computing and initializing the scan involve a matrix inversion, which can be performed in parallel with complexity $\mathcal{O}(\log^2(n))$ [Csanky, 1976]. The parallel associative scan can also be used to retrieve the optimal trajectory in $\mathcal{O}(\log n\log N+\log m)$. We start plugging into the

linearized dynamics the optimal control law we evaluated, obtaining:

$$\boldsymbol{\delta x}_{i+1} = \bar{\boldsymbol{A}}_i \boldsymbol{\delta x}_i + \bar{\boldsymbol{b}}_i \tag{4.13}$$

where: $\bar{\boldsymbol{A}}_i = \boldsymbol{A}_i + \boldsymbol{B}_i \boldsymbol{K}_i$ and $\bar{\boldsymbol{b}}_i = \boldsymbol{B}_i \boldsymbol{k}_i + \boldsymbol{b}_i$ We can define the conditional optimal trajectory as follows:

$$\bar{\boldsymbol{h}}_{i \to j}(\boldsymbol{\delta x}_i, \boldsymbol{\delta x}_j) = \bar{\boldsymbol{A}}_{i,j} \boldsymbol{\delta x}_i + \bar{\boldsymbol{b}}_{i,j} \tag{4.14}$$

[Särkkä and García-Fernández, 2023] showed that from the combination of two conditional optimal trajectory $\boldsymbol{h}_{i \to k}$ and $\boldsymbol{h}_{k \to j}$ we obtain $\boldsymbol{h}_{i \to j}$ that can be written in the same form as (4.14) with:

$$\bar{\boldsymbol{A}}_{i,j} = \bar{\boldsymbol{A}}_{i,k} \bar{\boldsymbol{A}}_{k,j} \quad \bar{\boldsymbol{b}}_{i,j} = \bar{\boldsymbol{A}}_{k,j} \bar{\boldsymbol{b}}_{i,k} + \bar{\boldsymbol{b}}_{k,j} \tag{4.15}$$

We can initialize the element of parallel scan $a_i = \boldsymbol{h}_{i \to i+1}(\boldsymbol{x}_i)$ as:

$$\bar{\boldsymbol{A}}_{i,i+1} = \bar{\boldsymbol{A}}_i \quad \bar{\boldsymbol{b}}_{i,i+1} = \bar{\boldsymbol{b}}_i \tag{4.16}$$

for $i = 1, \ldots, N$ while for $i = 0$ we set $\bar{\boldsymbol{A}}_{0,1} = \boldsymbol{0}$ and $\bar{\boldsymbol{b}}_{0,1} = \bar{\boldsymbol{A}}_0 \boldsymbol{\delta x}_0 + \bar{\boldsymbol{b}}_0$. From the parallel scan, we obtained the updated optimal state as:

$$\boldsymbol{\delta x}_i = a_0 \otimes a_1 \otimes \cdots \otimes a_{i-1} \tag{4.17}$$

4.3.5 Parallel Line Search

Once we update the optimal directions $\boldsymbol{\delta x}$, $\boldsymbol{\delta u}$, and $\boldsymbol{\delta \lambda}$, we can perform a backtracking line search to calculate the step length α to finally get the new guess as:

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \alpha \boldsymbol{\delta x} \quad \boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k + \alpha \boldsymbol{\delta \lambda} \\ \boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + \alpha \boldsymbol{\delta u} \end{aligned} \tag{4.18}$$

The SQP linear rollout allows for the computation to be easily performed in parallel for the horizon. To evaluate α, we implemented a filter line search similar to the one implemented in [Wächter and Biegler, 2006]. The filter line search ensures that the step taken in the iteration reduces the constraint violation or the cost. Such a method has already been successfully deployed for legged robot control in [Grandia et al., 2023]. We measure the

satisfaction of the dynamics constraints as:

$$\theta = \sum_{i=0}^{N+1} \|\boldsymbol{x}_{i+1} - \boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i)\|_2 \tag{4.19}$$

As for [Grandia et al., 2023], when the constraint violation θ is higher than a threshold, we reject the step if it further increases θ. If the violation is below the threshold and the current step is a descent direction, then we apply an Armijo condition [Armijo, 1966]. On the other hand, if the current iterate is not in a descent direction, we require that at least the cost or θ is decreased to accept the step. To leverage GPU parallelism, we evaluate the line-search in parallel (at a fixed grid of ten step sizes, i.e. $\alpha \in \{2^0, ..., 2^{-9}\}$) and select the largest α that satisfies one of the line-search conditions.

4.3.6 Practical Implementation

As it is commonly done in practice, we also perform a Gauss-Newton approximation, ignoring the second-order term of the dynamics in (4.3). We can further consider that in our case, the cost for the MPC can be expressed as a non-linear least squares problem in the form $\frac{1}{2}\sum_{i=0}^{N} \|\epsilon(x,u)\|_W$, where W is a weight matrix. The Hessian approximation reduces to:

$$\begin{aligned} \boldsymbol{Q}_{N+1} &= \nabla_x \boldsymbol{\epsilon}_{N+1}^\top W_{N+1} \nabla_x \boldsymbol{\epsilon}_{N+1} & \boldsymbol{S}_i &= \nabla_x \boldsymbol{\epsilon}_i^\top W_i \nabla_u \boldsymbol{\epsilon}_i \\ \boldsymbol{Q}_i &= \nabla_x \boldsymbol{\epsilon}_i^\top W_i \nabla_x \boldsymbol{\epsilon}_i & \boldsymbol{R}_i &= \nabla_u \boldsymbol{\epsilon}_i^\top W_i \nabla_u \boldsymbol{\epsilon}_i \end{aligned} \tag{4.20}$$

Such an approximation also guarantees that 4.3 is convex. As described in Sec. 4.3.1, the inequality constraints in (4.1c) and (4.1d) were not explicitly included in the solver formulation. However, these constraints are critical for MPC as they represent the physical limitations of the system, such as friction cones or joint limits. To account for these limitations in the optimization, we incorporated them as a relaxed barrier function:

$$\mathcal{B}(\xi, \mu, \delta) \begin{cases} -\mu \ln(\xi), & \text{if } \xi \geq \delta, \\ \frac{\mu}{2}\left(\left(\frac{\xi - 2\delta}{\delta}\right)^2 - 1\right) - \mu \ln(\delta), & \text{if } \xi < \delta. \end{cases} \tag{4.21}$$

where $\xi(\boldsymbol{x}, \boldsymbol{u}) < 0$ is the considered constraint, and μ and δ are parameters tuned for each constraint. Following the example of [Grandia et al., 2023], to maintain the convexity of

4.3 the hessian approximation of the constraints is constructed as

$$\begin{aligned}
\boldsymbol{Q}_{N+1} &= \nabla_x \boldsymbol{\epsilon}_{N+1}^\top W_T \nabla_x \boldsymbol{\epsilon}_{N+1} + \nabla_x \boldsymbol{\xi}_{N+1}^\top \nabla_{\xi\xi} \mathcal{B}_T \nabla_x \boldsymbol{\xi}_{N+1} \\
\boldsymbol{Q}_i &= \nabla_x \boldsymbol{\epsilon}_i^\top W_i \nabla_x \boldsymbol{\epsilon}_i + \nabla_x \boldsymbol{\xi}_i^\top \nabla_{\xi\xi} \mathcal{B}_i \nabla_x \boldsymbol{\xi}_i \\
\boldsymbol{S}_i &= \nabla_x \boldsymbol{\epsilon}_i^\top W_i \nabla_x \boldsymbol{\epsilon}_i + \nabla_x \boldsymbol{\xi}_i^\top \nabla_{\xi\xi} \mathcal{B}_i \nabla_u \boldsymbol{\xi}_i \\
\boldsymbol{R}_i &= \nabla_u \boldsymbol{\epsilon}_i^\top W_i \nabla_u \boldsymbol{\epsilon}_i + \nabla_u \boldsymbol{\xi}_i^\top \nabla_{\xi\xi} \mathcal{B}_i \nabla_u \boldsymbol{\xi}_i
\end{aligned} \tag{4.22}$$

As a common practice, while using the MPC in closed-loop with the robot, we perform one iteration only of the algorithm described in Sec. 4.3.1 per control loop. To increase the robustness of the method, we warm-start each call of the solver with the previous prediction just shifted by one time-step as shown by [Gros et al., 2020].

Finally, to guarantee the differentiability of the solver through JAX-based automatic differentiation, we avoided the use of $while$ loop in the code. Especially in line-search, we evaluate a fixed number of alpha values in parallel.

4.4 Model Predictive Control

In this section, we describe briefly the dynamics model $\boldsymbol{h}(\boldsymbol{x}_i, \boldsymbol{u}_i)$ presented in (4.1b), introducing two different models used for the formulation of the OCP presented in the result Sec. 4.5. The efficacy of the control input obtained from a MPC strategy is directly related to the rate at which such a control strategy can be re-planned and to the accuracy of its predictions, as shown by [Khazoom et al., 2024]. Therefore, we consider two main formulations from the literature: one based on the Single Rigid Body Dynamics (SRBD) model and one based on the Whole Body (WB) model. The first approximates the system as a single body with constant inertia, discarding the limb joint positions and the inertial variation from different limb configurations. This model is particularly effective in quadruped robots, where the limb mass is, in most cases, designed to be less than 10% of the total mass [Bledt et al., 2018]. On the other hand, the latter completely captures the dynamics of the robot, enabling the maximum exploitation of the system's capabilities. This benefit comes at the cost of handling a larger state space and highly non-convex models, thus affecting the maximum frequency at which the optimization problem can be solved. Moreover, the SRBD-MPC state is limited to the center of mass linear and angular positions and velocities. This means that, when representing the robot's orientation using quaternions, the state lies in $\mathbb{R}^7$. For such model, the control input consists only of the Ground Reaction Forces (GRF), and thus lies in $\mathbb{R}^{3n_c}$, where n_c is the number of contact

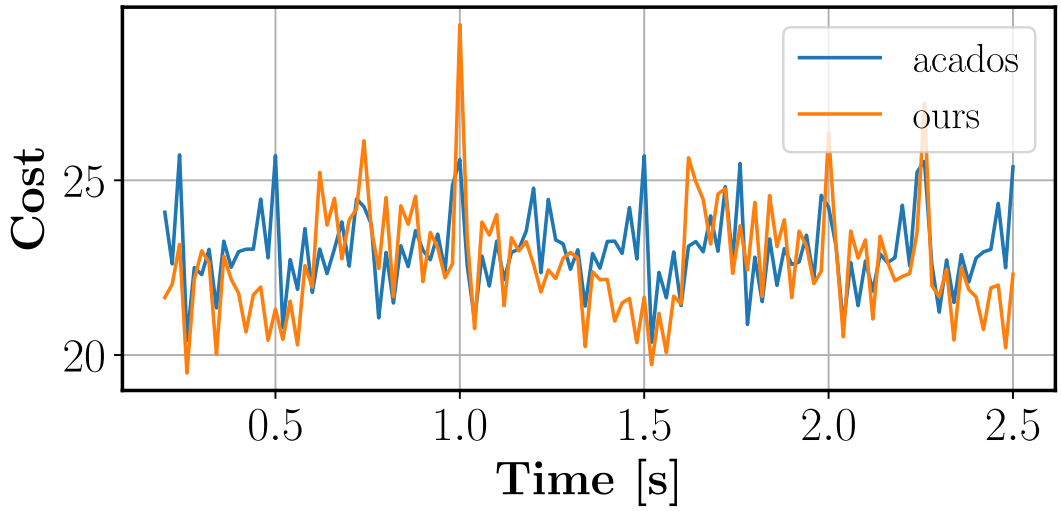

Figure 4.2 Comparison of the MPC optimal cost, evaluated from the same initial condition with acados [Verschueren et al., 2021] (in blue) and with our implementation (in orange). Both solvers are evaluated at each call of the MPC and the costs are recorded.

points, meaning that, for the evaluations with the quadruped, in Sec. 4.5 n is in $\mathbb{R}^{13}$ and m in $\mathbb{R}^{12}$.

In contrast, the WB-MPC also includes joint positions and velocities in the state, and joint torques in the control input. This increases the state dimensionality to $\mathbb{R}^{13+2n_{\text{joint}}}$ and the control input dimensionality to $\mathbb{R}^{3n_c+n_{\text{joint}}}$, that translate for the quadruped model in Sec. 4.5 in n in $\mathbb{R}^{37}$ and m in $\mathbb{R}^{24}$.

Beyond the difference in dimensionality, the WB model also introduces greater complexity. While the SRBD model features bilinear terms only in the angular dynamics, the WB model has highly nonlinear dependencies on the joint configuration. As a result, the overall OCP becomes significantly harder to solve.

4.5 Result

In this section, we present the results of various analyses that highlight the strength of the proposed approach. We evaluate the performance of our algorithm using the two models introduced in Sec. 4.4 (the SRBD and WB models). To benchmark our implementation against state-of-the-art solvers, we used acados [Verschueren et al., 2021] (SQP) for comparisons on the SRBD model and crocoddyl [Mastalli et al., 2020] (DDP) for the WB model. We used acados for the SRBD comparison, as it could be easily adapted for the evaluations with multiple robots, shown in Fig. 4.6 and 4.7. On the other hand, we used

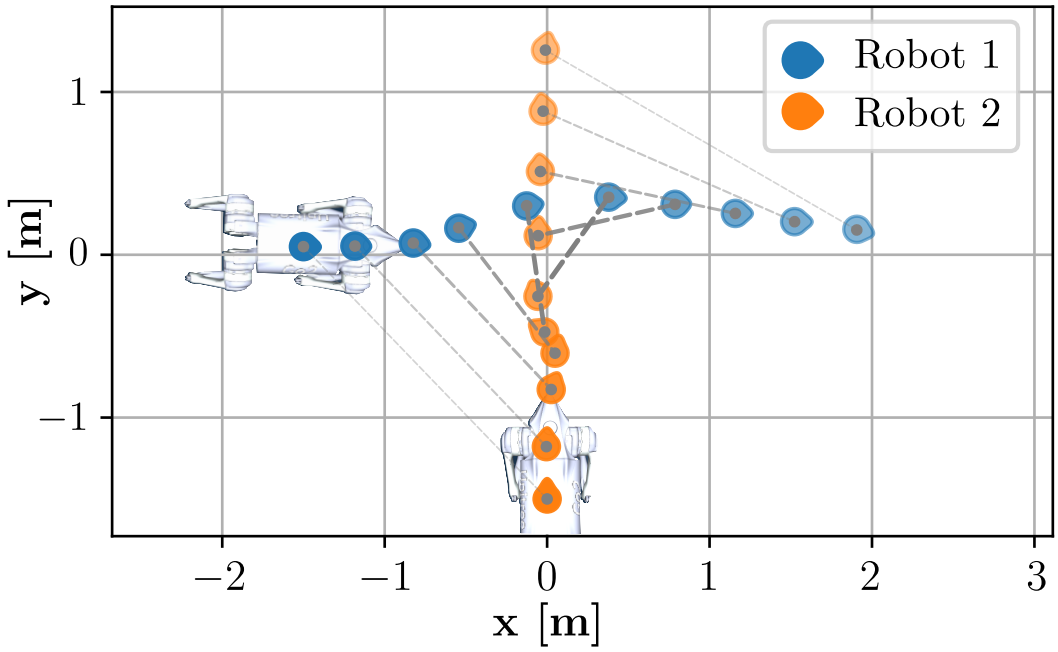

Figure 4.3 Path of the two robots tracking their desired trajectories that will lead to a collision. We visualize the distance between the two robots with dashed gray lines of different shades, the darker the smaller distance.

crocoddyl for the horizon length comparison with the WB model, as it showed significantly better performance than acados with the more complex model as shown in Fig. 4.4.

Our analysis includes investigating how the average solving time scales with the horizon length across all models and solvers. Additionally, we assess the solver's performance as the system state dimension increases, as well as the vectorization capabilities of our approach. All the solving time comparisons consider a single iteration for each algorithm. The tests were conducted on a desktop computer equipped with an Intel Core i7-13700KF and an NVIDIA RTX 3080. We used Mujoco XLA, to evaluate the WB dynamics. In all the presented benchmarks, if not stated differently, we used the 15 kg Unitree Go2 robot, a torque-controlled quadruped platform, with a horizon length N of 50 nodes. As Fig. 4.1 and the accompanying video of [Amatucci et al., 2025] show, our WB-MPC can also be implemented for real-time control of other robot morphologies, like humanoids.

