Innovative On-the-Fly Approach to Soft Landings in Quadruped Robotics

Edoardo Panichi

Acknowledgments

I would like to express my sincere gratitude to my supervisor Cosimo della Santina, and my daily supervisor Jiatao Ding, who proposed this exciting thesis to me and helped me during the past months with the development of the project.
After that, a heartfelt thank you also goes to my friends with whom I have spent long study sessions over the years, which have allowed me to complete this master's; and to my family and girlfriend who have supported me in every way during my university career.
Last but not least, I would like to thank Albino Dallio who, especially in this last period, has been a great benefactor during the finalization of my thesis.

Supervisors:
- Dr. Ding, J.
- Dr. Della Santina, C.

Thesis Committee:
- Dr. Ding, J.
- Dr. Della Santina, C.
- Dr. Prendergast, J.M.
- Dr. Mazo Espinosa, M.

Date of defense: 27th of February 2024
Awarding Institute: The Delft University of Technology

Student Number: 5630444

Innovative On-the-Fly Approach to Soft Landings in Quadruped Robotics

Edoardo Panichi

***Abstract*—In this thesis, we introduce a novel approach aimed at enhancing the jumping and landing capabilities of quadruped robots. Our method integrates both model-based and model-free strategies and features a behavioral cloning framework designed to reduce computational delays often encountered in trajectory optimization.**

Initially, we build upon an existing framework for quadruped jumps, where we refine the trajectory optimization (TO) algorithm and introduce a new Variable Impedance Control (VIC). The VIC is specifically developed to facilitate softer landings. This improved system was then utilized to generate a comprehensive synthetic dataset, including $11,000$ samples that cover a diverse range of jumping scenarios. This dataset served as the foundation for training a neural network. The primary objective of the network is to emulate the performance of the model-based approach. Structurally, the network is designed to process the robot's current state as input and generate the corresponding control actions for its 12 motors as output.

The most significant achievement of this research is the neural network's ability to closely replicate the outcomes of the model-based solution. Notably, it ensures more compliant behavior and lower stress on the motors during the landing phase than an MPC. The neural network demonstrates a 97.4% success rate. This high level of performance underscores its potential for on-the-fly application in robotic systems. The effectiveness of our method is further validated through a series of simulations and practical tests conducted on a Go1 quadruped robot.

I. Introduction

In recent years, robots have rapidly spread across various industries and applications. From manufacturing to logistics, healthcare to services, robots are being used to automate tasks, improve efficiency, and reduce costs. Technological advancements, such as in the field of artificial intelligence, machine learning, and sensors, have enabled robots to perform increasingly complex and sophisticated tasks while becoming more convenient, helpful, and accessible.

In the evolution of technological innovation, the natural world has consistently served as a pivotal source of inspiration by offering solutions refined over eons of evolution. The emergence of humanoid and quadrupedal design is a primary manifestation of this in the realm of legged robotics. Traditionally, wheel-based machines have dominated the field due to their simplicity, efficiency, robustness, and cost-effectiveness in both production and maintenance. However, recent years have brought a remarkable shift in research focus, driven by the compelling advantages of legged robots sparking significant interest and enthusiasm among researchers.

Legged robotics is gaining prominence due to its superior mobility and adaptability compared to traditional tracked or

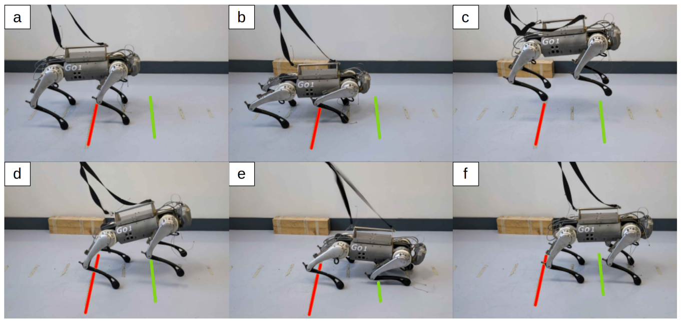


Fig. 1: The Go1 performs a jump using the method proposed in this paper. The distance between the green and the red line is $40cm$.

wheeled vehicles. Legged robots excel in navigating complex, uneven terrain, making them invaluable for disaster relief operations and exploring uncharted areas, including challenging environments like stairs, rocky terrain, and confined spaces. In the early years of the advent of quadrupeds, researchers focused their efforts on solving the locomotion problem [1][2][3], and only after they focused on more dynamic skills such as jumping [4][5][6]. Despite being different problems, locomotion and jumping share many solutions. Both of these problems can be solved following two main paths: *model-based* and *model-free* approaches. The former operates within a structured framework, primarily comprising a planner, such as a trajectory optimization [7], which delineates the motion to be executed, and a controller, which generates the control actions to track the planner's reference trajectory. The quintessential advantage of the model-based paradigm lies in its integration of a detailed robot model. This model enables the planner, and in certain instances the controller (e.g., Model Predictive Control), to anticipate the robot's motion and deduce the most effective strategy for attaining desired objectives. Solutions derived from this approach are anchored in theoretical robustness and offer straightforward interpretability due to their foundation on established mathematical principles and physical laws.

Conversely, model-free methodologies exhibit distinct advantages, particularly in contexts where the formulation of precise models is challenging. These strategies develop a control policy through direct interaction with the environment or by learning from a dataset. They excel in adapting to dynamic and unpredictable environments, due to their independence from pre-established models. This attribute renders model-free approaches apt for handling complex tasks characterized by unpredictable or non-linear elements, which are notoriously difficult to encapsulate accurately within a model.

Furthermore, it is imperative to consider the temporal efficiency of these systems. Model-based solutions, especially those employing complex models, often require substantial computational time to plan actions. In contrast, model-free solutions do not encounter this obstacle. Therefore, model-free methods emerge as a preferable alternative in contexts where adaptability, swift responses, and reduced computational demands are paramount.

In this work, we propose a novel approach that exploits the best of both worlds for robotic jumping tasks. Traditional model-based methods often require trade-offs for on-the-fly applications, compromising jump performance. Our solution involves a supervised learning framework that replaces the planner and controller, enabling on-the-fly jumps without performance loss.

Initially, we developed a model-based framework with a trajectory optimization (TO) algorithm and controller, used to generate a synthetic dataset of $11,000$ jumps. This dataset was used to train a neural network capable of jumping various distances with the same set of weights, achieving performance comparable to the model-based method but without its computational burden.

The focus of this study is not limited to executing accurate jumps but also discusses how to achieve soft landings with impact absorption techniques to enhance the practicality and safety of jumps. Our main contributions to this research are summarized as follows:

- We proposed an advanced single rigid body model (SRBM) trajectory optimization algorithm, inspired by [8], capable of generating more informed reference trajectories than those typically based on Spring-Loaded Inverted Pendulum (SLIP) models.
- We developed a compliant landing controller, incorporating the advantages of variable impedance controllers [9]. This approach aims to minimize impact forces and motor efforts during landings.
- We utilized our model-based framework to generate a synthetic dataset essential for training a behavioral cloning algorithm. This algorithm successfully replicates the results of the model-based solution mentioned in the two previous points, bypassing the computational inefficiencies associated with the planner.
- We thoroughly evaluated our contributions in both simulated environments and real-world scenarios using a Go1 quadruped robot by Unitree, demonstrating the practical application and effectiveness of our approach.

II. Relevant Literature

A. Model-Based Solutions

Model-based approaches, long the cornerstone of robotics, excel for basic robots and tasks. Even complex robots like quadrupeds can be effectively managed with simplified models for basic actions, facilitating real-time controller and on-the-fly planner operation while ensuring robust control.

Yet, challenges arise with complex robots and dynamic tasks like quadruped jumping, necessitating sophisticated models due to intricate dynamics. To manage this, planning and control are often split into two phases: an offline phase for complex motion problem-solving, and an online phase focused on tracking the planner's reference trajectory.