4.5.1 Performance evaluation

First, we evaluate the ability of our solver to generate complex dynamic maneuvers. In this scenario, we are generating a barrel roll motion with the Aliengo robot, as shown in Fig. 4.1. Thanks to its multiple shooting formulation, we can initialize the solver with an infeasible

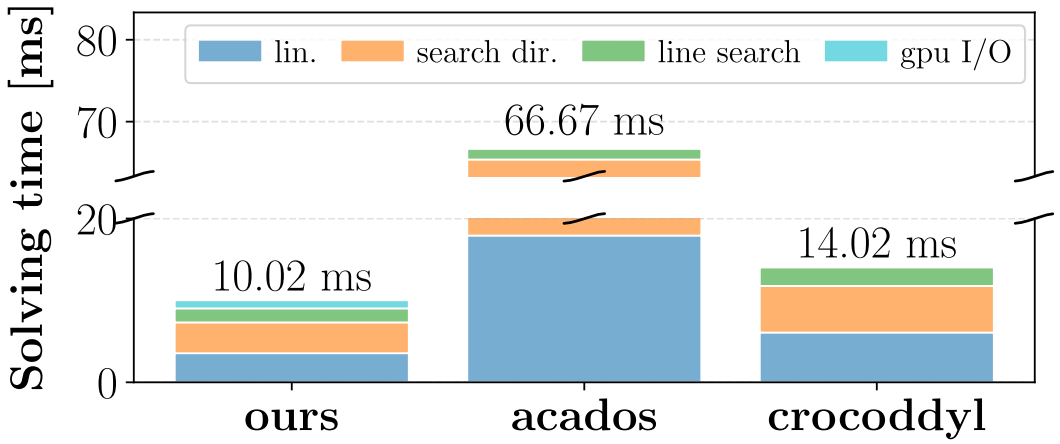

Figure 4.4 Breakdown of the solving times for our implementation, acados (SQP) and crocoddyl (FDDP).

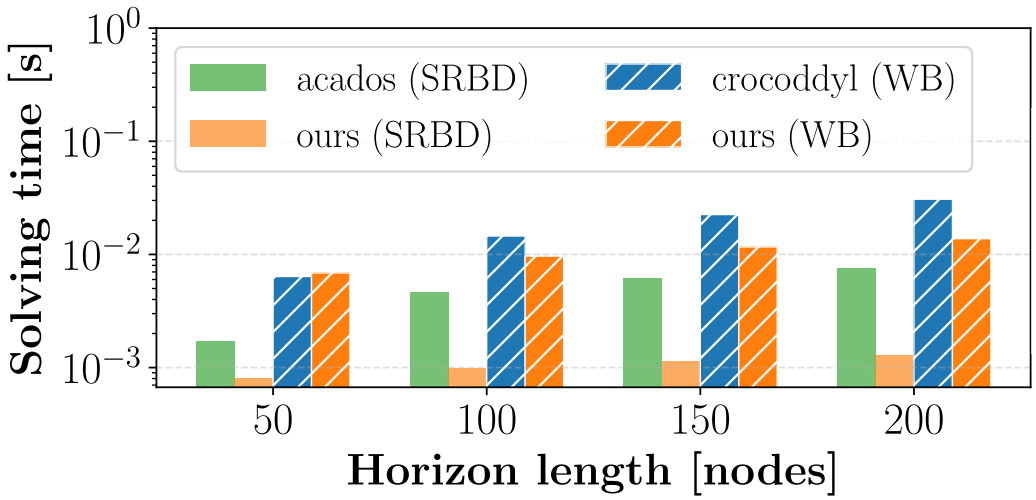

Figure 4.5 Average solving times when varying the horizon length (number of nodes) for the SRBD model (top) and WB model (bottom). In blue and green, we plot the solution time of two different state-of-the-art solvers (respectively acados [Verschueren et al., 2021] and crocoddyl [Mastalli et al., 2020]), while in orange, we plot the one of our implementation.

initial guess, and the solver converges in just 27 iterations to a solution that completes the task. We also assess the performance of the proposed solver in terms of solution optimality. In Fig. 4.2, we compared the values of the cost from the optimal solution given by our implementation against the one given by acados, which is chosen for this comparison because it allows using the same cost function as the one in our formulation. In this scenario, we are using the whole-body model and are controlling the robot in simulation at 50Hz. The robot is tasked to trot with a forward speed of 0.3 m/s while being randomly pushed by an external disturbance of 50N for 0.25s. At every control loop, we record the

value of the cost functional from both solvers, comparing solutions that start from the same initial condition. Figure 4.2 reports the values recorded during a 2.5s time span. Although the two solutions are comparably similar, our implementation results in a 20% reduction in the average cost along the trajectory.

To finally validate the MPC, we also performed experiments on the real hardware, as shown in Fig. 4.1 and in the accompanying video of [Amatucci et al., 2025]. For the real experiments, the solver runs on an Nvidia RTX 4050 laptop GPU. Figure 4.1 shows snapshots of the robot walking blindly up the steps and through uneven terrain using only proprioceptive data for state estimation [Nisticò et al., 2025]. We also evaluated the proposed approach when used to control two different robots at the same time in a centralized control implementation in simulation. In this example, the two robots are controlled by the same MPC that evaluates the control input for both systems at 50Hz. The MPC also includes a collision avoidance constraint, in the form of a quadratic penalty term. In the tested scenario, the two robots are tasked to track two perpendicular trajectories that cross each other and would lead to a collision. Figure 4.3 shows the resulting path of the two robots, where the dashed grey lines represent the distance between the two robots. As shown, both robots deviate from their desired paths to avoid collisions, demonstrating coordinated behavior. While being just a simple example, this result highlights the potential of our MPC in solving collaborative tasks, further implementation in more complex scenarios, such as collaborative carrying is left for future work. Figure 4.4 presents the average solving time breakdown for the three solvers in the quadruped trotting scenario with a horizon of N$=$100. It highlights the sequential backward pass in crocoddyl, the QP solver call in acados, and the parallel scan in our method for evaluating the search direction. A minor computational overhead of approximately $1ms$ arises from GPU communication, as only the initial state is sent and the commanded torques, joint positions, and velocities are retrieved.

4.5.2 Temporal Parallelization

In this subsection, we show the benefit of our implementation against the state-of-the-art solvers acados [Verschueren et al., 2021] and crocoddyl [Mastalli et al., 2020], in terms of average solving time for increasing horizon length. For this assessment, we initialized the robot in a feasible random pose and joint configuration and tasked the controller to perform a forward trot at a speed of 0.5m$/$s. Acados uses the same cost and dynamics formulation as in our controller, while for the comparison with crocoddyl we adapted our reference generator and cost to match the reference used in the example provided with

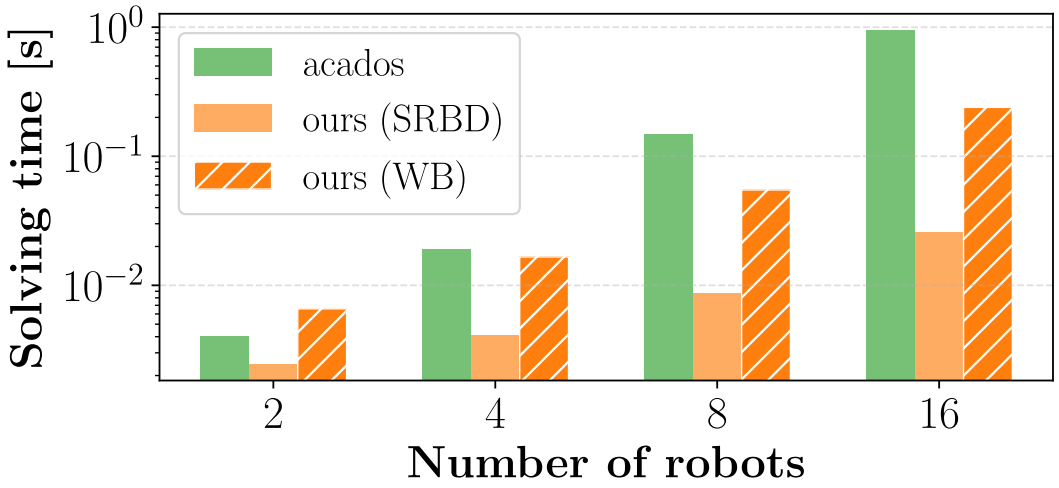

Figure 4.6 Our implementation can control multiple robots in the same optimization, a typical scenario found in centralized controllers. In green, we depict the solution time of acados, while in orange, ours, considering the SRBD model. Furthermore, in dashed orange, we report the result obtained by our implementation using the WB model.

crocoddyl's libraries. As reported in Fig. 4.5, our implementation shows faster update rates than both solvers in almost all tested scenarios. For the assesment against acados, with the SRBD model, our implementation is always faster. For the case of the WB formulation, crocoddyl is able to outperform our formulation with a prediction horizon smaller than 80 nodes. This last outcome is primarily attributed to overhead involved in transferring data to and from the GPU, which, for smaller problems, can be detrimental. Noticeably, our formulation can achieve an update rate of 50Hz for a horizon length of 200 nodes.

4.5.3 State Parallelization

In this subsection, we discuss the benefits of our implementation in the context of centralized controllers. In our tests, we evaluated the average solving times for the MPC while progressively increasing the number of robots included in the same optimization. This approach simulated a centralized controller managing n different robots. For the SRBD model, we compared our implementation against acados. For the WB model comparison, crocoddyl was excluded since its model interface does not allow for a customization that is needed for a fair evaluation. As Fig. 4.6 shows, our implementation outperformed acados when considering the SRBD model, achieving a computational time of 25 ms in the presence of 16 robots, enough for the real-time control of the systems. When using the WB model, instead, computation becomes significantly more demanding for the implementation with acados and we could not record data for the comparison. As shown in Fig. 4.6, our

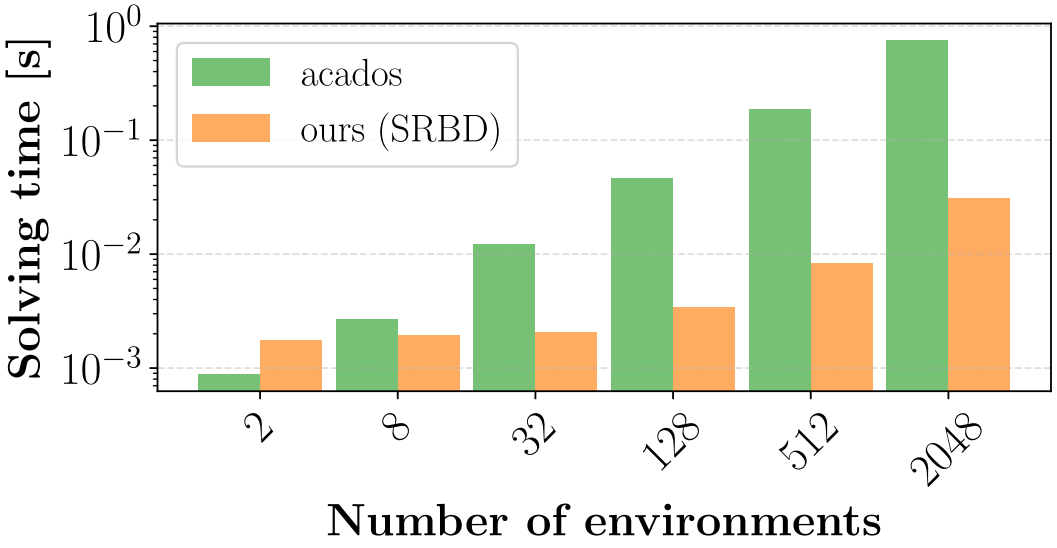

Figure 4.7 Comparison of the solving time achieved by our implementation against the batch-parallelized solver feature of acados, when varying the number of environments being simulated in parallel. In this example, each environment has a dedicated MPC.

algorithm successfully controls up to four robots (WB model case) at 20 Hz. To the best of the authors' knowledge, this level of performance has never been achieved before.

	n_{env}	Control Frequency	Real-Time Factor
SRBD-MPC	4096	50 Hz	370×
SRBD-MPC	4096	25 Hz	570×
WB-MPC	4096	50 Hz	35×
WB-MPC	4096	25 Hz	75×

Table 4.1 Performance metric

4.5.4 Model-Based Learning

In this subsection, we highlight the vectorization capabilities of our controller. Thanks to its full implementation in JAX, we can easily vectorize the computation of the MPC. Such ability is critical in the training scenario as shown in [Jenelten et al., 2024] or at runtime as shown in [Taouil et al., 2025]. In Fig. 4.7, we compare our approach to the batched version of acados, which uses OpenMP to parallelize the computation of independent MPC instances on the CPU. As the plot shows, acados is competitive with the proposed MPC only for small batch sizes. This limitation comes from the reduced number of cores available on a CPU compared to the parallelization capabilities of modern GPUs.

Finally, Table 4.1 summarizes the performance of our MPC when batched across multiple environments with Mujoco XLA as the simulator. In this setup, we generate 4096 environments, each one running its own MPC. For the SRBD model, we control the robot at an update rate of 50 Hz and we achieve a value of 570 seconds of simulated time per second. Meanwhile, for the WB model, closing the loop at 25 Hz, we achieve a real-time factor of 75x.

4.6 Discussion & Limitation

Here, we discuss aspects of the proposed approach that are not directly revealed by the reported results but are important for understanding its scope and limitations. Parallelizing the computation of the OCP solution requires a considerable amount of GPU memory. For the presented examples, memory usage ranged from approximately 100 MB to 1 GB of VRAM, depending on the model size and prediction horizon. Although modern GPUs offer more than enough memory to run a single MPC, to perform massively parallel simulations, the requirement grows substantially since the necessary memory size scales linearly with the number of environments. To mitigate this burden, our library includes the option to use a serialized backward pass, which drastically reduces memory consumption. Although this variant becomes slower for long horizons, it provides comparable performance for shorter ones. That said, for small problems, the benefits of GPU parallelization diminish, and the proposed method is no longer preferable. For example, for horizons below 50 nodes, our distributed optimization method presented in Chapter 2 outperforms the GPU-based solver for WB-MPC, due to the slower rate of the single computation in GPU and the considerable communication overhead between the CPU and GPU. Furthermore, as discussed earlier, the inequality constraints listed in (4.1) are handled through barrier terms in the cost function. This means that, even at convergence, we cannot guarantee strict satisfaction of these constraints. Although similar approaches have already been successfully deployed on real robots (e.g., [Grandia et al., 2023]), we aim to strengthen the guarantees of our method by incorporating a projection-based sweep in an outer loop, ensuring explicit constraint satisfaction.

4.7 Conclusion

This work presents a novel GPU-accelerated MPC framework implemented for legged robot locomotion. By leveraging the parallel processing capabilities of GPUs and implementing a Primal-Dual iLQR solver in JAX, we achieve logarithmic scaling in horizon length and square-log scaling with state and control dimensions. Our approach demonstrates significant improvements over state-of-the-art solvers, achieving higher computational efficiency and scalability. This allows for the optimization of centralized controllers for multiple robots and the integration of large-scale parallel environments, enhancing learning-based control frameworks.

Chapter 5

Conclusions

In the pursuit of general-purpose robots capable of performing a wide variety of tasks and adapting to different environments, Model Predictive Control (MPC) has emerged as a powerful framework for the formulation of the planning and control problem in a unified way. In this thesis, we have presented methods to improve the performance of MPC-based controllers for legged robots, focusing specifically on the computational bottlenecks that affect MPC's deployment.

First, we introduced a distributed optimization method that exploits multicore CPU architectures to accelerate the solution of the optimal control problem. This framework is designed to enable Whole Body (WB)-MPC to run on onboard hardware at frequencies sufficient for direct control of the robot, without relying on hierarchical control architectures such as instantaneous Quadratic Programming (QP)-based controllers, which can introduce suboptimal behaviors and require an additional set of gains to be tuned. The proposed architecture applies to all Optimal Control Problem (OCP) solvers based on Newton's method and can be extended to other robotic platforms and tasks beyond legged locomotion.

Next, we proposed a hierarchical framework that solves the contact-planning problem online by reformulating the original Mathematical Program with Complementarity Constraints (MPCC) as a Markov Decision Process (MDP), which is then solved using a Monte Carlo Tree Search (MCTS) algorithm bootstrapped with a learning-based Value Function (VF). We decided to tackle the contact-planning problem because it is widely regarded as one of the main challenges in MPC for legged robots. This difficulty primarily arises because contact modeling introduces complementarity constraints in the OCP, which violate Linear Independence Constraint Qualification (LICQ), a fundamental regularity condition required by standard solvers. At the same time, contact optimization is essential for legged robot control, as the set of active contacts determines both the system's reachability space

and, more importantly, controllability, e.g., a quadruped standing on only two feet cannot generate reaction torques about the line connecting the two support points, thereby limiting its balancing capability. By enabling online contact-planning, our hierarchical approach significantly improves the robustness and versatility of the MPC controller, allowing the robot to adapt its contact sequence in response to unforeseen disturbances.

Finally, we presented a novel solver optimized for Graphics Processing Unit (GPU). Our formulation leverages the massive parallelism of modern hardware accelerators to speed up the solution of the OCP, enabling the deployment of MPC with longer horizons and for systems with a high number of Degrees of Freedoms (DoFs). These benefits are particularly important in complex environments that require extended planning horizons. This is the case for tasks like navigating sparse stepping stones, or when combining Moving Horizon Estimator (MHE) and MPC into a single optimization. In this latter case, part of the horizon is used to look backward in time and estimate the current state of the system, eliminating the errors caused by communication delays between the two modules. Our formulation also enables the design of a centralized controller for multiple robots performing the same collaborative task, which is essential in scenarios where a single robot is physically incapable of completing the task alone, such as carrying heavy loads. In addition, the solver is differentiable and highly parallelizable, making it suitable for learning-based applications. We can simulate thousands of robots in parallel and compute gradients of the OCP solution with respect to arbitrary parameters. This opens up new possibilities for integrating MPC with modern machine-learning techniques, including Reinforcement Learning (RL) and imitation learning to further enhance the capabilities of legged robots.

5.1 Distinctions & Interconnections

While we have already mentioned the interconnections between our contributions throughout the thesis, we take this opportunity to clarify the distinct domains for which each method is best suited and discuss how they could be integrated in one framework. Starting from an important point, it could seem that the later GPU-base solver is the obvious default choice for any MPC application, but as we already pointed out, the distributed optimization approach was designed to fully exploit the computational capabilities of Central Processing Unit (CPU)s. Although its computational complexity scales less favorably than that of the GPU-based framework, the higher single-core performance of CPUs, and the absence of communication overhead between the CPU and GPU make the distributed optimization approach more suitable for smaller problems that do not require long horizons

or the coordination of multiple robots. While the two MPC formulations can be seen as alternatives to be selected depending on their respective trade-offs, the MCTS-based gait planner should be viewed as a plug-and-play module that can be seamlessly integrated into the pipeline of either control framework, immediately enhancing locomotion robustness. The capabilities of the MCTS-based gait planner could be further improved by leveraging the massive parallelization offered by the GPU-based solver. Such an integration would drastically increase the number of simulations that can be performed within the same time budget, leading to improved performance. This integration represents a promising and natural direction for further investigation.

5.2 Future Works

In the future, we plan to further extend the work presented in this thesis in several directions. First, building on the insights gained from [Taouil et al., 2025] and [Amatucci et al., 2024a], we aim to combine contact planning with the distributed optimization framework in the context of manipulation tasks. In this scenario, the object to be manipulated and the robot are treated as two separate subsystems, and the consensus-based Alternating Direction Method of Multipliers (ADMM) ensures that their interactions are dynamically consistent at convergence. The idea is to avoid enforcing complementarity constraints within each individual MPC subproblem and instead handle them at the outer consensus level. This approach could be further enhanced by sampling multiple initial guesses and solving the resulting problems in batches using the GPU-based solver introduced in [Amatucci et al., 2025]. We also plan to explore the integration of learning-based methods with our MPC framework. By leveraging the differentiable nature of our GPU-accelerated solver, we can investigate the use of RL techniques to learn control policies that can be directly incorporated into the MPC structure, potentially improving performance in complex and dynamic environments. Furthermore, we envision learning a differentiable contact model, possibly including both contact prediction and detection using a single Multi-Layer Perceptron (MLP) architecture. This could enable an end-to-end MPC controller capable of processing visual and proprioceptive inputs and outputting joint torques directly. Such an approach would significantly enhance the robot's ability to adapt to unstructured environments and perform a wide range of tasks autonomously. With respect to pure RL approaches, we believe that MPC can provide a strong inductive bias that can improve sample efficiency and generalization, while learning-based methods can help overcome some of the limitations of traditional MPC, such as modeling discontinuity.

References

[4Ocean, 2025] 4Ocean (last accessed on 19/DEC/2025). Bebot. https://www.4ocean.com/pages/4ocean-x-poralu-bebot.

[Aceituno-Cabezas et al., 2017] Aceituno-Cabezas, B., Mastalli, C., Dai, H., Focchi, M., Radulescu, A., Caldwell, D. G., Cappelletto, J., Grieco, J. C., Fernández-López, G., and Semini, C. (2017). Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538.

[Akizhanov et al., 2024] Akizhanov, R., Dhédin, V., Khadiv, M., and Laptev, I. (2024). Learning feasible transitions for efficient contact planning. *arXiv preprint arXiv:2407.11788*.

[Amatucci et al., 2022] Amatucci, L., Kim, J.-H., Hwangbo, J., and Park, H.-W. (2022). Monte carlo tree search gait planner for non-gaited legged system control. In *2022 IEEE ICRA*, pages 4701–4707.

[Amatucci et al., 2025] Amatucci, L., Sousa-Pinto, J., Turrisi, G., Orban, D., Barasuol, V., and Semini, C. (2025). Primal-dual ilqr for gpu-accelerated learning and control in legged robots. *IEEE Robotics and Automation Letters*, pages 1–8.

[Amatucci et al., 2024a] Amatucci, L., Turrisi, G., Bratta, A., Barasuol, V., and Semini, C. (2024a). Accelerating model predictive control for legged robots through distributed optimization. In *IEEE/RSJ IROS*.

[Amatucci et al., 2024b] Amatucci, L., Turrisi, G., Bratta, A., Barasuol, V., and Semini, C. (2024b). Vero: A vacuum-cleaner-equipped quadruped robot for efficient litter removal. *Journal of Field Robotics*, 41(6):1829–1842.

[Andersson et al., 2019] Andersson, J. A. E., Gillis, J., Rawlings, G. H. J. B., and Diehl, M. (2019). CasADiA software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36.

[Angsa Robotics, 2025] Angsa Robotics (last accessed on 19/DEC/2025). Angsa. *https://angsa-robotics.com/en-de/*.

[Araújo and Costa, 2019] Araújo, M. C. B. and Costa, M. F. (2019). A critical review of the issue of cigarette butt pollution in coastal environments. *Environmental Research*, 172:137–149.

[Armijo, 1966] Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3.

[Bai et al., 2018] Bai, J., Lian, S., Liu, Z., Wang, K., and Liu, D. (2018). Deep learning based robot for automatically picking up garbage on the grass. *IEEE Transactions on Consumer Electronics*, 64(3):382–389.

[Barasuol et al., 2013] Barasuol, V., Buchli, J., Semini, C., Frigerio, M., De Pieri, E. R., and Caldwell, D. G. (2013). A reactive controller framework for quadrupedal locomotion on challenging terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2554–2561.

[Baumgarte, 1972] Baumgarte, J. (1972). Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16.

[Belzagui et al., 2020] Belzagui, F., Buscio, V., Gutiérrez-Bouzán, C., and Vilaseca, M. (2020). Cigarette butts as a microfiber source with a microplastic level of concern. *The Science of the total environment*, 762:144165.

[Bertsekas, 2024] Bertsekas, D. P. (2024). Model predictive control and reinforcement learning: A unified framework based on dynamic programming. *arXiv preprint arXiv:2406.00592*.

[Bhuyan, 2022] Bhuyan, M. S. (2022). Effects of microplastics on fish and in human health. *Frontiers in Environmental Science*, 10.

[Bishop et al., 2024] Bishop, A. L., Zhang, J. Z., Gurumurthy, S., Tracy, K., and Manchester, Z. (2024). Relu-qp: A gpu-accelerated quadratic programming solver for model-predictive control. In *IEEE ICRA*, pages 13285–13292.

[Bledt and Kim, 2019] Bledt, G. and Kim, S. (2019). Implementing regularized predictive control for simultaneous real-time footstep and ground reaction force optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6316–6323.

[Bledt et al., 2018] Bledt, G., Powell, M. J., Katz, B., Di Carlo, J., Wensing, P. M., and Kim, S. (2018). Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In *IEEE/RSJ IROS*, pages 2245–2252.

[Blelloch, 1989] Blelloch, G. (1989). Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538.

[Bochkovskiy et al., 2020] Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv*.

[Boyd et al., 2011] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1).

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

[Bradbury et al., 2018] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs. Accessed 20 Feb 2024.

[Bratta et al., 2024] Bratta, A., Meduri, A., Focchi, M., Righetti, L., and Semini, C. (2024). Contactnet: Online multi-contact planning for acyclic legged robot locomotion. In *2024 UR*, pages 747–754.

[Bravo-Palacios and Wensing, 2022] Bravo-Palacios, G. and Wensing, P. M. (2022). Large-scale admm-based co-design of legged robots. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8842–8849.

[Budhiraja et al., 2019] Budhiraja, R., Carpentier, J., and Mansard, N. (2019). Dynamics consensus between centroidal and whole-body models for locomotion of legged robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6727–6733.

[Cheng et al., 2023] Cheng, X., Shi, K., Agarwal, A., and Pathak, D. (2023). Extreme parkour with legged robots. In *Corl*.

[Corbères et al., 2023] Corbères, T., Mastalli, C., Merkt, W., Havoutis, I., Fallon, M., Mansard, N., Flayols, T., Vijayakumar, S., and Tonneau, S. (2023). Perceptive locomotion through whole-body mpc and optimal region selection. *arXiv preprint arXiv:2305.08926*.

[Csanky, 1976] Csanky, L. (1976). Fast parallel matrix inversion algorithms. *SIAM Journal on Computing*, 5(4):618–623.

[Dantec et al., 2022] Dantec, E., Naveau, M., Fernbach, P., Villa, N., Saurel, G., Stasse, O., Taix, M., and Mansard, N. (2022). Whole-body model predictive control for biped locomotion on a torque-controlled humanoid robot. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 638–644.

[de Santos et al., 2012] de Santos, P., Garcia, E., and Estremera, J. (2012). *Quadrupedal Locomotion: An Introduction to the Control of Four-legged Robots*. Springer London.

[Deits and Tedrake, 2014] Deits, R. and Tedrake, R. (2014). Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS Humanoids*, pages 279–286.

[Deng et al., 2017] Deng, W., Lai, M.-J., Peng, Z., and Yin, W. (2017). Parallel multi-block admm with o(1/k) convergence. *Journal of Scientific Computing*, 71(2):712–736.

[Dhédin et al., 2024] Dhédin, V., Chinnakkonda Ravi, A. K., Jordana, A., Zhu, H., Meduri, A., Righetti, L., Schölkopf, B., and Khadiv, M. (2024). Diffusion-based learning of contact plans for agile locomotion. In *2024 IEEE-RAS Humanoids*, pages 637–644. IEEE.

[Diederik, 2015] Diederik, P. K. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

[Ding et al., 2021] Ding, Y., Pandala, A., Li, C., Shin, Y.-H., and Park, H.-W. (2021). Representation-free model predictive control for dynamic motions in quadrupeds. *IEEE Transactions on Robotics*, 37(4):1154–1171.

[Fahmi et al., 2023] Fahmi, S., Barasuol, V., Esteban, D., Villarreal, O., and Semini, C. (2023). Vital: Vision-based terrain-aware locomotion for legged robots. *IEEE Transactions Robotics*, 39(2):885–904.

[Fahmi et al., 2019] Fahmi, S., Mastalli, C., Focchi, M., and Semini, C. (2019). Passive whole-body control for quadruped robots: Experimental validation over challenging terrain. *IEEE Robotics and Automation Letter*, 4(3):2553–2560.

[Farshidian et al., 2023] Farshidian, F. et al. (2023). OCS2: An open source library for optimal control of switched systems. [Online]. Available: https://github.com/leggedrobotics/ocs2.

[Farshidian et al., 2017] Farshidian, F., Jelavic, E., Satapathy, A., Giftthaler, M., and Buchli, J. (2017). Real-time motion planning of legged robots: A model predictive control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 577–584.

[Featherstone, 1987] Featherstone, R. (1987). *Robot Dynamics Algorithm*. Springer New York, NY.

[Frasch et al., 2015] Frasch, J. V., Sager, S., and Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*.

[Giftthaler et al., 2018] Giftthaler, M., Neunert, M., Stäuble, M., Buchli, J., and Diehl, M. (2018). A family of iterative gauss-newton shooting methods for nonlinear optimal control. In *IEEE/RSJ IROS*, pages 1–9.

[Grandia et al., 2019] Grandia, R., Farshidian, F., Dosovitskiy, A., Ranftl, R., and Hutter, M. (2019). Frequency-aware model predictive control. *IEEE Robotics and Automation Letters*, 4(2):1517–1524.

[Grandia et al., 2023] Grandia, R., Jenelten, F., Yang, S., Farshidian, F., and Hutter, M. (2023). Perceptive locomotion through nonlinear model-predictive control. *IEEE Transactions on Robotics*, 39(5):3402–3421.

[Gros et al., 2020] Gros, S., Zanon, M., Quirynen, R., Bemporad, A., and Diehl, M. (2020). From linear to nonlinear mpc: bridging the gap via the real-time iteration. *International Journal of Control*, 93(1):62–80.

[Hartley et al., 2019] Hartley, R., Jadidi, M. G., Eustice, R. M., and Grizzle, J. W. (2019). Contact-aided invariant extended kalman filtering for robot state estimation. *The International Journal of Robotics Research*, 39:402 – 430.

[Herzog et al., 2015] Herzog, A., Rotella, N., Schaal, S., and Righetti, L. (2015). Trajectory generation for multi-contact momentum control. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, page 874–880. IEEE.

[Hong et al., 2020] Hong, S., Kim, J.-H., and Park, H.-W. (2020). Real-time constrained nonlinear model predictive control on so(3) for dynamic legged locomotion. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3982–3989.

[Howell et al., 2019] Howell, T. A., Jackson, B. E., and Manchester, Z. (2019). Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679.

[Hwangbo et al., 2019] Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872.

[Hwangbo et al., 2018] Hwangbo, J., Lee, J., and Hutter, M. (2018). Per-contact iteration method for solving contact dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902.

[Immersive Limit, 2025] Immersive Limit (last accessed on 19/DEC/2025). Immersive limit. *https://www.immersivelimit.com/blog/training-an-ai-to-recognize-cigarette-butts*.

[Jenelten et al., 2022] Jenelten, F., Grandia, R., Farshidian, F., and Hutter, M. (2022). Tamols: Terrain-aware motion optimization for legged systems. *IEEE Transactions on Robotics*, 38(6):3395–3413.

[Jenelten et al., 2024] Jenelten, F., He, J., Farshidian, F., and Hutter, M. (2024). Dtc: Deep tracking control. *Science Robotics*, 9(86):eadh5401.

[Jeon et al., 2025] Jeon, S. H., Hong, S., Lee, H. J., Khazoom, C., and Kim, S. (2025). Cusadi: A gpu parallelization framework for symbolic expressions and optimal control. *IEEE Robotics and Automation Letters*, 10(2):899–906.

[Jordana et al., 2023] Jordana, A., Kleff, S., Meduri, A., Carpentier, J., Mansard, N., and Righetti, L. (2023). Stagewise implementations of sequential quadratic programming for model predictive control.

[Jorge and Stephen, 1999] Jorge, N. and Stephen, J. W. (1999). *Numerical Optimization*. Springer New York, NY.

[Kamidi et al., 2022] Kamidi, V. R., Kim, J., Fawcett, R. T., Ames, A. D., and Akbari Hamed, K. (2022). Distributed quadratic programming-based nonlinear controllers for periodic gaits on legged robots. *IEEE Control Systems Letters*, 6:2509–2514.