A prevalent approach in the planning phase is trajectory optimization, which involves defining a cost function and a set of constraints. This method has enabled various achievements, including planning jumps through window-shaped obstacles [10], continuous jumping [11], and omnidirectional jumps [12]. To decrease computational load, some strategies involve manually setting contact phases and durations, sacrificing optimality in the process [13]. Others adopt a reduced-order model for optimizing these specific parameters [14]. However, even with these approaches, planning a reference trajectory can require several minutes.

Efforts to achieve online trajectory optimization are also present in the literature. Chignoli et al. [15] propose a solution that disregards joint dynamics and kinematics during the stance phase, under the assumption that feet within an approximate Cartesian workspace relative to their hips will not violate kinematics limits or collide. Alternatively, Mastalli et al. [16] showcase agile online locomotion through Differential Dynamic Programming-based model predictive control, though this too relies on a separate planner for contact scheduling. Recent developments have also successfully integrated model-based approaches with machine learning algorithms to reduce computational complexity, achieving remarkable online results [12][17]. None of the work mentioned though, emphasizes soft landing.

B. Model-Free Solutions

Model-free strategies, particularly reinforcement learning (RL) and imitation learning, are emerging as potent tools for complex tasks like jumping. These methods depart from traditional model-based approaches, using data-driven techniques to learn control policies either through direct environmental interaction or by analyzing existing datasets. RL has facilitated significant advancements in teaching legged robots to perform complex maneuvers, including locomotion [18][19][20][21][22], and jumping [23][24][25]. For instance, Bellegarda et al. [26] showcase the ability of RL to facilitate robust jumping, overcoming environmental challenges. However, their study does not address the issue of soft landing. Another notable work by Qi et al. [27] employs RL to enable landing on an asteroid, encompassing take-off, attitude adjustments, and soft landing, with the lower gravitational field simplifying control requirements.

Imitation learning [28] has also shown its efficacy in this domain. This technique involves teaching quadruped robots complex skills, such as jumping, by mimicking behaviors of expert human operators or simulations. This approach, combined with RL, has been applied to refine jumping techniques from various inspirations, including the animal kingdom [29][30], manually crafted trajectories [31][32], and optimization-based methods [23][33][26]. Behavioral cloning (BC), a variant of imitation learning, has yielded impressive results in other areas of robotics, though less explored in dynamic quadruped movements [34][35][36].

Concerning jumps, there seems to be a lack of studies utilizing BC techniques. Kurtz et al. [37] appears to be the closest study to a jumping application, where a synthetic dataset was used to train a model on how to land a robot on its feet after a fall, using inertia-shaping techniques for midair reorientation. An advantage of BC, particularly when starting from trajectories generated by optimal strategies (e.g., TO paired with MPC), is that it allows the model to learn a policy that replicates precise results while avoiding the computational overhead of TO. Furthermore, unlike RL, BC avoids the complexities associated with tuning reward functions to reach desired outcomes [38].

C. Soft Landing

Soft landing is a critical feature in quadruped robotics, essential for reducing mechanical stress and ensuring stability post-jump. It extends the robot's lifespan and ensures safe, balanced landings, especially in uneven or challenging environments, making it a key focus for enhancing robotic agility and durability. A soft landing is characterized by two main features: compliance in the movements and low stress on the motors [39]. In this context, Jeon et al. [8] developed an optimal landing controller that operates online without pre-specified contact schedules, effectively managing touchdown postures and force profiles for drops up to 8 meters. Complementing this, Nguyen et al. [14] applied an optimization framework to determine optimal contact timings and a reference trajectory for real hardware, achieving stable landings from 2 meters. Both of these works show impressive achievements but are focused on falls rather than jumps. Qi et al. [40] diverged with a model-free reinforcement learning approach, focusing on adaptive landings on asteroids, highlighting the importance of orientation adjustments for optimal ground contact, which is much more difficult when dealing with Earth's gravity. A different approach to the soft landing problem solves the issue by acting directly on the hardware, designing the legs of the robot to include compliant elements that help to reduce stress on the motors and improve the compliance of the movements.

Additionally, advancements in hardware, such as incorporating compliant elements into robot legs, offer direct stress reduction on motors and enhanced movement compliance, as demonstrated in multiple works [41][42][43][44][45].

Given the current state of the art, the **research goal** of this work is to develop a *on-the-fly* algorithm to perform *precise* jumps and *soft landing*.

III. Problem Statement

The primary objective of this research is to design and implement a technique enabling a quadruped robot to execute a precise jump, culminating in a soft landing. This process must be efficient enough to allow for on-the-fly execution without necessitating extensive computation to determine the optimal motion trajectory. Specifically, the robot should be capable of jumping distances ranging from $0.1m$ to $0.55m$, maintaining minimal error at the final position. Our experimental platform is the Unitree Go1, chosen for its representative features common to similar quadruped robots.

The state of the robot at any given moment is encapsulated by a 35-dimensional vector:

$$\mathbf{X} = [z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z, q, \dot{q}, d]. \tag{1}$$

Here, z represents the center of mass (CoM) position, while ϕ, θ, ψ denote roll, pitch, and yaw (RPY) respectively. The linear velocities of the CoM are expressed as $\dot{x}, \dot{y}, \dot{z}$, and angular velocities are captured by $\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z$. The joint positions and velocities are represented by $q \in \mathbb{R}^{12}$ and $\dot{q} \in \mathbb{R}^{12}$, respectively, and d signifies the targeted jump distance. Notably, the state vector $\mathbf{X}$ omits the x and y positions of the robot's CoM since the jump mechanics are invariant to these dimensions—different values of x and y do not alter the necessary control sequence for the desired jump. The conventional state of the robot that includes x, y and neglects d will be addressed as $\mathbf{X}^+$

$$\mathbf{X}^+ = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z, q, \dot{q}]. \tag{2}$$

In this setup, the input $\mathbf{u}$ to our robot is represented by the joint torques τ. The problem we seek to solve is formally defined as:

Problem: *Starting from an initial stance position denoted by* $\mathbf{X(0)}$*, identify the control action sequence* $\mathbf{u}$ *that enables the robot to achieve a jump of the desired length* d*. This sequence should be optimized to minimize motor effort and ensure smooth deceleration during the landing phase for a soft landing. Moreover, the computation of* $\mathbf{u}$ *needs to happen in real-time while ensuring small landing errors.*

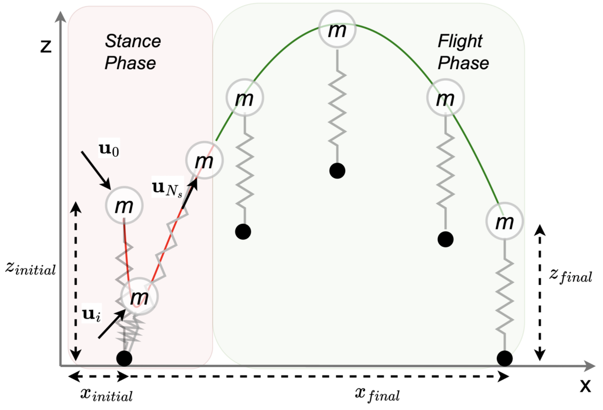


Fig. 2: This figure presents an example trajectory generated through TO using the aSLIP model. The trajectory of the CoM in the X-Z plane is delineated in green for the stance phase and red for the flight phase. Control inputs during the stance phase are indicated by vectors $\mathbf{u}_1$ where $i \in 0, ..., N_s$. Inputs to the TO are the initial and targeted final positions of the CoM. For illustrative purposes, the foot position is also shown but is not included in the TO. The TO outputs the CoM's position and velocity at discrete intervals defined by dt_s for the stance phase and dt_f for the flight phase.

IV. Methodology

A. Starting Framework

Our research builds upon an established framework, previously developed by colleagues of our research group. This framework integrates a kino-dynamic trajectory optimization algorithm with a Model Predictive Control for effective trajectory tracking. The TO algorithm, central to our approach, employs an actuated Spring-Loaded Inverted Pendulum (aSLIP) model [46]. It is characterized by a predefined number of steps, N, equally divided into stance (N_s) and flight (N_f) phases. The step size, dt, is dynamically optimized by the TO algorithm, with an initial value provisioned to enhance computational efficiency. The full problem formulation for the TO of the starting framework is reported in the Appendix A.

The initial implementation also includes an MPC, adapted from Di Carlo et al. [47], for real-time reference tracking at a $500Hz$ frequency. Additionally, it employs *PD feedback controller* for maintaining the desired pose and a *contact controller* to ensure effective ground contact during the jump. This comprehensive setup aims to optimize the robot's jumping trajectory while respecting its physical constraints and operational capabilities. The overall framework flow can be visualized in Figure 3.

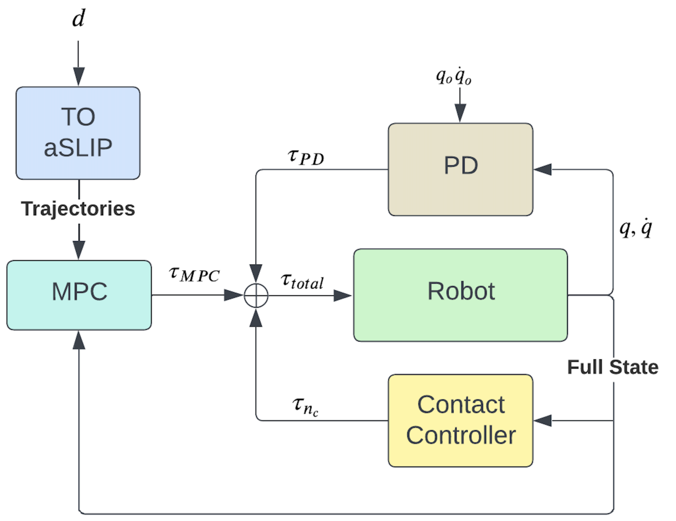


Fig. 3: This figure presents the overall starting framework described in subsection IV-A. The light blue block represents the planner and it gets as input d the desired jump distance. From the graph we see the action of the three controllers: MPC, PD, and Contact Controller.

B. Single Rigid Body Model Trajectory Optimization

The first step to achieving the research goal of this study is to improve the accuracy of the starting framework. The initial setup's main limitation was its oversimplified model used in the TO able to capture only partial dynamics. This led to less trackable and physically viable trajectories for the robot. To overcome this, we shifted from the aSLIP model to a more comprehensive Single Rigid Body Model (SRBM). The SRBM, with six D.o.F, offers greater planning accuracy without significantly increasing complexity. Our revised TO draws inspiration from the work of Jeon et al. [8], albeit with significant alterations. Notably, while Jeon et al.'s work primarily addresses falls, our revisitation modifies its application to work with jumps. The reformed TO algorithm is delineated over $N = N_s + N_f$ steps, and is expressed as follows:

$$\begin{aligned}
\underset{\mathbf{X}^+}{\text{argmin}} \quad & \|\tilde{\mathbf{X}} - \tilde{\mathbf{X}}_{\text{ref}}\|_Q^2, && \text{(3a)}\\
\text{s.t.} \quad & \textbf{Kinematic constraints:} \\
& \mathbf{X}^+(0) = \mathbf{X}_0^+, && \text{(3b)}\\
& r(0) = r_0, && \text{(3c)}\\
& \mathbf{X}_{1:3}^{+,\text{goal}} - \epsilon \le \mathbf{X}_{1:3,N}^+ \le \mathbf{X}_{1:3}^{+,\text{goal}} + \epsilon, && \text{(3d)}\\
\forall k \in & [1, 2, \dots, N_s + N_f] : \\
& r_k = \mathcal{F}(q_k), && \text{(3e)}\\
& \|h_k - r_k\| \le L_{\text{leg,max}}, && \text{(3f)}\\
& \mathbf{X}_k^+ \in \mathcal{B}, && \text{(3g)}\\
& \textbf{Dynamics constraints:} \\
& \dot{\mathbf{X}}_{k+1}^+ = f(\mathbf{X}_k^+, r_k, F_k), && \text{(3h)}\\
\forall k \le & N_s : \\
& F_k \in \mathcal{S}, && \text{(3i)}\\
& CCC(F_k, r_k), && \text{(3j)}\\
& |J^T F_k| \le \tau_{\max}, && \text{(3k)}\\
& -\mu F_{k,z} \le F_{k,x} \le \mu F_{k,z}, && \text{(3l)}\\
& -\mu F_{k,z} \le F_{k,y} \le \mu F_{k,z}. && \text{(3m)}
\end{aligned}$$

The algorithm optimizes the robot state, denoted as $\mathbf{X}^+$, which is elaborated in Equation (2) (Section III). The core of this algorithm is the cost function (Equation (3a)), which minimizes the error between the desired final state $\tilde{\mathbf{X}}_{\text{ref}}$ and the achieved state $\tilde{\mathbf{X}}$. $\tilde{\mathbf{X}}$ contains the first six states of $\mathbf{X}^+$ and the error is weighted by the matrix Q.

Constraints (3b) and (3c) define the initial conditions of the system. These include the starting state of the robot $\mathbf{X}_0^+$ and the initial position of the robot's feet r_0, expressed in global coordinates. Conversely, constraint (3d) delineates the terminal conditions, specifically targeting the CoM's final x, y, and z positions, with ϵ serving as a slack parameter to allow some flexibility. The kinematic aspect of the algorithm is captured in Equation (3e), which describes the forward kinematics for leg configurations. This equation transforms the joint configurations, denoted by q, into the feet's positions r in a global coordinate system. To aid the algorithm's convergence, inequality (3f) is introduced. This inequality relates the global position of the hip joint h to the maximum extendable length of the leg, represented as $L_{\text{leg,max}}$. Additionally, the constraint (3g) sets boundaries for the state variables $\mathbf{X}_k^+$.

The robot's dynamics, as defined in constraint (3h), treats it as a single rigid body. This simplification is justified by the fact that the legs constitute less than 10% of the robot's total mass.

$$m\ddot{\mathbf{X}}_{1:3}^+ = \sum_{i=1}^{n_c} F_i - f_g, \tag{4a}$$

$$\frac{d}{dt}(I\omega) = \sum_{i=1}^{n_c} (r_i - \mathbf{X}_{1:3}^+) \times F_i. \tag{4b}$$

The linear and rotational dynamics of the robot are governed by Equations (4a) and (4b). Specifically, the linear dynamics are defined as the sum of the ground reaction forces (GRF) acting on each foot (F_i), minus the gravitational force (f_g), acting on the CoM. The rotational dynamics are determined by the torque generated by these forces about the CoM. The