[Khazoom et al., 2024] Khazoom, C., Hong, S., Chignoli, M., Stanger-Jones, E., and Kim, S. (2024). Tailoring solution accuracy for fast whole-body model predictive control of legged robots. *IEEE Robotics and Automation Letters*, 9(12):11074–11081.

[Kiessling et al., 2024] Kiessling, D., Baumgärtner, K., Frey, J., Decré, W., Swevers, J., and Diehl, M. (2024). Fast generation of feasible trajectories in direct optimal control. *IEEE Control Systems Letters*, 8:1168–1173.

[Kim et al., 2023] Kim, J., Fawcett, R. T., Kamidi, V. R., Ames, A. D., and Hamed, K. A. (2023). Layered control for cooperative locomotion of two quadrupedal robots: Centralized and distributed approaches. *IEEE Transactions on Robotics*, 39(6):4728–4748.

[Koenemann et al., 2015] Koenemann, J., Del Prete, A., Tassa, Y., Todorov, E., Stasse, O., Bennewitz, M., and Mansard, N. (2015). Whole-body model-predictive control applied to the hrp-2 humanoid. In *2015 IEEE/RSJ IROS*, pages 3346–3351.

[Lee et al., 2020] Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5.

[Lee et al., 2022] Lee, Y., Cho, M., and Kim, K.-S. (2022). Gpu-parallelized iterative lqr with input constraints for fast collision avoidance of autonomous vehicles. In *IEEE/RSJ IROS*, pages 4797–4804.

[Lee et al., 2024] Lee, Y., Choi, K. H., and Kim, K.-S. (2024). Gpu-enabled parallel trajectory optimization framework for safe motion planning of autonomous vehicles. *IEEE Robotics and Automation Letters*, 9(11):10407–10414.

[Lembono et al., 2020] Lembono, T. S., Mastalli, C., Fernbach, P., Mansard, N., and Calinon, S. (2020). Learning how to walk: Warm-starting optimal control solver with memory of motion. In *2020 IEEE ICRA*, pages 1357–1363.

[L'Erario et al., 2022] L'Erario, G., Nava, G., Romualdi, G., Bergonti, F., Razza, V., Dafarra, S., and Pucci, D. (2022). Whole-body trajectory optimization for robot multimodal locomotion. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 651–658.

[Li and Wensing, 2024] Li, H. and Wensing, P. M. (2024). Cafe-mpc: A cascaded-fidelity model predictive control framework with tuning-free whole-body control. *IEEE Transactions on Robotics*, pages 1–20.

[Li et al., 2023] Li, Y., Chen, Z., Wu, C., Mao, H., and Sun, P. (2023). A hierarchical framework for quadruped robots gait planning based on ddpg. *Biomimetics*, 8(5):382.

[Liu et al., 2021] Liu, J., Balatti, P., Ellis, K., Hadjivelichkov, D., Stoyanov, D., Ajoudani, A., and Kanoulas, D. (2021). Garbage collection and sorting with a mobile manipulator using deep learning and whole-body control. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 408–414.

[Ma et al., 2021a] Ma, W.-L., Csomay-Shanklin, N., and Ames, A. D. (2021a). Coupled control systems: Periodic orbit generation with application to quadrupedal locomotion. *IEEE Control Systems Letters*, 5(3):935–940.

[Ma et al., 2021b] Ma, W.-L., Csomay-Shanklin, N., Kolathaya, S., Hamed, K. A., and Ames, A. D. (2021b). Coupled control lyapunov functions for interconnected systems, with application to quadrupedal locomotion. *IEEE Robotics and Automation Letters*, 6(2):3761–3768.

[Martinez et al., 2024] Martinez, S., Griffin, R., and Mastalli, C. (2024). Multi-contact inertial estimation and localization in legged robots. In *IEEE/RSJ IROS*.

[Mastalli et al., 2020] Mastalli, C., Budhiraja, R., Merkt, W., Saurel, G., Hammoud, B., Naveau, M., Carpentier, J., Righetti, L., Vijayakumar, S., and Mansard, N. (2020). Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Mastalli et al., 2023] Mastalli, C., Chhatoi, S., Corbéres, T., Tonneau, S., and Vijayakumar, S. (2023). Inverse-dynamics mpc via nullspace resolution. *IEEE Transactions on Robotics*, 39(4):3222–3241.

[Mastalli et al., 2022] Mastalli, C., Merkt, W., Xin, G., Shim, J., Mistry, M., Havoutis, I., and Vijayakimar, S. (2022). Agile maneuvers in legged robots: A predictive control approach. *IEEE Transactions on Robotics*.

[McGhee and Frank, 1968] McGhee, R. and Frank, A. (1968). On the stability properties of quadruped creeping gaits. *Mathematical Biosciences*, 3:331–351.

[Meduri et al., 2023] Meduri, A., Shah, P., Viereck, J., Khadiv, M., Havoutis, I., and Righetti, L. (2023). Biconmp: A nonlinear model predictive control framework for whole body motion planning. *IEEE Transactions on Robotics*.

[Micevska et al., 2006] Micevska, T., Warne, M. S. J., Pablo, F., and Patra, R. (2006). Variation in, and causes of, toxicity of cigarette butts to a cladoceran and microtox. *Archives of Environmental Contamination and Toxicology*, 50:205–212.

[Miki et al., 2022] Miki, T., Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2022). Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822.

[Mittal et al., 2023] Mittal, M., Yu, C., Yu, Q., Liu, J., Rudin, N., Hoeller, D., Yuan, J. L., Singh, R., Guo, Y., Mazhar, H., Mandlekar, A., Babich, B., State, G., Hutter, M., and Garg, A. (2023). Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747.

[Mordatch et al., 2012] Mordatch, I., Todorov, E., and Popović, Z. (2012). Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics*, 31(4):1–8.

[Neunert et al., 2018] Neunert, M., Stäuble, M., Giftthaler, M., Bellicoso, C. D., Carius, J., Gehring, C., Hutter, M., and Buchli, J. (2018). Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465.

[Nganga and Wensing, 2021] Nganga, J. N. and Wensing, P. M. (2021). Accelerating second-order differential dynamic programming for rigid-body systems. *IEEE Robotics and Automation Letters*, 6(4):7659–7666.

[Nisticò et al., 2025] Nisticò, Y., Soares, J. C. V., Amatucci, L., Fink, G., and Semini, C. (2025). Muse: A real-time multi-sensor state estimator for quadruped robots. *IEEE Robotics and Automation Letters*, 10(5):4620–4627.

[Novotny et al., 2009] Novotny, T. E., Lum, K., Smith, E. A., Wang, V., and Barnes, R. L. (2009). Cigarettes butts and the case for an environmental policy on hazardous cigarette waste. *International Journal of Environmental Research and Public Health*, 6:1691 – 1705.

[Ocean Conservancy, 2023] Ocean Conservancy (2023). 2023 cleanup report. Technical report, Ocean Conservancy. https://oceanconservancy.org/trash-free-seas/international-coastal-cleanup/annual-data-release/.

[Omar et al., 2023] Omar, S., Amatucci, L., Barasuol, V., Turrisi, G., and Semini, C. (2023). Safesteps: Learning safer footstep planning policies for legged robots via model-based priors. In *2023 IEEE-RAS Humanoids*, pages 1–8.

[Orin et al., 2013] Orin, D. E., Goswami, A., and Lee, S.-H. (2013). Centroidal dynamics of a humanoid robot. *Autonomous robots*, 35(2-3):161–176.

[Pandala et al., 2022] Pandala, A., Fawcett, R. T., Rosolia, U., Ames, A. D., and Hamed, K. A. (2022). Robust predictive control for quadrupedal locomotion: Learning to close the gap between reduced- and full-order models. *IEEE Robotics and Automation Letters*, 7(3):6622–6629.

[Pandala et al., 2019] Pandala, A. G., Ding, Y., and Park, H.-W. (2019). qpswift: A real-time sparse quadratic program solver for robotic applications. *IEEE Robotics and Automation Letters*, 4(4):3355–3362.

[Pixies Urban Lab, 2025] Pixies Urban Lab (last accessed on 219/DEC/2025). Pixies. *https://www.pixiesurbanlab.com/*.

[Plancher and Kuindersma, 2020] Plancher, B. and Kuindersma, S. (2020). A performance analysis of parallel differential dynamic programming on a gpu. In Morales, M., Tapia, L., Sánchez-Ante, G., and Hutchinson, S., editors, *Algorithmic Foundations of Robotics XIII*, pages 656–672, Cham. Springer International Publishing.

[Ponton et al., 2021] Ponton, B., Khadiv, M., Meduri, A., and Righetti, L. (2021). Efficient multicontact pattern generation with sequential convex approximations of the centroidal dynamics. *IEEE Transactions on Robotics*, 37(5):1661–1679.

[Proença and Simões, 2020] Proença, P. F. and Simões, P. (2020). Taco: Trash annotations in context for litter detection. *arXiv*.

[Project Beach Bot, 2025] Project Beach Bot (last accessed on 19/DEC/2025). Beachbot. *https://project.bb/en*.

[Rangel-Buitrago et al., 2022] Rangel-Buitrago, N., Williams, A. T., Neal, W. J., Gracia C, A., and Micallef, A. (2022). Litter in coastal and marine environments. *Marine Pollution Bulletin*, 177:113546.

[Rathod et al., 2021] Rathod, N., Bratta, A., Focchi, M., Zanon, M., Villarreal, O., Semini, C., and Bemporad, A. (2021). Model predictive control with environment adaptation for legged locomotion. *IEEE Access*, 9:145710–145727.

[Rawlings et al., 2022] Rawlings, J. B., Mayne, D. Q., and Diehl, M. M. (2022). *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Pub.

[Risiglione et al., 2022] Risiglione, M., Barasuol, V., Caldwell, D. G., and Semini, C. (2022). A whole-body controller based on a simplified template for rendering impedances in quadruped manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Romualdi et al., 2022] Romualdi, G., Dafarra, S., L'Erario, G., Sorrentino, I., Traversaro, S., and Pucci, D. (2022). Online non-linear centroidal mpc for humanoid robot locomotion with step adjustment. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10412–10419.

[Ross et al., 2011] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *2021 AISTATS*, pages 627–635. JMLR Workshop and Conference Proceedings.

[Shorten and Khoshgoftaar, 2019] Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48.

[Slaughter et al., 2010] Slaughter, E., Gersberg, R. M., Watanabe, K., Rudolph, J., Stransky, C., and Novotny, T. E. (2010). Toxicity of cigarette butts and their chemical components to the marine and freshwater fishes, atherinops affinis and pimephales promelas. *Tobacco Control*.

[Sleiman et al., 2021] Sleiman, J.-P., Farshidian, F., Minniti, M. V., and Hutter, M. (2021). A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters*, 6(3):4688–4695.

[Song et al., 2015] Song, Q., Li, J., and Zeng, X. (2015). Minimizing the increasing solid waste through zero waste strategy. *Journal of Cleaner Production*, 104:199–210.

[Särkkä and García-Fernández, 2023] Särkkä, S. and García-Fernández, A. F. (2023). Temporal parallelization of dynamic programming and linear quadratic control. *IEEE Transactions on Automatic Control*, 68(2):851–866.

[Taouil et al., 2025] Taouil, I., Amatucci, L., Khadiv, M., Dai, A., Barasuol, V., Turrisi, G., and Semini, C. (2025). Non-gaited legged locomotion with monte-carlo tree search and supervised learning. *IEEE Robotics and Automation Letters*, 10(2):1265–1272.

[Tassa et al., 2012] Tassa, Y., Erez, T., and Todorov, E. (2012). Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ IROS*, pages 4906–4913.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033.

[Todorov and Li, 2005] Todorov, E. and Li, W. (2005). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference, 2005.*, pages 300–306 vol. 1.

[Tonneau et al., 2020] Tonneau, S., Song, D., Fernbach, P., Mansard, N., Taïx, M., and Del Prete, A. (2020). Sl1m: Sparse l1-norm minimization for contact planning on uneven terrain. In *2020 IEEE ICRA*, pages 6604–6610.

[Tsounis et al., 2020] Tsounis, V., Alge, M., Lee, J., Farshidian, F., and Hutter, M. (2020). Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):3699–3706.

[Turrisi et al., 2024] Turrisi, G., Modugno, V., Amatucci, L., Kanoulas, D., and Semini, C. (2024). On the benefits of gpu sample-based stochastic predictive controllers for legged locomotion. In *2024 IEEE/RSJ IROS*.

[UNEP, 2006] UNEP (2006). 2006 annual report. Technical report, United Nations Environment Programme.

[unitree, 2025] unitree (last accessed on 19/DEC/2025). Aliengo. https://www.unitree.com/.

[Vanroye et al., 2023] Vanroye, L., Sathya, A., De Schutter, J., and Decré, W. (2023). Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10036–10043.

[Verschueren et al., 2021] Verschueren, R., Frison, G., Kouzoupis, D., Frey, J., van Duijkeren, N., Zanelli, A., Novoselnik, B., Albin, T., Quirynen, R., and Diehl, M. (2021). acados – a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*.

[Villarreal Magaña et al., 2019] Villarreal Magaña, O. A., Barasuol, V., Camurri, M., Franceschi, L., Focchi, M., Pontil, M., Caldwell, D. G., and Semini, C. (2019). Fast and continuous foothold adaptation for dynamic locomotion through cnns. *IEEE Robotics and Automation Letters*, 4(2):2140–2147.

[Vincenti and Coros, 2023] Vincenti, F. D. and Coros, S. (2023). Centralized model predictive control for collaborative loco-manipulation. In *Robotics Science and Systems*.

[Wang et al., 2024] Wang, J., Kim, S., Lembono, T. S., Du, W., Shim, J., Samadi, S., Wang, K., Ivan, V., Calinon, S., Vijayakumar, S., and Tonneau, S. (2024). Online multi-contact receding horizon planning via value function approximation. *IEEE Transactions on Robotics*, 40:2791–2810.

[Winkler et al., 2018] Winkler, A. W., Bellicoso, C. D., Hutter, M., and Buchli, J. (2018). Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567.

[World Health Organization, 2022] World Health Organization (2022). Tobacco: poisoning our planet. *Environmental Research*.

[Wright, 1991] Wright, S. J. (1991). Partitioned dynamic programming for optimal control. *SIAM Journal on Optimization*, 1(4):620–642.

[Wu and Sreenath, 2015] Wu, G. and Sreenath, K. (2015). Variation-based linearization of nonlinear systems evolving on $so(3)$ and $\sim^2$. *IEEE Access*, 3:1592–1604.

[Wächter and Biegler, 2006] Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106.

[Yoon et al., 2024] Yoon, Z., Kim, J.-H., and Park, H.-W. (2024). Invariant smoother for legged robot state estimation with dynamic contact event information. *IEEE Transactions on Robotics*, 40:193–212.

[Zapata-Impata et al., 2018] Zapata-Impata, B. S., Shah, V., Singh, H., and Platt, R. W. (2018). Autotrans: an autonomous open world transportation system. *ArXiv*.

[Zhou and Zhao, 2020] Zhou, Z. and Zhao, Y. (2020). Accelerated admm based trajectory optimization for legged locomotion with coupled rigid body dynamics. In *2020 American Control Conference (ACC)*, pages 5082–5089.

[Zhu et al., 2023] Zhu, H., Meduri, A., and Righetti, L. (2023). Efficient object manipulation planning with monte carlo tree search. In *2023 IEEE/RSJ IROS*, pages 10628–10635.

Appendix A

VERO: A vacuum-cleaner-equipped quadruped robot for efficient litter removal

L. Amatucci, G. Turrisi, A. Bratta, V. Barasuol, and C. Semini, "VERO: A vacuum-cleaner-equipped quadruped robot for efficient litter removal" Journal of Field Robotics, vol. 41, no. 6, pp. 1829-1842, 2024, doi: https://doi.org/10.1002/rob.22350.

Abstract

Litter nowadays presents a significant threat to the equilibrium of many ecosystems. An example is the sea, where litter coming from coasts and cities via gutters, streets, and waterways, releases toxic chemicals and microplastics during its decomposition. Litter removal is often carried out manually by humans, which inherently lowers the amount of waste that can be effectively collected from the environment. In this work, we present a novel quadruped robot prototype that, thanks to its natural mobility, is able to collect cigarette butts autonomously, the second most common undisposed waste worldwide, in terrains that are hard to reach for wheeled and tracked robots. The core of our approach is a convolutional neural network for litter detection, followed by a time-optimal planner for reducing the time needed to collect all the target objects. Precise litter removal is then performed by a visual servoing procedure which drives the nozzle of a vacuum cleaner that is attached to one of the robot legs on top of the detected cigarette butt. As a result of

Figure A.1 A snapshot of our quadrupedal robot prototype walking at the beach of Vernazzola, Genova, Italy.

this particular position of the nozzle, we are able to perform the collection task without even stopping the robot's motion, thus greatly increasing the time-efficiency of the entire procedure. Extensive tests were conducted in six different outdoor scenarios to show the performance of our prototype and method. To the best knowledge of the authors, this is the first time that such a design and method was presented and successfully tested on a legged robot.