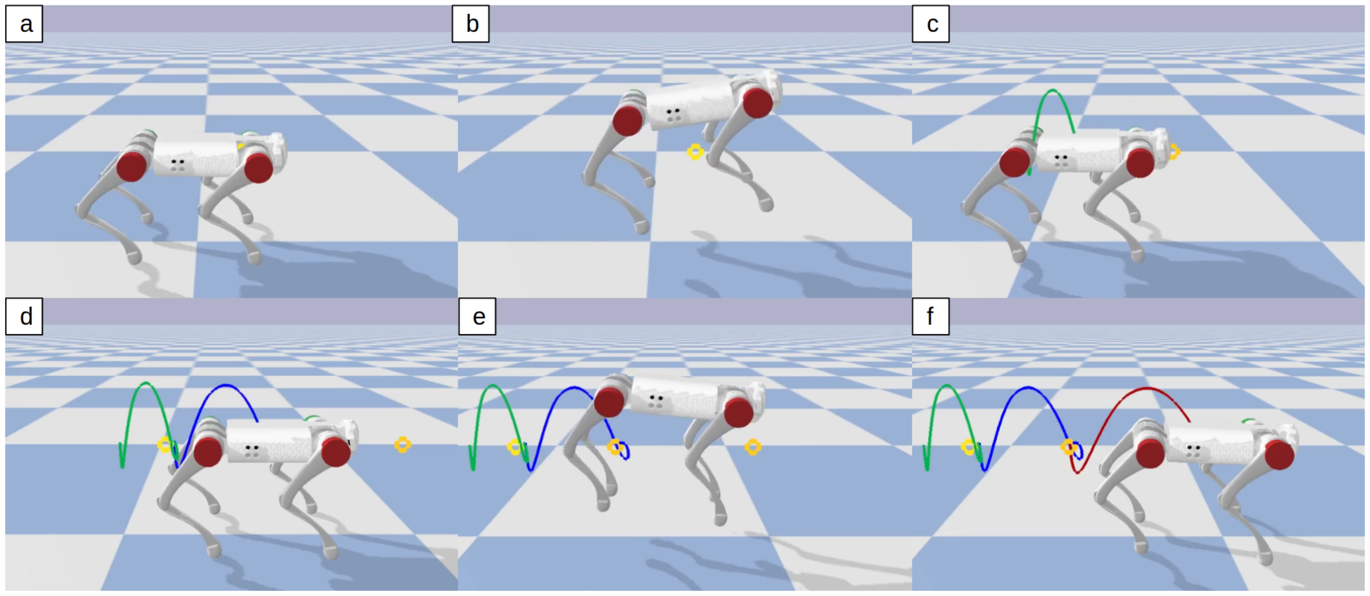


Fig. 4: The images from *a* to *f* show the robot making three jumps in a row. Each jump gets progressively longer, with the first jump (green) being 0.15 meters, the second (blue) 0.30 meters, and the third (red) 0.45 meters. The yellow circles indicate where the robot aims to land for each jump. Throughout this sequence, the robot is controlled by the *feedback NN*.

variable n_c represents the number of feet in contact with the ground, I is the rotational inertia tensor, and ω is the angular velocity of the robot's body.

When the robot is airborne, represented by the condition $k \in (N_s, N_s + N_f]$, the dynamics simplify considerably due to the absence of the GRF ($F_i = 0$). This simplification affects both linear and rotational dynamics, as described in the equations above.

The TO algorithm includes several critical constraints that ensure the stability and feasibility of the robot's jump. One such constraint is (3i), which establishes a specific set, denoted as $\mathcal{S}$, for the GRF. A GRF F_k within this set implies a symmetric distribution of forces across the robot's legs, as defined by the following relations:

$$\begin{aligned} F^{FR}_{k,z} &= F^{FL}_{k,z}\,, \\ F^{FR}_{k,y} &= -F^{FL}_{k,y}\,, \\ F^{RR}_{k,y} &= -F^{RL}_{k,y}\,, \end{aligned}$$

where FR, FL, RR, and RL refer to the Front-Right, Front-Left, Rear-Right, and Rear-Left legs, respectively. This symmetric configuration of GRFs is crucial for ensuring a straight trajectory during the jump, effectively mitigating any lateral (*y-axis*) or rotational (*yaw*) deviations.

Further contributing to the algorithm's robustness is the constraint (3j), inspired by [8], which enforces Contact Complementary Constraints. These constraints are crucial for preventing slippage and enhancing the convergence of the TO algorithm. They are represented as follows:

$$\begin{aligned} & r_{z,k} \geq 0, \\ & F_{k,z} \geq 0, \\ & F_{k,z} \cdot r_{k,z} \leq \epsilon, \\ & F_{k,z} \cdot (r_{k+1} - r_k) = 0. \end{aligned}$$

The inequality (3k), using the foot jacobian J and the maximum torque produced by each motor $\tau_{\max}$, limits the maximum GRF found by the TO, and to conclude, the compound inequalities (3l) and (3m) implement a friction cone constraint where μ is the friction coefficient between the feet and the ground.

C. Impedance Controller for Soft Landing

The second major challenge addressed in this work is the achievement of soft landings. A flexible solution, adaptable across various legged robotic platforms, encounters significant obstacles, particularly when limitations are imposed on auxiliary equipment. Our approach seeks to develop an effective solution that operates independently of additional sensors such as force sensors or vision systems. The absence of these tools necessitates a focus on enhanced control strategies. Consequently, we have opted to explore the development of an advanced Impedance Controller.

Impedance controllers, first introduced by Hogan in 1984 [48], are widely used in robotics for compliance, especially in human-interactive applications [49][50], and for reducing motor pick effort [51]. The key challenge with these controllers is the task-specific adjustment of impedance gains (stiffness K and damping D), where fixed gains are often inadequate. Current methods for variable impedance planning, relying on advanced force/torque sensors, are based on inertia shaping, which requires precise sensing [49]. This reliance limits their adaptability in various applications.

Addressing this challenge, Angelini et al. [52], and subsequently, Pollayil [9], introduced an innovative algorithm for autonomously planning impedance gains, stiffness, and damping in a Cartesian impedance controller. This approach, aimed at mitigating impact perturbations while maintaining tracking performance, has been successfully applied in quadrupeds for compliant locomotion and appears promising for achieving soft landings post-jump.

In practice, this translates to deactivating the MPC upon landing detection and transitioning to the impedance controller, as conceptualized by Angelini and Pollayil. The main goal of their algorithm is to minimize accelerations while also keeping tracking errors within an acceptable range. This is especially important during the landing phase, where achieving a compliant, or soft, landing is more crucial than maintaining exact tracking. The control scheme after this modification can be visualized in Fig 5. More information about this variable impedance controller (VIC) can be found in the Appendix B and the works of Angelini and Pollayil.

The pseudo-code block (1) illustrates the overall logic for the robot's controller. In the algorithm $\mathcal{S} = \{l_{k_{i,j}}, u_{k_{i,j}}, l_{d_{i,j}}, u_{d_{i,j}}, l_{\tilde{x}_{0_i}}, u_{\tilde{x}_{0_i}}, l_{\dot{\tilde{x}}_{0_i}}, u_{\dot{\tilde{x}}_{0_i}}, b_i, \Lambda\}$. Look into Appendix B for more details about each element of $\mathcal{S}$.

Algorithm 1 Control Algorithm for Soft Landing

```
Require: $\mathcal{S}, \mathcal{T}_d, \mathbf{X}^+$
Ensure: $\tau_{total}$
 1: repeat
 2:     if $\neg$Landed then
 3:         $\tau_{mc} \leftarrow$ SolveMPC($\mathcal{T}_d, \mathbf{X}^+$)
 4:     else
 5:         $(K_n, D_n) \leftarrow$ SolveOptimization($S$)
 6:         if StabilityPassivityCheck($\mathcal{S}, K_n, D_n$) then
 7:             $(K_{final}, D_{final}) \leftarrow$ FinalGains($K_n, D_n$)
 8:         else
 9:             $(K_n^*, D_n^*) \leftarrow$ EnforceStability($\mathcal{S}, K_n, D_n$)
10:             $(K_{final}, D_{final}) \leftarrow$ FinalGains($K_n^*, D_n^*$)
11:         end if
12:         $W_{com} \leftarrow$ SolveVIC($\mathcal{T}_d, \mathbf{X}^+$)
13:         $\tau_{mc} \leftarrow$ MapComTorquesToLegs($W_{com}$)
14:     end if
15:     $\tau_{PD} \leftarrow$ SolvePD($\mathcal{T}_d, \mathbf{X}^+$)
16:     $\tau_{n_c} \leftarrow$ SolveContactController($\mathbf{X}^+$)
17:     $\tau_{total} \leftarrow$ Sum($\tau_{mc}, \tau_{PD}, \tau_{n_c}$)
18: until StoppingCondition
```

Whereas $\mathcal{T}_d$ includes the desired CoM trajectories, desired joint positions q_d, and velocities $\dot{q}_d$. The algorithm's output, τ_{total}, is the aggregated control torque sent to the robot. Each iteration of the algorithm comprises the following steps:

1) Calculate the Main Controller joint torques, τ_{mc}, the MPC if the robot has not yet landed.
2) Upon landing, compute the damping D_n and stiffness K_n gains using Equation (18).
3) Validate the gains against stability criteria of inequality (21); if passed, assign D_n and K_n as the final gains, D_{final} and K_{final}.
4) In case of stability failure, assign to D_{final} the right-hand side of Equation (21) plus a small increment, and K_{final} can be recalculated using Equation (20).
5) Solve the impedance control problem to determine $W_{com} \in \mathbb{R}^6$, the wrench at the center of mass, as per the following equation:

$$W_{com} = K_{final}\tilde{x} + D_{final}\dot{\tilde{x}}. \tag{7}$$

6) The wrenches W_{com} computed for the CoM cannot be directly applied as control actions to the robot. Instead, they need to be translated from wrenches at the CoM level to joint torque actions $\tau_{mc} \in \mathbb{R}^{12}$. Before this conversion, the feasibility of W_{com} must be ensured. This is achieved by solving a Quadratic Programming (QP) optimization problem aimed at minimizing the deviation of the resultant wrench W from W_{com}, subject to several constraints:

$$\underset{F_k}{\text{argmin}} \quad \|W - W_{com}\|_Q^2, \tag{8a}$$

$$\text{s.t.} \quad W = [f_x, f_y, f_z, \tau_x, \tau_y, \tau_z], \tag{8b}$$

$$f_x = F_{k,1} + F_{k,4} + F_{k,7} + F_{k,10}, \tag{8c}$$

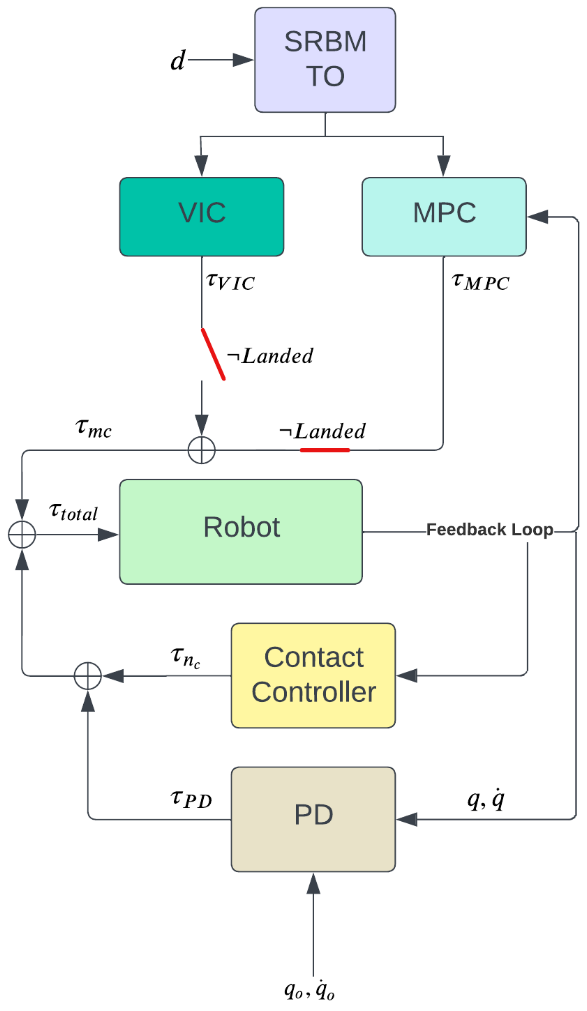


Fig. 5: This figure presents the overall framework obtained after improving the TO algorithm and introducing a variable impedance controller (VIC). From the graph, we see that the planner communicates the desired trajectories to both the MPC and the VIC. They both produced a control action as output but the algorithm considers only one of the two solutions, based on the state of the robot that can be *Landed* or $\neg$*Landed*, we call the resultant control action τ_{mc}, where mc stand for *main controller*.

$$f_y = F_{k,2} + F_{k,5} + F_{k,8} + F_{k,11}, \tag{8d}$$

$$f_z = F_{k,3} + F_{k,6} + F_{k,9} + F_{k,12}, \tag{8e}$$

$$\begin{bmatrix}\tau_x\\ \tau_y\\ \tau_z\end{bmatrix} = \sum_{i=1}^{4} \begin{bmatrix}F_{k,3i-2}\\ F_{k,3i-1}\\ F_{k,3i}\end{bmatrix} \times \begin{bmatrix}\hat{r}_{3i-2}\\ \hat{r}_{3i-1}\\ \hat{r}_{3i}\end{bmatrix}, \tag{8f}$$

$$-\mu F_z \leq F_x \leq \mu F_z, \tag{8g}$$

$$-\mu F_z \leq F_y \leq \mu F_z, \tag{8h}$$

$$f_z \geq 0, \tag{8i}$$

$$|J^T F_k| \leq \tau_{\text{max}}. \tag{8j}$$

Here, $F_k \in \mathbb{R}^{12}$ denotes the GRFs to be applied at the robot's feet. Specifically, the first three elements of F_k are the x, y, z components of the GRF on the FR foot, and similar for the other feet. The equations (8c), (8d), and (8e) sum up the GRFs in x, y, and z directions, respectively. Constraint (8f) defines the torques τ_x, τ_y, and τ_z around the CoM, caused by the GRFs on the feet, where $[\hat{r}_{3i-2}, \hat{r}_{3i-1}, \hat{r}_{3i}]^T$ represent the x, y, and z Cartesian distances between the i_{th} foot position and the CoM.
Constraints (8g), (8h), and (8i) represent the friction cone and non-negativity of GRFs in the z direction. Constraint (8j) ensures that the torques remain within the mechanical limits of the robot.

7) The optimized solution F_k from the QP problem is then converted to joint torques τ_{mc} using the foot Jacobian:

$$\tau_{mc} = J^T F_k. \tag{9}$$

8) Once τ_{mc} is computed, the PD control action is retrieved using the equation:

$$\tau_{PD} = P(q - q_o) + D(\dot{q} - \dot{q_o}). \tag{10}$$

Here, P and D are the proportional and derivative gains, respectively. The values q_o and $\dot{q}_o$ are references manually chosen to keep the robot in a standard configuration. It is important to note that τ_{PD} is modulated to reduce its effect during the initial phase of landing. This modulation is designed to prevent interference with the compliance level established by the VIC during the most critical moment of landing, specifically in the first few milliseconds.

9) For each foot not in contact with the ground, an additional vertical force, F_{push}, is applied. This is calculated to push the leg downwards, generating the contact torque τ_{n_c}:

$$\tau_{n_c} = J^T \begin{bmatrix}0\\ 0\\ F_{\text{push}}\end{bmatrix}.$$

10) The final control action τ_{total} for the current iteration is obtained by summing τ_{mc}, τ_{PD}, and τ_{n_c}. This aggregated torque is then used to control the robot.

D. Supervised Learning Algorithm Structure

In this paper's Introduction, we highlighted the imperative of executing all processes on-the-fly, specifically to avoid delays caused by trajectory optimization solvers. This is crucial for deploying responsive quadruped robots in real-world environments. Achieving on-the-fly results using model-based solutions often requires renouncing accuracy, but this would not satisfy the *research goal* of the work. The solution we found exploits the concept of *behavioral cloning* (BC). This methodology, as a component of imitation learning, serves as a straightforward method to imitate expert behavior by directly learning a mapping from observations to actions based on expert demonstrations [53].

Leveraging the existing model-based framework (refer to Fig. 5), which had already demonstrated desirable outcomes, we generated a synthetic dataset to train our neural network.

Following the methodology of Kurtz et al. [37], we developed two distinct neural network architectures, named *Feedforward NN* and *Feedback NN.*

Feedforward NN: The core concept of the Feedforward approach involves a single pre-jump prediction by the neural network, forecasting all necessary jump parameters. The input is a 35-element vector representing the robot's initial state:

$$\mathbf{X}_{FF} = [z_0, \phi_0, \theta_0, \psi_0, \dot{x}_0, \dot{y}_0, \dot{z}_0, \dot{\omega}_{x,0}, \dot{\omega}_{y,0}, \dot{\omega}_{z,0}, q_0, \dot{q}_0, d].$$

Here, FF denotes *"feedforward"*.

Conversely, the network's output comprises five predicted trajectories:

$$\mathcal{O}_{FF} = [\mathbf{x}_p, \mathbf{z}_p, \boldsymbol{\tau}_p, \mathbf{q}_p, \dot{\mathbf{q}}_p],$$

with p indicating predictions. These bolded terms represent the trajectories' evolution during the jump, rather than instantaneous values. The first two trajectories, i.e. $\mathbf{x}_p$ and $\mathbf{z}_p$, belongs to $\mathbb{R}^{150\times1}$ and the remaining to $\mathbb{R}^{150\times12}$. The total output size, combining these elements, is therefore 5700.