A.1 Introduction

Litter is an economic, biological and health-related problem [UNEP, 2006], and along with climate change, loss of biodiversity, ocean acidification, and overpopulation, it is widely recognized as one of the five main challenges that we are facing as humanity nowadays [Rangel-Buitrago et al., 2022]. The increase in urbanization and consumerism

are driving factors in the rise of trash worldwide [Song et al., 2015], and a considerable percentage of the produced waste is not properly disposed of every year. This behavior contaminates soil and water, increasing the risk for our planet and its inhabitants. Litter can easily reach marine environments by ending up in gutters, streets, and waterways [Novotny et al., 2009]. When it breaks down, it can release toxic chemicals, metals, and microplastic, causing changes in the fauna gene and protein expression, inflammation, and disruption of feeding behavior [Bhuyan, 2022], finally altering the equilibrium of our ecosystem.

Among all the different types of waste, Cigarette Butts (CBs) or cigarette filters are the second most common undisposed waste worldwide [Rangel-Buitrago et al., 2022]. More than 5 trillion cigarettes were consumed in 2016, two-thirds of which were supposedly not thrown away correctly [Araújo and Costa, 2019], causing, therefore, a vast environmental problem. In fact, CBs are usually composed of a cellulose-acetate-based material which does not biodegrade and can remain in the environment for more than ten years [Belzagui et al., 2020], freeing up in the marine environments microplastic and more than 700 toxic chemicals [Slaughter et al., 2010], [Micevska et al., 2006], e.g., nicotine, 70 of which are known to cause cancer in humans and animals [Belzagui et al., 2020].

Given the dreadful impact on the environment caused by the incorrect disposal of CBs [World Health Organization, 2022], many awareness campaigns have been promoted by governments over the years to mitigate the problem. Nevertheless, a waste collection procedure performed directly in-loco, such as in cities, beaches, and other coastal areas, is still necessary, as demonstrated in the *Ocean Conservation* Cleanup reports [Ocean Conservancy, 2023]. Unfortunately, the collection procedure faces many challenges that make an effective scale-up impossible. While CBs on streets can be efficiently removed with dedicated vehicles, the majority of other areas require labor-intensive, manual collection. This is mainly due to the difficulties given by the CBs' different sizes, shapes, and materials, as well as the uneven terrain present in the different environments they are dispersed into.

Many projects have been developed, in both industry and academia, trying to automate the aforementioned procedure. Most of these works have been designed with a specific target environment in mind. For example, the robots Angsa [Angsa Robotics, 2025] and Pixies [Pixies Urban Lab, 2025] are small-sized wheeled systems that aim for a more conventional scenario, such as grass and urban environments, whereas BeachBot [Project Beach Bot, 2025] and Bebot [4Ocean, 2025] have been developed to be operated on sandy beaches. For this reason, they utilized tracks and specially designed wheels to move. The developed machines also differ in the approach chosen for the collection. Works such as [Zapata-Impata et al., 2018, Bai et al., 2018, Liu et al., 2021] utilize

an arm to selectively collect litter; while this method allows for the precise selection of the object to collect, it requires a more sophisticated vision and control pipeline that slows down considerably the collection speed. On the other hand, other projects like [4Ocean, 2025] opt for a more conventional mechanism that collects everything on the ground without the possibility of differentiating between trash, plants, or animals. Even though all these approaches have brought progress towards autonomous litter removal, their traversability capability is ultimately limited by their design. In fact, in the case of small obstacles on their path or uneven terrain, which is a common situation in coasts and cities, wheeled robots have clear limitations that reduce the amount of litter that can be collected.

Recently, quadrupedal robots have shown astonishing results in traversing challenging terrain. A learning-based controller able to cope with unknown disturbances and uneven terrains was proposed in [Lee et al., 2020]. Their outdoor experiments clearly show the great stability and versatility achievable by a quadruped robot. Furthermore, safety can be explicitly considered during locomotion by adopting vision-based correction for precise footstep planning to avoid having a foot stuck [Villarreal Magaña et al., 2019], [Jenelten et al., 2022]; in this way, a long-term autonomous operation of these systems can be achieved. The achievable safety and the great agility that distinguishes quadruped robots from wheeled robots show great potential in adopting these systems for litter removal, especially in human-designed, uneven, and complex scenarios. Yet, to the best of the author's knowledge, no legged robot prototypes have been developed specifically for this application. We present the autonomous litter removal prototype *VERO* that builds upon a state-of-the-art control framework for legged locomotion. Based on the quadrupedal robot Aliengo of Unitree[1], our design is able to autonomously collect undisposed CBs while traversing a variety of complex scenarios that are mostly inaccessible to wheeled robots. We equipped our prototype with a vacuum cleaner attached to its trunk and a custom-made nozzle connected to one of its legs, enabling the possibility of performing collection just by stepping with the attached nozzle over the detected CBs (Fig. A.1). From a software point of view, we obtain a precise collection via a multilevel framework that comprehends image recognition via neural networks, planning for time-optimal collection paths, and visual servoing for precise positioning of the robot's foot.

[1] https://www.unitree.com/products/aliengo/

A.1.1 Contribution

The main contributions of this work are:

- the design and integration of a novel platform for litter collection based on a legged robot, a vacuum cleaner, and an innovative foot nozzle design;
- the development of a perception, estimation and control pipeline that identifies the litter, filters out duplicated detections, and plans the optimal collection path;
- an extensive real-world validation of the overall system performance, with numerical and qualitative analysis of the platform advantages and limitations, in scenarios where CBs are commonly dispersed into.
- finally, we want to point out that this work represents, to the best of the author's knowledge, the first instance of utilizing a legged quadruped robot where its legs performs an additional task *during* locomotion, eliminating the need for extra limbs to accomplish the designated function.

A.1.2 Outline

The work is organized as follows: Sec. A.2 gives an overview of our litter collection prototype, whereas Sec. A.3 describes the proposed approach delving into the developed CBs detection module, along with a state estimation and planning procedure for precise and time-optimized waste collection. In Sec. A.4, we detail our locomotion controller, while in Section A.5 we report on the application of the proposed prototype, showing both indoor and outdoor experimental results. Finally, a discussion and some general conclusions about the approach are drawn respectively in Sec. A.6 and Sec. A.7.

A.2 Vacuum Cleaner hardware

For this work, the commercial quadruped robot Aliengo has been adapted by adding a 2 kg commercial vacuum cleaner on top of its trunk, while a custom-designed nozzle was attached to its left-front foot (Fig. A.2). As will be explained later, our design can be easily modified to connect the vacuum cleaner to two or all the available feet, increasing the overall system efficiency, described here as the time needed to perform the collection procedure and adaptability. We designed and 3D-printed a custom-made nozzle to maximize the area around the foot of the robot where the CBs can be collected without sacrificing

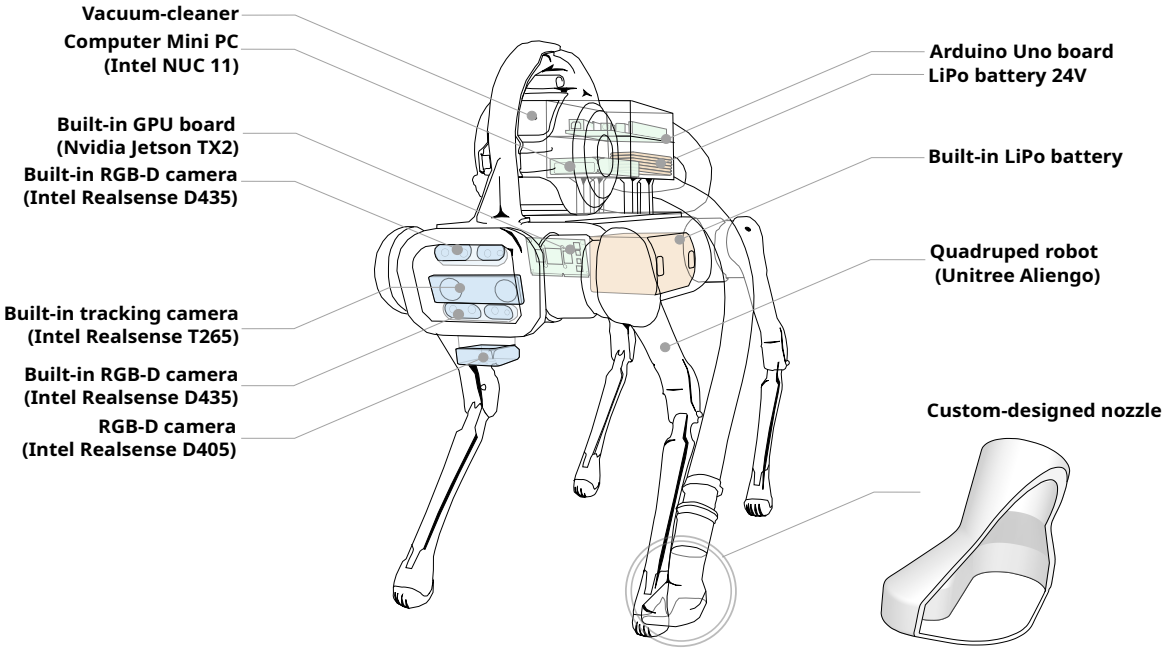

Figure A.2 Prototype hardware: Image of the presented autonomous litter removal prototype with labels for the key components. The system comprises a quadruped robot with built-in sensors and computation, a vacuum cleaner with custom-designed nozzle, additional sensors, computers, electronic boards, and a LiPo battery.

the suction ability of the vacuum cleaner and the locomotion capability of the robot. In fact, a wider suction area would have required either a more powerful vacuum cleaner or a smaller gap between the nozzle and the terrain. The last point has been observed to be the most critical in our experiments since the need to have the nozzle close to the terrain increased the collisions with the ground. All the impacts that do not happen with the foot disturb the stability of the motion and the accuracy of the stepping. In Fig. A.2, the reader can clearly observe the adopted solution. Even if our framework enables precise footstep placement on top of the detected CBs, a few centimeter mismatches can result in a non-successful collection procedure. With a bigger nozzle area, this tolerance is increased.

The vacuum cleaner mounted on top of the platform is a modified commercial machine. We changed the power supply to allow external control with a microcontroller board and a relay, as shown in Fig. A.2. In particular, an Arduino UNO board is connected to an onboard MiniPC and is powered with 5V. A 24V Lipo battery and a DC-DC converter 24V-12V power the relay. Commanded by the microcontroller, the relay is only closed when the vacuum cleaner must be turned on in order to optimize battery life and avoid collecting undesired objects during walking.

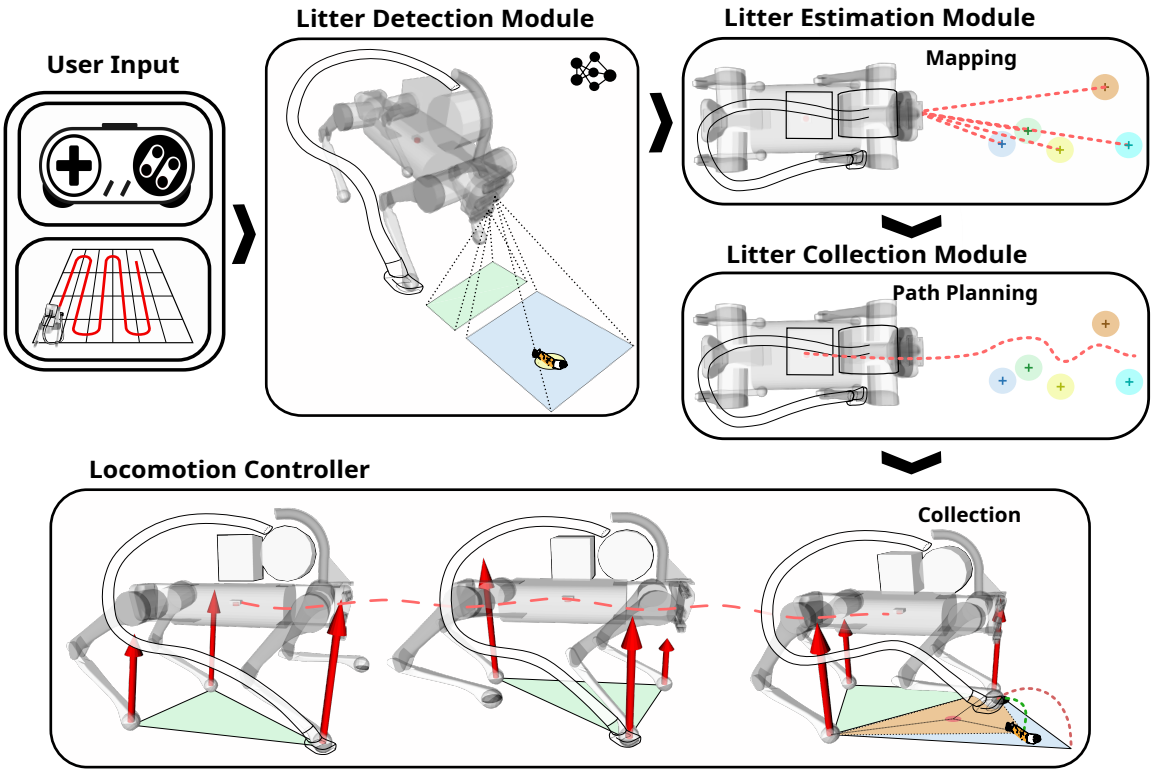

Figure A.3 Block scheme of the proposed collection procedure. Starting from the left, first, an area to clean is selected by an operator via a user interface, or alternatively, the robot is directly commanded using a joystick. The robot starts to detect the CBs while moving and creates a map of the collectable objects, and an optimal path planner is computed after each detection. Finally, the locomotion controller drives the robot to the collection area while a visual servoing procedure precisely commands the robot's footsteps on top of the CBs. In the locomotion controller block, the current support polygon is highlighted in green, while the Ground Reaction Forces are in red. In the last snapshot on the right, the future nominal support polygon is highlighted in blue, while the one defined by the new stepping location is in orange. Finally, the corresponding swings for the two stepping locations are shown, respectively, in red and green.

A.3 The Litter Collection Procedure

In this section, we describe the main components of our collection approach. At its core, there are three main modules, namely: Litter Detection Module (LDM), Litter Estimation Module (LEM), and Litter Collection Module (LCM). All modules run continuously during robot motion.

During the deployment of the quadruped robot, the operator chooses a desired area to clean, and subsequently, the robot calculates an exploration path to fully cover the

space. For this, we divide the chosen area into parallel lanes with a width that is relative to the field of view of the robot's frontal camera. Alternatively, the quadruped can be controlled via dedicated joystick commands by the operator. During locomotion, the LDM is responsible for the CBs detection via the onboard cameras, while the LCM generates a time-optimal collection path between the target objects. The same object can appear in more than one camera frame during locomotion, and in this case, multiple detections of the same item can happen. Therefore, a data association routine, together with a filtering procedure, is performed in the LEM to avoid unnecessary robot motions. Finally, once the quadruped reaches the desired CB, a visual servoing routine that leverages a secondary camera pointing down in the direction of the robot's front feet is activated for precise step placement.

A block diagram of the proposed approach is shown in Fig. A.3.

A.3.1 Litter Detection Module

The LDM is characterized by a CB detection neural network that is used for a first identification of the object to collect utilizing the images from the forward-facing camera. This information is then passed to the LCM for planning an optimal path and for performing a visual servoing procedure for a precise footstep placement using the downward-facing camera. The latter is necessary for coping with inevitable state estimation drifts.