Before applying these predictions to the robot, it is essential to interpolate them to synchronize with the controller's frequency. This interpolation ensures that each trajectory, now transformed from its original vector space, aligns seamlessly with the real-time operational tempo of the robot's control system.

In $\mathcal{O}_{FF}$, the first two elements, $\mathbf{x}_p$ and $\mathbf{z}_p$, serve primarily to assess the predicted jump's quality. Smooth trajectories in these dimensions suggest a higher likelihood of a successful jump. This evaluation is especially useful in real robot tests, as it enables operators to halt a jump based on the instability of these graphs. Meanwhile, $\boldsymbol{\tau}_p$ replaces τ_{mc} in Algorithm (1), and $\mathbf{q}_p$ and $\dot{\mathbf{q}}_p$ facilitate the computation of τ_{PD} as per equation (10). The *Feedforward NN*'s architecture is depicted in Fig. 6.

Feedback NN: The second architecture we designed is the *Feedback NN*, which more closely resembles a traditional feedback control system. In this model, the neural network computes torque actions at each iteration, based on the current state of the robot. The input to this network is a 36-element vector, detailed as follows:

$$\mathbf{X}_{FB} = [z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z, q, \dot{q}, d, t]. \tag{11}$$

Here, FB signifies 'feedback', and t represents the elapsed time since the start of the motion.

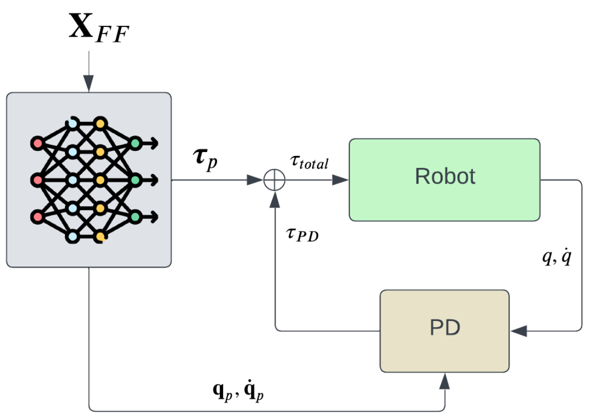


Fig. 6: Control scheme using a *Feedforward NN*. Note that the NN outputs belong to $\mathbb{R}^{150\times12}$ and cannot be directly summed with feedback loop quantities. Instead, the vectors $\boldsymbol{\tau}_p, \mathbf{q}_p$, and $\dot{\mathbf{q}}_p$ are interpolated to match the control loop frequency, with corresponding elements selected at each iteration.

The output of this network is defined in the vector space $\mathbb{R}^{12}$:

$$\mathcal{O}_{FB} = \tau_p,$$

where τ_p represents the predicted torque, which can be directly applied as a control action to the robot, as depicted in Figure 7.

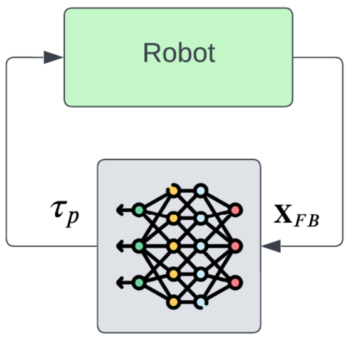


Fig. 7: A simplified representation of the *Feedback NN*'s operational principle.

The inherent design of each network type confers distinct operational characteristics. The *Feedback NN* excels in dynamic adaptability; its embedded feedback mechanism enables robust control even under mid-jump disturbances. Furthermore, the *Feedback NN* is characterized by its limited output, consisting of merely 12 distinct functions to learn. This limited output scope enhances the precision and reliability of its learning process.

In contrast, the *Feedforward NN* is challenged by its considerably larger output range. This 5700-dimensional output inherently increases the likelihood of inaccuracies in learning individual functions. The possibility of even a single imprecise output can potentially lead to operational failures. Acknowledging this challenge, it's important to note that the *Feedforward NN* is also defined by its trajectory determinism; once a jumping trajectory is initiated, it cannot be altered, which limits the system's adaptability to in-flight disturbances or environmental changes.

However, this very limitation of the *Feedforward NN* is somewhat mitigated by its reduced reliance on extensive data. Unlike the *Feedback NN*, which requires a comprehensive learning base encompassing a wide array of potential in-jump states, the *Feedforward NN* focuses on a more restricted set of initial states. The structure of this network not only eases the data requirement but also enhances the interpretability of the network. Specifically, the trajectories generated by the *Feedforward NN* can be scrutinized for critical issues, such as excessive joint torques or self-collision risks, before their execution.

In the context of supervised learning, especially for neural network architectures, the quality and volume of training data are paramount. For this project, we employed the *pybullet* simulation engine to generate a diverse dataset, comprising approximately $11,000$ simulated jumps under varying conditions. These jumps were executed by the robot using the framework outlined in Fig. 5. Specifically, the robot was programmed to jump distances ranging from $0.10m$ to $0.55m$, in increments of $0.01m$. This approach ensured that each discrete distance was represented equally in the dataset, providing a comprehensive basis for training the algorithm. The selection of the jumping range warrants an explanation. The minimum jump length of $0.1m$ was chosen as the threshold below which a forward jump would be meaningless. Conversely, the maximum distance of $0.55m$ was dictated by the limitations of the SRBM TO's design, which is optimized for shorter jumps. As this research focuses not on long-distance jumping, extending beyond this range was deemed beyond the study's scope.

A critical aspect of data collection is ensuring a thorough exploration of the robot's state space. To this end, we introduced three types of noise into the simulation:

- Gaussian noise on the initial configuration ($\mathbf{X}_0^+$), facilitating a broad range of starting positions akin to those the real robot might encounter.
- Gaussian noise on the state readings ($\mathbf{X}^+$), to explore the neighborhood of the trajectory and promote resilience to measurement errors.
- An external disturbance force applied to the robot's CoM, designed to test the robot's trajectory stability and balance recovery capabilities.

The disturbance force, equating to 10% of the robot's mass, was applied with a 30% probability at each control iteration, with its direction randomized. This methodology not only broadened the trajectory data domain but also contributed to the robustness of the resulting neural network by including scenarios where the robot must adjust to unexpected forces.

Out of the $11,000$ jump simulations, one-third incorporated all three types of noise. The remaining two-thirds included only the first two noise types, thus ensuring a balanced and comprehensive dataset for the training of our behavior cloning algorithm.

V. Experimental Validation

In this section, we present the empirical assessment of the methods proposed in the earlier sections of this thesis.

The goal is to validate the theoretical models and simulations through concrete data, bridging the gap between theory and application in robotic systems.

A. SRBM vs. SLIP Trajectory Optimization Algorithm

In the methodology section, we introduced the SRBM trajectory optimization problem to generate more feasible trajectories for robotic jumps. A solid method to assess trajectory feasibility is to evaluate the tracking error, representing the discrepancy between the desired and actual trajectories. Under a constant controller setting, a reduction in this error indicates a more realistic and achievable trajectory for the robot.

To quantify the tracking error, we employed the mean square error (MSE), calculated as:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(P_i - P_i^{\text{TO}})^2. \tag{12}$$

Here, P_i and P_i^{TO} denote the position values at point i of the actual and the predicted trajectories, respectively. Accordingly to the model used in the trajectory optimization, P_i^{TO} could become P_i^{SLIP} or P_i^{SRBM}. The robot was tasked to jump distances ranging from $0.1m$ to $0.55m$ in increments of $0.05m$. Each distance was attempted 20 times (starting each time from a slightly different initial configuration) to calculate the mean and standard deviation of the results.

As Figure 8 demonstrates, the SRBM TO consistently outperforms SLIP TO across various metrics, aligning with our hypothesis that a refined model yields improved movement predictions.

B. Soft Landing - MPC vs. VIC

The primary focus of the second contribution in this work is addressing the challenge of achieving a soft landing in quadruped robot jumps. As detailed in the methodology section, we enhance the existing framework by replacing the MPC with a Variable Impedance Controller (VIC) at the onset of the landing phase. A successful soft landing is distinguished by two key features: minimal stress exerted on the robot's motors and a compliant response in the robot's body and legs.

To perform a comparative analysis between the MPC and the VIC, we employed multiple metrics, each examining the robot's performance in jumping distances ranging from $0.1m$ to $0.55m$, increasing in $0.05m$ increments. Following the approach adopted to create Fig. 8, for each distance, the robot completed 20 trials to facilitate the calculation of mean values and standard deviations. When evaluating the MPC, we adhered to the scheme outlined in Figure 5, while keeping the VIC's switch perpetually open. Conversely, in the VIC assessment, we utilized the same framework but allowed for the controller's normal operation, activating the switch when the robot's four feet made contact with the ground. As a reference, we conducted the same experiments on the *starting framework* of Figure 3, called "*baseline*" in the graphs.

For the evaluation of motor effort, we devised two metrics: *rotational effort* and *peak torque*. The *rotational effort* is defined as:

$$\text{Rotational Effort} = \sum_{i=1}^{12}\int |\tau_i(t)|\, dt. \tag{13}$$

This metric quantifies the cumulative effort exerted by all motors during landing, with lower *rotational effort* values indicating a softer landing. As depicted in Figure 9, the VIC generally demonstrates reduced *rotational effort* across various jump distances, except the $0.50m$ distance.

While the *rotational effort* provides insights into the overall load on the motors, we also focus on the *peak effort* required by these motors. High *rotational effort* might accelerate wear in motor components, whereas *peak effort* is closely associated with the motors' capacity to handle sudden forces.

The *peak effort* is defined as:

$$\text{Pick Effort} = \max_i\left(\sqrt{\sum_{j=1}^{12}\tau_{ij}^2}\right). \tag{14}$$

Here, i represents the time instance, and j denotes the motor. This equation calculates the norm of the torques at each time instant and then identifies the maximum value across the time instances.

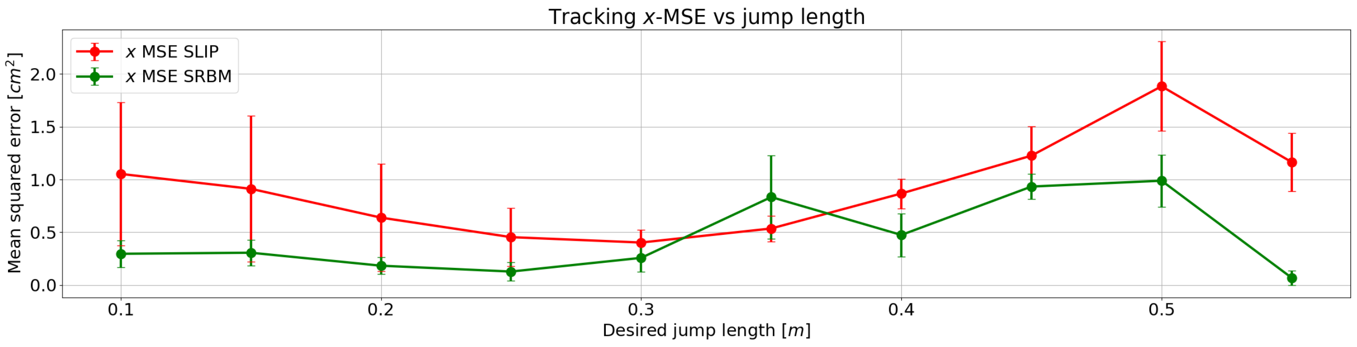
From the graph in Figure 10 we can see how the VIC outperforms the MPC.

The *rotational effort* and *peak effort* effectively gauge motor stress; however, they do not fully capture the compliance aspect of a soft landing. To assess this, we examined the acceleration of the center of mass during landing. A compliant robot is characterized by its ability to adapt to external forces, which can be quantitatively inferred from the CoM acceleration. Continuing with our experimental methodology, we collected data summarized in Figure 11. This figure includes three graphs showing the maximum CoM acceleration along the x, y, and z axes. The VIC consistently outperforms the MPC in reducing z axis acceleration across all jumping distances. For the x axis, a similar trend is observed except at a $0.55m$ jumping distance. The y axis results are less uniform, but the VIC generally surpasses the MPC in performance.

C. Feedback vs. Feedforward NN

In the methodology section, we introduced two neural network architectures: *Feedback* and *Feedforward*. Extensive testing led to the optimal *Feedback NN* configuration of two 1024-neuron layers with *ReLU* activation function, while the *Feedforward NN* performed best with two 128-neuron layers using *eLU* activation function.

Comparative analysis showed the *Feedback NN* as superior for our task. The landing accuracy of the two solutions was tested by performing 100 jumps of random lengths to compute the mean and standard deviation of the landing error for x and z position of the CoM. The results are reported in Table II inside Appendix C. At the same time, the *Feedforward NN*



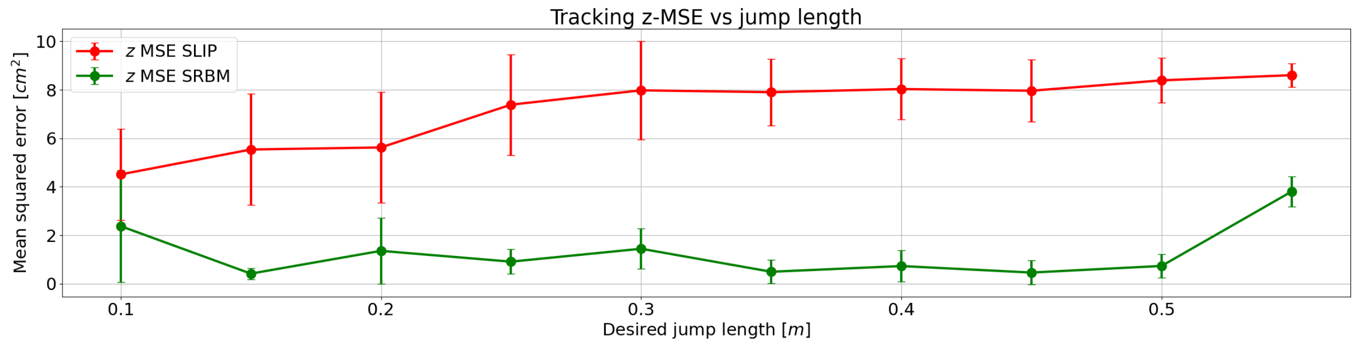


Fig. 8: The graph illustrates the MSE comparison between SRBM TO (in green) and SLIP TO (in red), focusing on the x and z positions of the Center of Mass (CoM). The SRBM TO consistently exhibits lower MSE values and standard deviations, indicating more reliable performance, except the x-MSE at $0.35m$. The y-MSE is not reported, as this study concentrates on forward jumps.

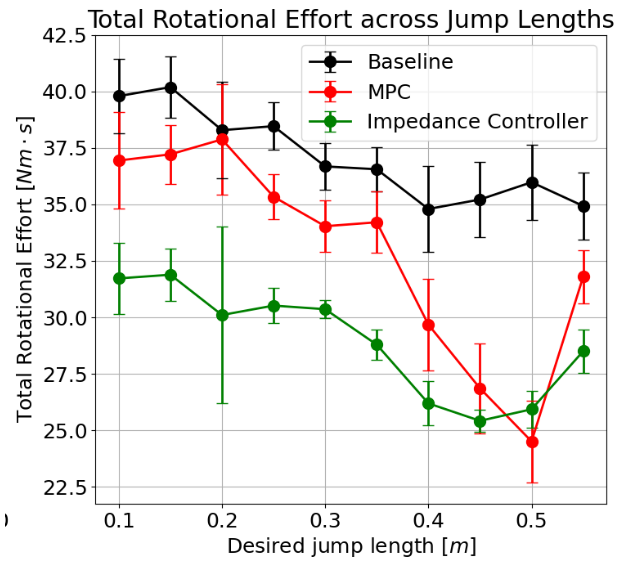


Fig. 9: Comparative analysis of *rotational effort* between the MPC (in red) and the VIC (in green), showing the VIC's overall reduced demand on the robot's 12 motors. In black, as a reference, the data related to the "starting framework" of subsection IV-A.