The network utilized in this work is the broadly adopted Yolo-V4-tiny [Bochkovskiy et al., 2020], a Convolutional Neural Network renowned for its computational speed and accuracy. We used the tiny version of the network, which has a reduced number of parameters, to ensure online computation on the onboard PC, an Nvidia Jetson TX2. Yolo builds a bounding box around an object in the image and returns the probability that it contains a CB. To avoid false positives, we consider a CB detection only if its probability is higher than 90%. The network was retrained on the target objects exploiting the combination of two different datasets available online [Proença and Simões, 2020], [Immersive Limit, 2025]. The number of images was increased by a further manual annotation of other one hundred images and by different data-augmentation techniques, such as image rotation, crop, background, and luminance modification [Shorten and Khoshgoftaar, 2019] generating two thousand more images. The network achieves a final overall accuracy of 91% on the validation data.

Starting from the LDM, the outputs of the detection procedure are bounding boxes surrounding the CBs present in the video feed. The network is employed two times in our

collection procedure. First, it is fed with 720p RGB images coming from an Intel RealSense D435 mounted on the front of the robot (forward-facing camera). The CBs' detections in this case are used for planning purposes. Second, it is queried by feeding the images from an Intel RealSense D405, mounted on the bottom of the robot's trunk (downward-facing camera). Both cameras configuration is shown in Fig. A.2. This second pipeline is used for the visual-servoing procedure to drive precisely the foot of the robot (see Sec. A.3.3), therefore, considering the camera positioning, we opt for the D405 given its shorter-range detection capability. In fact, putting the camera above the trunk (Fig. A.2) could produce more occlusions in the detection when the CB is located directly below the robot, and given the normal walking height of the system (around 30cm), a camera with a short-range detection capability is needed. On the other hand, the D435 is used for a first detection of the CBs from a distance. These cameras also provide a depth map, aligned with the color image, that gives information on the distance of each pixel. This data is used to calculate the relative position of the detected CB with respect to the camera frame as

$$^{\mathcal{C}}x_{CB} = \frac{x_{bb} - c_x}{F_x} \qquad ^{\mathcal{C}}y_{CB} = \frac{y_{bb} - c_y}{F_y} \tag{A.1}$$

$$^{\mathcal{C}}z_{CB} = depth(x_{bb}, y_{bb}) \tag{A.2}$$

where $^{\mathcal{C}}x_{CB}$, $^{\mathcal{C}}y_{CB}$, and $^{\mathcal{C}}z_{CB}$ are the coordinates of the selected pixel expressed in the camera frame $\mathcal{C}$ and x_{bb} and y_{bb} are the pixel coordinates of the center of the bounding box. Furthermore, F_x, F_y, c_x, and c_y are the intrinsic parameters of the camera, while $depth(\cdot, \cdot)$ is the depth map information of the desired coordinates. The litter position in the camera frame is then converted into the world frame to perform the planning procedure in the LCM.

Remark: the superscript at the left of a variable indicates in which frame it is expressed ($\mathcal{C}$ camera frame, $\mathcal{H}$ horizontal frame, $\mathcal{B}$ base frame). If the superscript is omitted, the quantity is expressed in the world frame.

A.3.2 Litter Estimation Module

During the robot motion, the detected CBs are used to generate a map of waypoints, used by the LCM to compute a time-optimal path for the collection process. To avoid unnecessary motions in the presence of duplicate detection, a correct data-association routine is needed. For this, we implemented a Right-Invariant Extended Kalman Filter (Right-IEKF) to reduce the amount of noise and errors due to the state estimation drift. As

demonstrated in [Hartley et al., 2019], this filter has shown promising performance thanks to the log-linear error dynamics.

Given the robot state and the CB positions in the world frame, the filter updates these values each time a new CB is detected. At a generic time instant t, we express the Right-IEKF state representation as

$$\boldsymbol{X}_t = \begin{bmatrix} \boldsymbol{R}^{\theta}_{b,t} & \boldsymbol{v}_{c,t} & \boldsymbol{p}_{c,t} & \boldsymbol{p}_{CB_1,t} & \cdots & \boldsymbol{p}_{CB_N,t} \\ \mathbf{0}_{1,3} & 1 & 0 & 0 & \vdots & 0 \\ \mathbf{0}_{1,3} & 0 & 1 & 0 & \vdots & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 1 & \vdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_{1,3} & 0 & 0 & 0 & \vdots & 1 \end{bmatrix} \tag{A.3}$$

where $\boldsymbol{X}_t \in SE_{N+2}(3)$ is an extension of the $SE(3)$ Lie Group (see [Hartley et al., 2019]) for the case of N different objects saved in the map; $\boldsymbol{R}^{\theta}_b$, $\boldsymbol{v}_{c,t}$, $\boldsymbol{p}_{c,t}$ are respectively, the estimated yaw of the robot's base, the velocity, and the position of the robot's Center of Mass (CoM); finally, $\boldsymbol{p}_{CB_i,t}$ is the world position of the i^{th} CB.

The described filter fuses the drifting state, coming from the onboard odometry-based state estimator, with the measured position of the collectable CB, used as a fixed landmark. The right invariant observation model for the i^{th} CB measurement is written as

$$\begin{aligned} \boldsymbol{h}_i &= \boldsymbol{R}_c \begin{bmatrix} {}^{\mathcal{C}}x_{CB,i} & {}^{\mathcal{C}}y_{CB,i} & {}^{\mathcal{C}}z_{CB,i} \end{bmatrix}^T \\ \boldsymbol{H}_t &= \begin{bmatrix} \boldsymbol{h}_i & 0 & 1 & \mathbf{0}_{N-i,1} & -1 & \mathbf{0}_{i-1,1} \end{bmatrix}^T \\ \boldsymbol{s} &= \begin{bmatrix} \mathbf{0}_{3,1} & 0 & 1 & \mathbf{0}_{N-i,1} & -1 & \mathbf{0}_{i-1,1} \end{bmatrix}^T \\ \boldsymbol{Y}_t &= \boldsymbol{X}_t^{-1} \boldsymbol{s} \end{aligned} \tag{A.4}$$

where ${}^{\mathcal{C}}x_{CB,t}$, ${}^{\mathcal{C}}y_{CB,t}$, ${}^{\mathcal{C}}z_{CB,t}$ are the measurement at time t from Eq. (A.1), and $\boldsymbol{R}_c$ the rotation matrix from the camera frame to world frame. The IEKF update is then written as

$$\begin{aligned} \boldsymbol{S} &= \boldsymbol{H}\boldsymbol{P}_t\boldsymbol{H}^T + \boldsymbol{N}_t \\ \boldsymbol{K} &= \boldsymbol{P}_t\boldsymbol{H}^T\boldsymbol{S}^{-1} \\ \boldsymbol{Z} &= \boldsymbol{X}_t\boldsymbol{Y}_t - \boldsymbol{s} \\ \widetilde{\boldsymbol{X}}_t &= exp(\boldsymbol{K}\boldsymbol{\Pi}\boldsymbol{Z})\boldsymbol{X}_t \\ \widetilde{\boldsymbol{P}}_t &= (\mathbb{I} - \boldsymbol{K}\boldsymbol{H})\boldsymbol{P}_t(\mathbb{I} - \boldsymbol{K}\boldsymbol{H})^T + \boldsymbol{K}\boldsymbol{N}_t\boldsymbol{K}^T \end{aligned} \tag{A.5}$$

where $\boldsymbol{\Pi}$ and $\boldsymbol{H}$ are selection matrices, $\boldsymbol{K}$ is the Kalman gain and $\boldsymbol{N}_t$ is the measurement covariance matrix. $\widetilde{\boldsymbol{X}}_t$ and $\widetilde{\boldsymbol{P}}_t$ are respectively the updated state and covariance matrix and $exp(\cdot)$ is the exponential map of the state Lie Group.

In order to properly exploit the filter, data association between the mapped objects and the new incoming measurement at each iteration is crucial. We periodically verify if the same CBs have been added twice to the map due to outliers in the measurement.

For this, we decided to utilize a threshold on the Mahalanobis distance between each CB, paired with a minimum distance threshold for checking the possibility of unresolved duplicate detection. The selection of this threshold is crucial when dealing with densely distributed CBs. If the threshold is chosen too conservatively, it may lead to the inadvertent exclusion of CBs that are located close to each other. In our experimental results (see Sec. A.5), we opt for a minimum threshold of 2cm.

To avoid the filter becoming too computationally expensive, we erase all the previously collected objects from the filter state.

A.3.3 Litter Collection Module

We mounted a custom-designed suction nozzle to the left-front leg of our robot prototype. Therefore, planning a path is needed to allow the quadruped to place its foot/feet close to the side of each detected CB. One of the main challenges of this approach is that we still need to guarantee the stability of the robot when changing the footstep location. It is important to recall that the robot needs to constantly actively balance on its legs, and the foot placement represents one of the variables for the feasibility of the planned motion. To account for this problem, we decided to decouple planning and safety in a hierarchical manner. We split the LCM into two submodules, one that uses the data from the LEM to plan a first path for the CoM of the robot, and a second module that directly modifies the footstep placement to ensure the precision and safety of the collection procedure.

For the planning problem, given that multiple CBs can be detected during motion, and hence multiple collection sequences can be performed, we apply the Dijkstra search algorithm [Dijkstra, 1959], which returns both the optimal collection order and the optimal poses that the robot should have when placing the nozzle on top of the CBs. The computational time will depend on the complexity of the scenario, i.e., the number of CBs on the ground. Here we only consider the X-Y positions and yaw angle of the robot's base, leaving to the locomotion controller the optimization of the robot's roll and pitch base angles (see Sec. A.4.1). Additionally, we simplify the problem by disregarding the robot's

height. We assume that the CBs are located in regions accessible by the robot directly from any other position. In other words, we consider a scenario wherein the robot has a perfect traversability, i.e. the CBs can be reached by our system starting from *any* starting position. In reality, this assumption could not hold, and can lead to unfeasibility of the collection order. We leave the lifting of this assumption for future works.

Figure A.4 Snapshots and illustrations detailing the collection procedure. From left to right: the system first detects the cigarette butt (CB) using the front-facing camera, and afterward, the planner defines a new optimal pose for collection (highlighted in the first snapshot by the coordinate frame centered on the hip of the robot's leg associated with the collection mechanism) deviating from the predefined path shown in red. Once the robot reaches the collection area, it utilizes the down-facing camera to refine the previous estimate of the CB's location. Finally, the robot proceeds to perform the collection by stepping on the target and activating the suction.

Starting from an initial robot configuration, expressed as x_c^0, y_c^0 and θ_c^0, defining respectively the X-Y positions and yaw orientation of the CoM of the robot, Dijkstra expands all the detected objects sequentially. For each of them, we compute the desired hip configuration that the robot should have to perform the collection. Defining for clarity

$$e_x = x_{CB} - x_c^0 \quad e_y = y_{CB} - y_c^0 \quad e_\theta = \theta_{CB} - \theta_c^0 \tag{A.6}$$

for a generic $\boldsymbol{p}_{CB,k}$, the optimal hip configuration of the robot can be derived in closed form by solving the following problem

$$\begin{aligned} \min_{\boldsymbol{y}} \quad & \frac{1}{2}\boldsymbol{y}^T\boldsymbol{Q}\boldsymbol{y} \\ \text{s.t.} \quad & \boldsymbol{R}_{2\times 2}{}^{\mathcal{B}}\bar{\boldsymbol{p}}_{bh,i} + \begin{bmatrix} x_{CB} & y_{CB} \end{bmatrix}^T = \boldsymbol{T}\boldsymbol{p}_{CB,k} \\ & \zeta = \frac{(e_y)\cos(e_\theta) - (e_x)\sin(e_\theta)}{(e_y)\sin(e_\theta) + (e_x)\cos(e_\theta)} \\ & \boldsymbol{y} = \begin{bmatrix} e_x & e_y & e_\theta & \zeta \end{bmatrix} \end{aligned} \tag{A.7}$$

where $\bar{\boldsymbol{p}}_{bh,i}$ represents the X-Y components of the vector connecting the base to the hip of the i^{th} leg used for collection, while $\boldsymbol{T} = \begin{bmatrix} \mathbb{I}_2 & \boldsymbol{0}_{2,1} \end{bmatrix} \in \mathbb{R}^{3\times 2}$ extracts the X-Y coordinates of $\boldsymbol{p}_{CB,k}$. The matrix $\boldsymbol{Q}$ is a diagonal weight matrix defining the importance of each cost term. $\boldsymbol{R}_{2\times 2} \in \mathbb{R}^{2\times 2}$ maps the X-Y coordinates of the base frame into the world frame. We collect all the expanded CBs in an ordered priority list, from which we can expand the configuration with the minimum cost. This will be used as the new initial robot position for the next Dijkstra step.

Once the quadruped is close enough to one of the CBs, the robot moves to the optimal posture, and the second submodule computes the footstep necessary to perform the collection. This is done by modifying the nominal footholds via a continuous visual servoing procedure, which utilizes the LDM fed by the images coming from the downward-facing camera. The position of the detected CB is then extrapolated by the output of Yolo, converted into the base frame of the robot, and used as the next touch-down position of the robot's foot. More details about the locomotion controller can be found in Sec. A.4.1. Furthermore, this submodule performs a continuous safety check to guarantee the robot's stability. For this, we check if the robot's CoM position projection is still inside the support polygon when a nominal footstep is modified. If the safety check fails, the nominal foothold is restored, preventing the robot from falling.

Fig. A.4 shows some snapshots of the collection procedure performed by the real robotic platform. Furthermore, we summarize the litter collection procedure in Algorithm 2, where the **goTo()** function refers to a simple PD controller that drives the robot to the desired goal commanding a desired speed to the **LocomotionController()** (Sec. A.4). On the other hand, the **UserInput**() function returns a desired speed read from a joystick, or from a speed profile computed to track a predefined path. Finally, the **isSafe()** function corresponds to the stability check described above, while **doCollection()** activates the vacuum cleaner and changes the target step location.

Algorithm 2 Litter collection procedure

```
while robot_moving do
    [CBs]⟵ LDM(front_facing_camera)
    if [CBs] not empty then
        map⟵ LEM([CBs])
    end if
    if map not empty then
        collection_pose⟵ LCM(map)
    end if
    if collection_pose then
        desired_speed⟵ goTo(collection_pose)
    else
        desired_speed⟵ UserInput()
    end if
    LocomotionController(desired_speed) (Sec. A.4)
    if collection_pose == robot_pose then
        CB⟵ LDM(down_facing_camera)
        if isSafe(CB) then
            doCollection()
        end if
    end if
end while

function LDM(camera_image)
    bbox⟵ Yolo(camera_image)
    measurement⟵get3dMeasurement(bbox) (Eq. A.1,A.2)
    return measurement
end function

function LEM(measurement)
    checkDataAssosiation(measurement)
    map⟵IEKF(measurement) (Eq. A.5)
    return map
end function

function LCM(map)
    collection_poses⟵DijkstraSearch(map) (Eq. A.7)
    return collection_poses[0]
end function
```

A.4 Locomotion controller

Our locomotion control framework is composed of proprioceptive and exteroceptive elements. To obtain a robust locomotion controller, we focus on a robust proprioceptive and reactive layer able to deal with irregular surfaces and uncertainties from the exteroceptive feedback (visual information). To do so, we implement a structure based on our Reactive Controller Framework (RCF) [Barasuol et al., 2013], which is composed of *Motion Generation* and *Motion Control* blocks. Both blocks, illustrated in Fig. A.5, are explained in detail in the following sections.

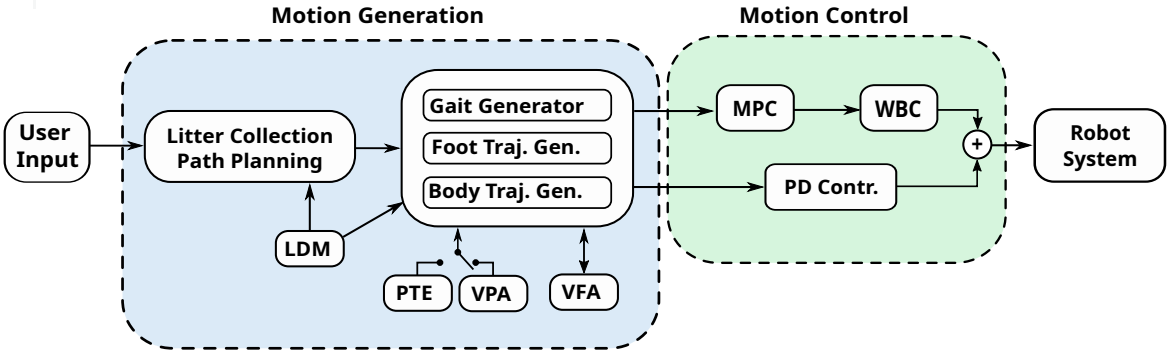

Figure A.5 Block diagram illustrating the dataflow of the motion generation and control blocks of the locomotion framework.

A.4.1 Motion Generation

As depicted in Fig. A.5, the motion generation block computes the desired robot base and leg motions. The RCF's motion generation can provide various gait patterns, like trotting and crawling gaits. For the sake of space, we concentrate only on the description of the crawl gait used for the litter collection.

The crawl gait represents a sequence for the motion of the legs that allows achieving static stability during locomotion. Static stability is achieved when the vertical projection of the robot's CoM position lies inside the convex hull (so-called support polygon) formed by the contact foot locations [McGhee and Frank, 1968]. A robot that executes such a gait operates in slow velocities but very robustly, being able to have long stance periods on various postures and to deal with very irregular surfaces. Two crawling gait sequences are implemented: leg sequence left-front⇒right-front⇒right-hind⇒left-hind when moving forward, and leg sequence left-front⇒left-hind⇒right-front⇒right-hind when moving back-

wards. The different sequences are used to prevent support polygon shrinkage in one of the speed directions, and thus the reduction of the static stability margin and maximum velocity the robot can achieve. The desired touch-down location of each foot is computed in a robot-centric approach [Rathod et al., 2021], and depends on high-level trajectory references (high-level commands provided by the user or by the collection path planner) and also proprioceptive and exteroceptive information. All the leg motion planning is performed in the *Horizontal Frame* (located at the robot's base frame with Z coordinates aligned with the gravity vector and the X coordinate aligned with the robot's base longitudinal axis projected in the world X-Y plane) [Barasuol et al., 2013]. The velocity references are provided without considering the terrain irregularities and are treated as the tentative body motion path for an effective litter collection. From the reference velocities, the nominal foot touch-down position and corresponding leg swing motion are computed. The timing for the desired leg stance and swing phases is provided by the *Gait Generator* (based on desired step frequency and duty factor). It is important to highlight that the periodicity of the crawl gait, in our controller, is stability-dependent. In other words, even with pre-defined step frequency and duty factor for each leg [de Santos et al., 2012], a leg lift-off event is only executed if 1) the robot is under a static stability condition defined by a given stability margin value, and 2) the robot is in full-stance condition (all legs in contact with the ground). These conditions confer extra robustness to the locomotion when compared to approaches that consider a strictly periodic crawling gait (e.g., in situations that may lead to surface collapses, contact losses, leg slippage, or inaccuracies in the terrain map or visual feedback). The desired foot references are provided by the *Foot Trajectory Generator* and are computed based on the actual foot lift-off position, the desired foot touch-down position, and the desired step clearance (the so-called step height). The nominal foothold position of a leg i is defined as $\boldsymbol{p}_{n,i} = [p_{nx,i}\ p_{ny,i}\ p_{nz,i}]^T$. The desired planar coordinates X-Y of the i^{th} nominal foot touch-down position are computed as:

$$\boldsymbol{p}_{n,xy} = \boldsymbol{p}_{0,xy} + {}^{\mathcal{H}}\dot{\boldsymbol{p}}_{c,xy}\frac{T_s}{2} + [{}^{\mathcal{H}}\boldsymbol{\omega} \times \boldsymbol{p}_{h,xy}]|_{x,y} \tag{A.8}$$

where $\boldsymbol{p}_0$ is the foot home position, T_s is the desired stance period, ${}^{\mathcal{H}}\boldsymbol{\omega} = [0\ 0\ {}^{\mathcal{H}}\dot{\phi}_c]^T$ is the desired body angular velocity, and $\boldsymbol{p}_h$ is the hip position. The operator $|_{x,y}$ selects the planar coordinates from the vector product. We removed the dependency from leg i for the sake of clarity in the notation. The corresponding component $p_{n,z}$ depends on the Z component of the lift-off position p_{lo}, the pair $(p_{n,x}, p_{n,y})$, and the average terrain inclination estimated by the *Proprioceptive Terrain Estimator* (PTE). The PTE approximates the

terrain surface with an average plane whose normal is described in the robot horizontal frame and inclination represented by a terrain roll angle ϕ_T and pitch angle θ_T. Thus, $p_{n,z}$ is computed as:

$$p_{n,z} = p_{lo} + [\boldsymbol{R}_h(\phi_T, \theta_T)\boldsymbol{p}_n]|_z \tag{A.9}$$

where $\boldsymbol{R}_h$ is the terrain rotation matrix, with zero yaw angle. The operator $|_z$ selects the scalar value corresponding to the Z coordinate.

During the litter collection, the nominal foothold can be modified by the visual feedback from the Visual Foothold Adaptation (VFA) [Villarreal Magaña et al., 2019], or by the LDM. The VFA modifies the nominal foothold to a safe location on the terrain surface if the nominal location brings a risk to the locomotion. The LDM, instead, modifies the nominal foothold for collection in case a litter is detected inside the local collection region underneath the corresponding hip, at the moment just before the leg lift-off. If a detected litter is located in an unsafe region classified by the VFA, the modified foothold is not used for support and the collection is executed when the foot is still in the air. The CoM motion planning is also robot-centric and designed in the horizontal frame, whose target location is continuously computed according to the support polygon formed by the current location of the contact feet. The target can be tuned to prioritize the stability margin or the robot speed. The desired body height and orientations depend on the quality of the visual information. If the terrain map acquired around the robot (with exteroceptive sensors like depth cameras) is accurate, the desired body height and roll/pitch orientations are provided by our Visual Pose Adaptation (VPA) [Fahmi et al., 2023]. In case of unreliable terrain mapping, a desired relative body height is set and the body orientation follows the one estimated by the PTE. This selection is represented by the switch between VPA and PTE illustrated in Fig. A.5. The target location inside the support polygon is tuned for requirements regarding maximum locomotion speed and robustness. The motion generation provides the motion control block with desired body CoM positions, trunk orientation, and desired joint trajectories. The desired joint trajectories are obtained by transforming the desired foot trajectories from the horizontal frame into the robot base frame and applying inverse kinematics (see [Barasuol et al., 2013] for further details).

The next section describes the control actions to track the desired references from the motion generation block.

A.4.2 Motion Control

The robot's motion control block is composed of a Model-Predictive Controller (MPC) [Amatucci et al., 2022] in series with a Whole-body Controller (WBC) [Fahmi et al., 2019], and a joint proportional-derivative controller. The MPC controller computes ground reaction forces to track the desired CoM references from the motion generation block. It runs at 150Hz and considers the Single-rigid Body Model (SRBM) [Orin et al., 2013] as a simplified robot model used for state predictions [Grandia et al., 2019, Bledt and Kim, 2019, Wu and Sreenath, 2015]. The WBC runs at 250Hz and computes the joint torques to track the equivalent CoM wrench (obtained from the MPC ground reactions forces), considering the influence of leg inertial effects and the physical constraints given by the robot and environment. Completing the control action, a joint proportional-derivative controller contributes with joint torques to cope with model uncertainties and to better track the leg swing motion. The SRBM used for the MPC state predictions has shown to be representative even though the inertia of the robot legs are neglected. Using this model approximation, the system dynamics can be written as follows:

$$\ddot{\boldsymbol{p}}_c = \frac{1}{m}\sum^{n_{\text{leg}}} \boldsymbol{f}_i + \boldsymbol{g} \tag{A.10}$$

$$\dot{\boldsymbol{R}} = \boldsymbol{R}^{\mathcal{B}}\hat{\boldsymbol{\omega}} \;\; \boldsymbol{R} \in SO(3) \tag{A.11}$$

$${}^{\mathcal{B}}\boldsymbol{I}^{\mathcal{B}}\dot{\boldsymbol{\omega}} = \boldsymbol{R}^T(\sum^{n_{\text{leg}}} \boldsymbol{r}_i\hat{\boldsymbol{f}}_i) - {}^{\mathcal{B}}\hat{\boldsymbol{\omega}}{}^{\mathcal{B}}\boldsymbol{I}^{\mathcal{B}}\boldsymbol{\omega} \tag{A.12}$$

where $\ddot{\boldsymbol{p}}_c$ is the CoM acceleration, $\boldsymbol{f}_i$ is the GRF acting on the i^{th} leg, $\boldsymbol{R}$ and $\dot{\boldsymbol{R}}$ are the rotation matrix representing the rotation between the world and the robot base and its derivative, while $\boldsymbol{\omega}$ and $\dot{\boldsymbol{\omega}}$ are the angular velocity and its derivative. The variable $\boldsymbol{r}_i$ is the vector connection the CoM and foot positions, the operator $\widehat{(\cdot)}$ maps the vector in a screw-symmetric matrix. Finally, $\boldsymbol{I}$ and m are the inertia matrix and mass of the robot body, and $\boldsymbol{g}$ is the gravity vector. Equation (A.12) is non-linear in the angular part, therefore we use the variation-based linearization scheme presented in [Ding et al., 2021]. We express the rotational error in $SO(3)$, considering the variation to the operating point to be free from singularities in the representation.

We then perform a first-order Taylor expansion of the matrix exponential to then vectorize the error expressed in $SO(3)$ as $\boldsymbol{\xi} \in \mathbb{R}^3$ such that $\hat{\boldsymbol{\xi}} = \delta\boldsymbol{R}$. The linearized dynamics is finally discretized using the forward Euler scheme. The system state is defined as $\boldsymbol{x} = \left[\boldsymbol{p}_c, \dot{\boldsymbol{p}}_c, \boldsymbol{\xi}, {}^{\mathcal{B}}\boldsymbol{\omega}\right] \in \mathbb{R}^{12}$ and the control input as $\boldsymbol{u} = [\boldsymbol{f}_1, \boldsymbol{f}_2, \boldsymbol{f}_3, \boldsymbol{f}_4] \in \mathbb{R}^{12}$. Given the simplified model, the system state, and control input, we can define the Optimal Control

Problem (OCP) as:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \mathcal{L}_T(\boldsymbol{x}(N)) + \sum_{k=0}^{N-1} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ s.t. \quad & \boldsymbol{x}_{k+1} = \boldsymbol{D}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{c}_k \\ & \boldsymbol{u}_k \ \in \ \mathsf{U}_k \\ & k = 0, 1, ..., N-1 \\ & \boldsymbol{x}_{k=0} = \boldsymbol{x}_{op} \end{aligned} \tag{A.13}$$

where $\mathcal{L}(\boldsymbol{x}(\cdot))$ is a convex quadratic cost over the user commanded velocities and body posture. $\boldsymbol{D}_k, \boldsymbol{B}_k, \boldsymbol{c}_k$ are the linearized dynamics and U_k is the set of feasible ground reaction forces constrained by the outer pyramid approximation of the friction cone to guarantee non-slipping conditions. $\boldsymbol{x}_{op}$ is the state variable at the operating point. To solve the OCP we used a specialized QP solver [Pandala et al., 2019], that exploits the sparse structure of the problem. Finally, the GRFs obtained by solving the optimization problem (A.13) are then converted into the desired CoM wrench $\boldsymbol{w}_{u^*}$ sent to the WBC, as:

$$\boldsymbol{w}_{u^*} = f_w(\boldsymbol{u}^*)$$

where $f_w(\cdot)$ is the function that maps the optimal ground reactions forces from the MPC controller into forces and moments around the robot CoM.

To realize $\boldsymbol{w}_{u^*}$, taking into account the robot full-dynamics, limitations, and environment constraints, the WBC solves a QP problem with a cost functional that penalizes the wrench tracking errors and the deviations from $\boldsymbol{u}^*$, i.e.:

$$\begin{aligned} \min_{\boldsymbol{\gamma}=[\ddot{\boldsymbol{q}}_{\boldsymbol{b}}{}^T, \ddot{\boldsymbol{q}}_{\boldsymbol{l}}{}^T, \boldsymbol{f}_g^T, \boldsymbol{\epsilon}]^T} \quad & ||\boldsymbol{M}_b \ddot{\boldsymbol{q}}_b - \boldsymbol{w}_{u^*}||^2_{\boldsymbol{Q}_w} + ||\boldsymbol{f}_g - \boldsymbol{u}^*||^2_{\boldsymbol{Q}_f} \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{\gamma} = \boldsymbol{b} \\ & \underline{\boldsymbol{d}} < \boldsymbol{C}\boldsymbol{\gamma} < \bar{\boldsymbol{d}} \end{aligned} \tag{A.14}$$

where the decision variables are the base accelerations $\ddot{\boldsymbol{q}}_{\boldsymbol{b}}$, the leg accelerations $\ddot{\boldsymbol{q}}_{\boldsymbol{l}}$, the ground reaction forces $\boldsymbol{f}_g$, and the slack variables $\boldsymbol{\epsilon}$ for constraint relaxation. The equality constraints take into consideration dynamic consistency, leg inertial effects, joint kinematic and torque limitations, and friction constraints. All constraints are stacked and described through the matrices $\boldsymbol{A}$ and $\boldsymbol{C}$ and boundaries $\underline{\boldsymbol{d}}$ and $\bar{\boldsymbol{d}}$. Finally, the values optimized by Eq. (A.14) are mapped into joint torques using the whole-body dynamics equation. For

further details on the description of the constraints and the mapping into joint torques, see [Risiglione et al., 2022].

The readers can observe in the accompanying video of [Amatucci et al., 2024b]the behavior of our locomotion controller in challenging scenarios, such as the one with gravel or the one with stairs.

A.5 Results

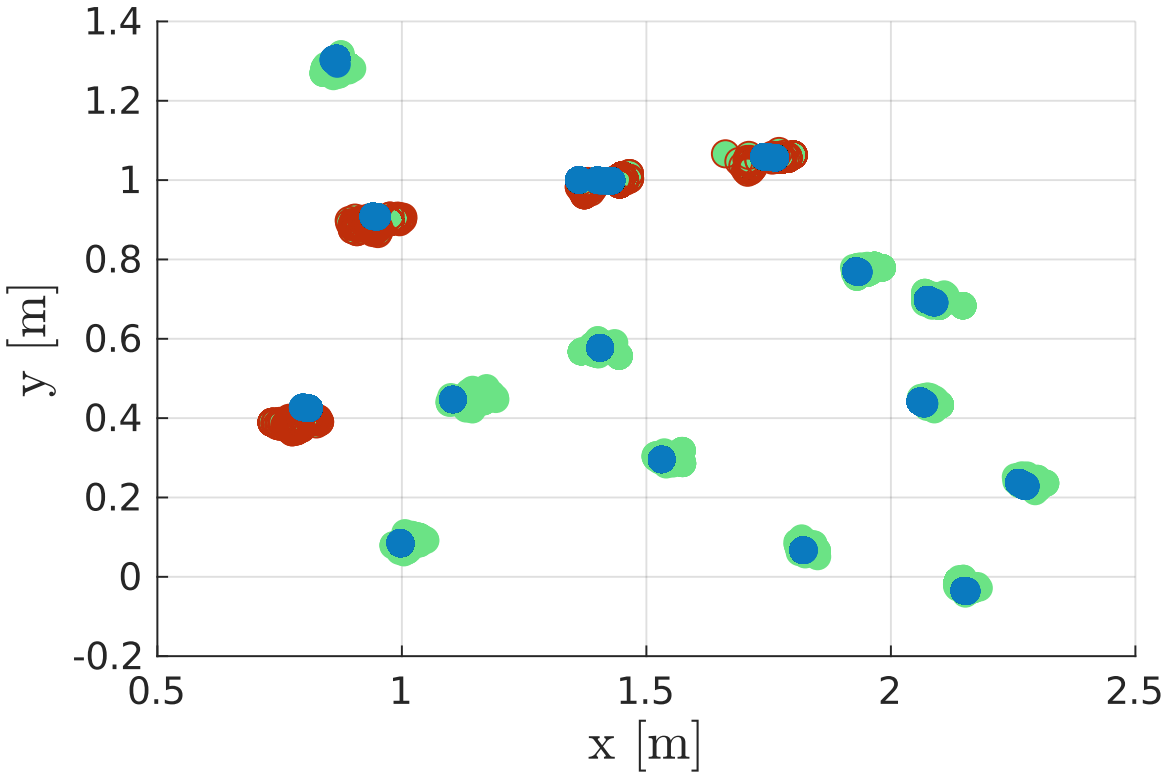

Figure A.6 The estimated CB positions during a collection test in our laboratory, with (blue) and without (green) employing the IEKF. In red, we highlight the errors in data association due to unfiltered outliers, which will cause the robot to perform additional collection attempts.

This section presents the results obtained with our prototype in both indoor and outdoor scenarios. The indoor scenario, represented by our laboratory, was chosen to validate the IEKF described in Sec. A.3.2 in a controlled environment, where multiple CBs are placed on the floor. The test result is shown in Fig. A.6. Due to the drifts in the estimator, the raw CBs positions (green dots) vary significantly in the map with respect to the estimation

Figure A.7 The prototype has been tested successfully in six different scenarios. Starting from the top left corner, we show the robot, controlled with a joystick for safety reason, traversing environments named *beach* (A), *urban* (B), *industrial* (C), *natural* (D), *non-flat* (E) and *park* (F). All these scenarios are characterized by unique challenges both from the locomotion point of view, such as stairs and slopes (scenarios A, D, E, and F), and from the detection point of view due to the presence of similar objects in size and color (e.g. small leaves in scenarios D and F).

coming from the filter (blue dots) during robot motion. It can be observed that without the filtering procedure, the data association process can fail without the possibility of recovery (red dots), meaning that the same object is added more than once to the list of the CBs to collect. In this case, the robot tries to collect the same object multiple times, drastically affecting the speed and collection time of the overall collection process.

Fig. A.7 shows six real-world scenarios in which we successfully tested our solution: *beach* (A), *urban* (B), *industrial* (C), *natural* (D), *non-flat* (E) and *park* (F). During our experiments, we repeatedly found CBs dispersed into these environments, showing that the

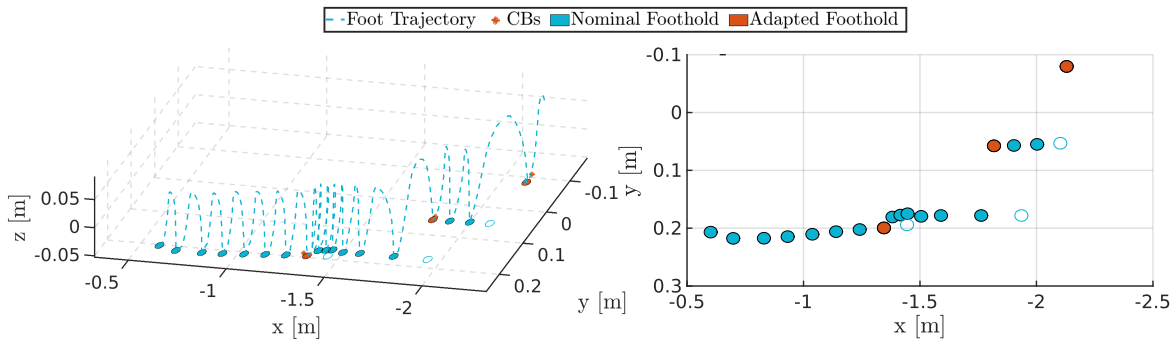

Figure A.8 Trajectory of the left-front foot (where the nozzle of the vacuum cleaner is attached) during a collection procedure in Scenario A. On the left, the dot-lines represent the swing trajectory, while the circles represents the footholds. A circle is blue when the touch-down coincides with the nominal foothold, computed by Eq. (A.8). The red circles represent the touch-down points corrected by the LCM. The corresponding nominal foothold is depicted by the white circles with blue borders. On the bottom, we plot the top-down view of the same foothold locations

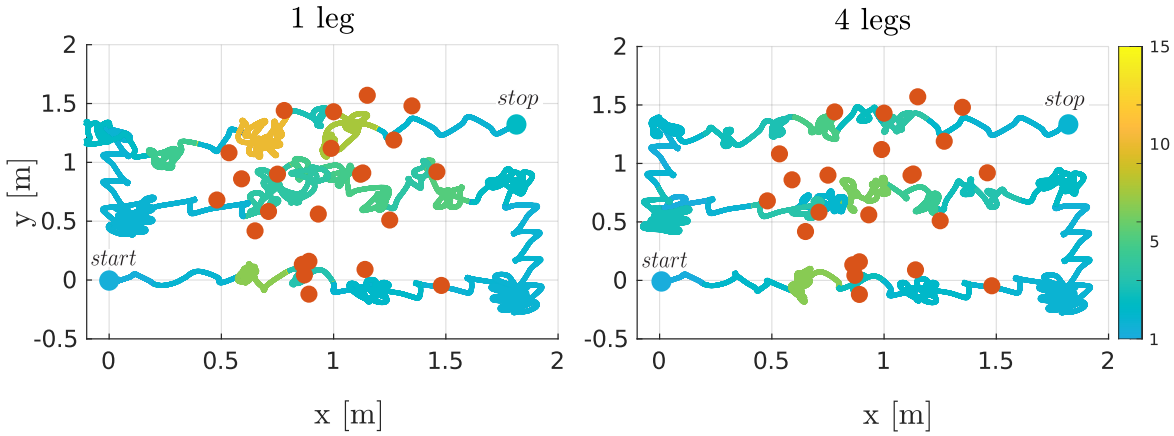

Figure A.9 Comparison of the robot CoM position during a collection procedure when the nozzle is attached to only one leg (left) and when the vacuum cleaner is connected to all four legs (right). The red disks represent the CBs. The line color indicates the ratio between the constant velocity obtained by the robot simply following the path and the actual velocity during collection. In both cases, the robot shows a sway motion due to the crawl gait, but as expected, more maneuvers are needed when only one leg is used to collect the CBs.

chosen scenarios are representative of the tackled problem. We chose six different scenarios, instead of a single one, to test the unique challenges that each of them contains that can hinder a reliable CBs collection. Specifically, Scenario A is characterized by the presence of

small pebbles that can get sucked in by the nozzle of the vacuum cleaner and deformable terrain, which represents a challenge for the locomotion controller of a quadruped. Scenario B is a common city street, where sidewalks and the presence of cars limit the traversability capabilities of many robotic systems. Similar limitations can be found in an industrial setting, such as the one in Scenario C, and in a small alley (Scenario E), where the presence of stairs limits the usage of wheeled robots. Scenario D represents a trail outside Genova, where the detection of CBs can be problematic given the presence of flora with similar sizes and colors. The same reasoning applies to Scenario F.

We observed difficulties in the detection of the CBs in the scenario depicted in Fig. A.7D given the presence of similar objects, in color and shape, in the local flora. This issue has been mitigated by collecting additional images in that scenario and performing a successive fine-tuning of the detection model. This result shows that still, many misclassifications can still happen in a real environment, and additional data-synthesis techniques, such as using realistic simulators for data gathering, should be adopted to enhance the precision of the LCM.

In our final tests, our prototype successfully recognized and collected all the undisposed CBs. To give an intuition of the performance of the LCM, in Fig. A.8 we plot the trajectory of the left-front foot in the X-Y-Z plane. The task is to collect three different CBs. The circles represent the touch-down points, and the dashed lines the swing trajectories. The blue color indicates that the visual servoing module did not change the desired foothold locations, hence the robot has stepped in the nominal location computed by Eq. (A.8). The LCM is activated when the foot is close to a CB (red cross), causing the robot to perform a touch-down event in the vicinity of the CB (the red circle) instead of on the discarded nominal location (white circle with the blue border). In this experiment, the robot was controlled with a joystick and commanded with a user-defined non-constant forward velocity. Since the step frequency remains fixed in our locomotion controller (Sec. A.4), the nominal footholds are not distributed uniformly in the images. Here the reader can observe that, since the nominal footholds are coherent with the reference velocity, in order to suck the CBs the robot needed to step toward the left. In our pipeline the hip is positioned on top of the CB during the sucking, so after performing the collection, the robot keeps going straight forward from the reached position till the next CB.

The readers can refer to the accompanying video of [Amatucci et al., 2024b] for the highlights of our results.

Table A.1 Statistical Analysis

Description	Success	Maximum	Rate
Overall pipeline: collected/total CBs	29	37	78,37
Detection: detected/total CBs	33	37	89,18
Neural Network: detected/seen CBs	33	35	94,28
Collection: collected/detected CBs	29	33	87,87

A.6 Discussion

Our prototype is endowed with a vacuum cleaner and a nozzle connected to the left-front leg. We want to point out that our design can easily be modified to connect the vacuum cleaner to all the available feet. In this way, the number of maneuvers required to collect the CBs can be reduced if a different foot can be exploited more easily from the current robot position. For this, only the Dijkstra algorithm in the LCM (Sec. A.3.3) needs to be updated to consider multiple hip positions in the planning problem, hence computing the optimal robot's pose and selecting which leg to use to collect each CB. No changes are required in the visual servoing of the LCM.

To highlight this concept, we compare the performance in simulation of the same pipeline having the nozzle attached to one leg only (Fig. A.9 left) and on all four legs (Fig. A.9 right). In simulation, we defined an S-shaped path with 90-degree angles placing some CBs to be collected (the red disks in Fig. A.9). The lines' colors indicate the velocity deviation with respect to the "ideal" one obtained with the robot walking on the desired path without performing any collection. By comparing the left and the right plot, the readers can see the advantages of having multiple nozzles (higher velocity). This is explained by considering that if, for example, a CB is on the right side of the robot, the robot needs to perform a rotation to bring the left-front foot close to the CB. Even though we have not tested the vacuum cleaner with four nozzles, we have demonstrated that the software is general and can be adapted to a different configuration of the hardware prototype.

To deal with the problem of the sucking of the small pebbles in Scenario A, we experimentally adapted the distance of the nozzle with respect to the base of the foot. Given the usually lighter weight of the CBs, we were able to tune this parameter and obtain successful collections with only few pebbles sucked in by the machine. Nevertheless, we needed to empty the vacuum cleaner from time to time during the experiment. In future works, we plan to integrate a simple mechanical separation system tailored to deal with this

specific problem into our prototype. We will also assess in more detail the effect of nozzle design and position from the foot base, taking also into consideration a variable sucking power. We plan in this way to enable a higher level of autonomy in all scenarios.

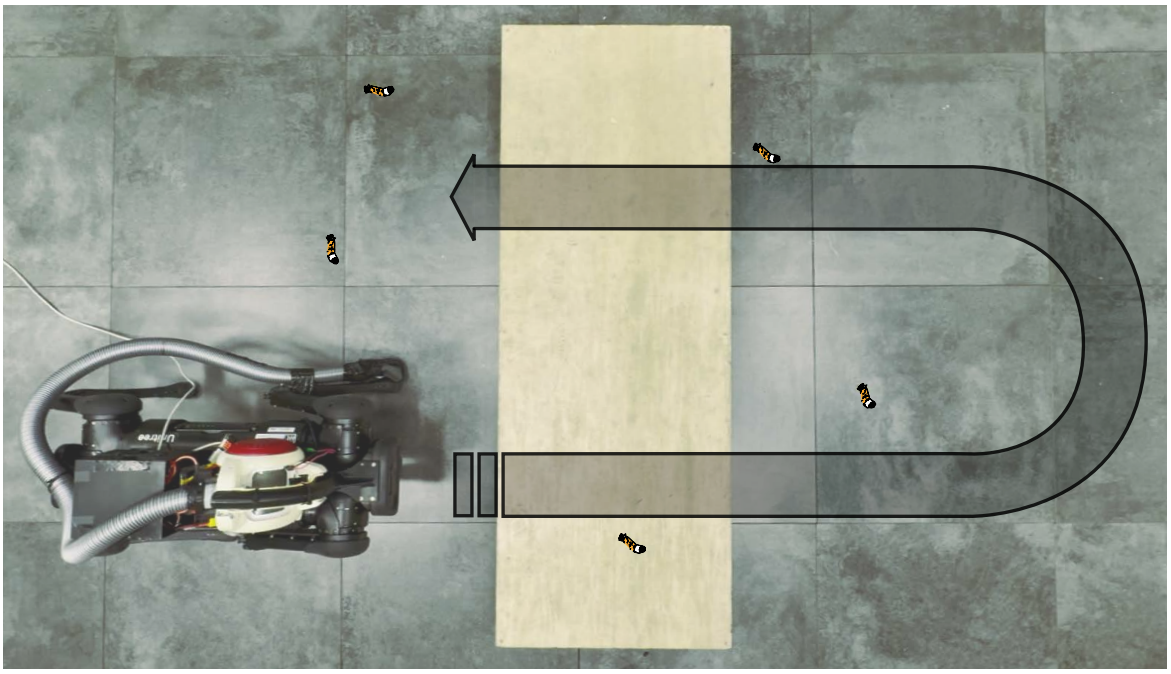

Figure A.10 Configuration of the test course used to assess the framework under controlled conditions. The robot's predefined path is marked with a translucent gray arrow, and multiple cigarette butts (CBs) are randomly distributed on the ground. Additionally, a wooden obstacle 12cm high is placed in the center of the scenario.

To evaluate the performance of the overall pipeline, we designed a test course in our laboratory, as shown in Fig. A.10, where the robot is commanded to explore repeatedly a predefined area, and a wooden obstacle is placed in the center to challenge the litter collection procedure. In this area, during each pass of the robot, we randomly spread a different number of CBs (4, 5, or 6) to evaluate the efficiency of the entire litter removal pipeline and its critical points. During 8 trials, the robot collected 29 out of 37 CBs. which corresponds to a success rate of 78,37% for the overall system. Further analysis of the failures (uncollected CBs) revealed that in 4 cases the CB was not detected: in two cases the CBs never entered the field of view of the robot's cameras during the entire lap, while in the other two cases, the network in the LDM did not identify the CBs with sufficient certainty. Considering both cases, the overall detection accuracy was 89% (33 detected out of 37). Considering only the CBs which have been seen (35) the accuracy of the network is

94,28%. The remaining 4 misses were caused by the LCM. In three instances, the robot's nozzle edge made contact with the CBs, rendering them impossible to be sucked. This failure indicates the importance of stepping precision. In the last cases, the CBs never entered the down-facing camera's field of view due to misalignment with the collection pose. Consequently, the robot was unable to perform the visual servoing correction and relied solely on the mapped position of the CBs. However, due to some drift, the position in the map had degraded to the extent that proper collection performance was compromised, resulting in the robot missing the CBs. Nevertheless, our collection procedure, considering how many of the detected CBs have been collected, showed a success rate of 87,87% (29 out of a total of 33). A summary of this statistic can be found in Table A.1 where we report the achieved results, while the execution of this experiment can be observed in the accompanying video of the [Amatucci et al., 2024b].

A.7 Conclusion

In this work, we have presented both the hardware and the software of a quadruped robot able to autonomously and efficiently collect small litter during locomotion. The presented approach takes advantage of a currently poorly exploited feature of legged robots: the ability and freedom to select footholds not only for a stable locomotion, but also for specific applications. To the best of the authors' knowledge, this is the first time that the legs of a legged robot are *concurrently* utilized for locomotion and for a different task. Differently from arms mounted onto mobile robots, which add weight, complexity, cost, and energy demand, exploiting legs for purpose-oriented interactions with the ground is an elegant and minimalist solution. On one hand, it is time-efficient because the feet need to touch the ground anyway for locomotion, so no additional whole-body arm control is needed that is generally slower when contact with the environment is required. On the other hand, it is potentially more precise since the kinematic chain of a leg is usually shorter compared to the one of a mobile base with an onboard arm.

This minimalist approach can be exploited for many other applications that are different from the presented use-case of litter removal, including for example:

- in agriculture and forestry: for the selective spraying of weed in crop fields (e.g. salads), punctual measurements of soil properties (e.g. humidity), or for the injection of seeds into the ground (e.g. reforestation).

- in infrastructure inspection: for surface sensing where contact with the surface is required (e.g. crack detection).
- construction: placement of nails and rivets (e.g. construction of large wooden structures or steel vessels).

In future works, we want to analyze the possible gain in power-efficiency that our prototype can enable compared to the one achievable by using an additional arm on top of the robot to perform the litter collection. Furthermore, we aim to build and test the necessary hardware for enabling our prototype to use all the available feet for litter collection. Besides increased collection speed, equipping multiple feet with nozzles can easily be used for separate collection of different types of waste. In fact, our detection module (Sec. A.3.1) can be easily generalized for multiple objects, and each foot can take care of a specific type of waste.