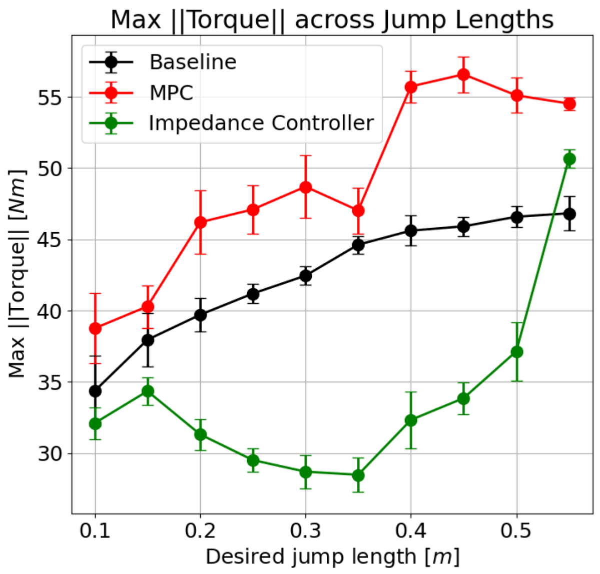


Fig. 10: Comparative analysis of *peak effort* between the MPC (in red) and the VIC (in green), demonstrating the VIC's lower instantaneous torque demands on the robot. In black, as a reference, the data related to the "starting framework" of subsection IV-A.

showed worse performances in cloning the behavior of the model-based controller as shown by the plots in Appendix C.

Due to the demonstrated superiority of the *Feedback NN*, the subsequent section will analyze in detail the performance of this architecture.

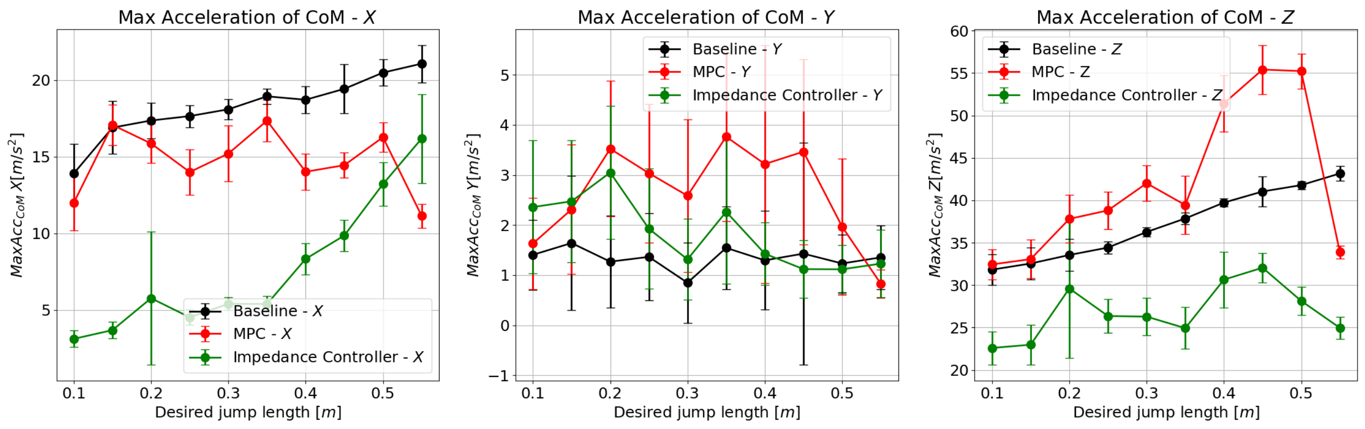


Fig. 11: Three graphs depicting CoM accelerations during landing: in green, the VIC controller's performance, and in red, the MPC. Lower acceleration values indicate higher compliance. In black, as a reference, the data related to the "starting framework" of subsection IV-A.

D. *Feedback NN Performance*

In the previous paragraphs, we reported the performances of the new TO algorithm and landing controller. But, as mentioned in the methodology section, the idea of this work is to use the model-based solution as the source to create a synthetic dataset on which we train a neural network. This method's advantage is to eliminate the computational overhead of an accurate trajectory optimization algorithm while maintaining the performance. The following graphs will be used to support this statement.

The first step we take is plotting again the metrics of Subsection V-B adding to the graphs the data obtained while controlling the robot with the *Feedback NN* (to keep the graph readable we removed the *baseline* data).

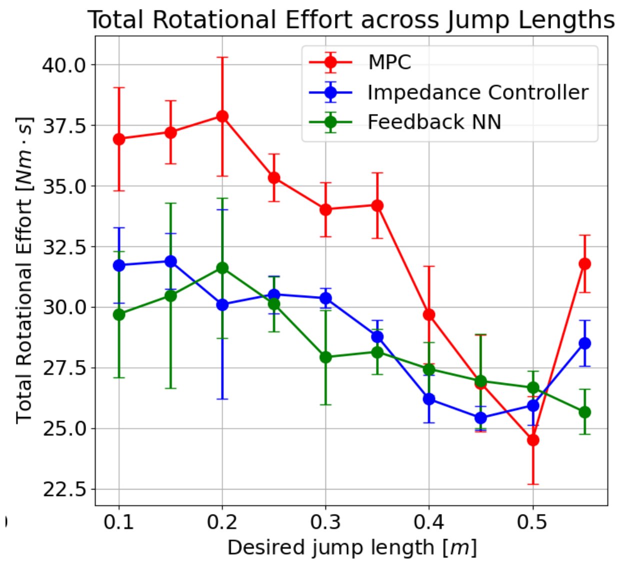


Fig. 12: Comparative analysis of *rotational effort* between the MPC (in red), the VIC (in blue), and the *Feedback NN* (in green), showing the NN's overall reduced demand on the robot's 12 motors similarly to the VIC.

Looking at Figures 12, 13, and 14 we can see that the NN produces results that map the behavior of the VIC, with

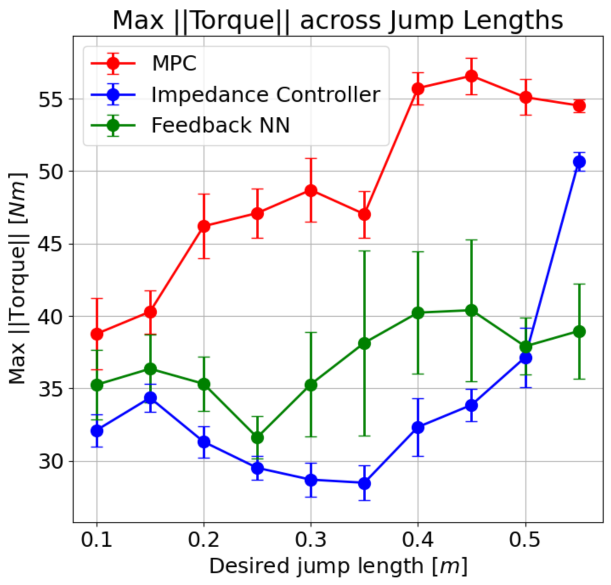


Fig. 13: Comparative analysis of *peak effort* between the MPC (in red), the VIC (in blue), and the *Feedback NN* (in green), demonstrating the NN's ability to preserve the qualities of the VIC.

similar values.

The second evaluation to conduct is related to the solving time of the TO plus MPC/VIC solution versus the *Feedback NN*. To do so we calculated the time to solve the trajectory optimization and then the time to solve the MPC/VIC at each iteration summing these quantities together. For the neural network, we summed up all the prediction times. We repeated this process ten times to extract the mean and standard deviation at each jump length. The results are reported in Figure 15, showing a significant difference in favor of the model-free method. Note that in the results of Figure 15, the TO is implemented in C++ using *Opti* framework for *CasADi*, the MPC is developed in C++, and the NN in Python.

It is imperative to compare also the solving time of the MPC only with the prediction time of the NN. This comparison is crucial as the control frequency of the real robot is contingent

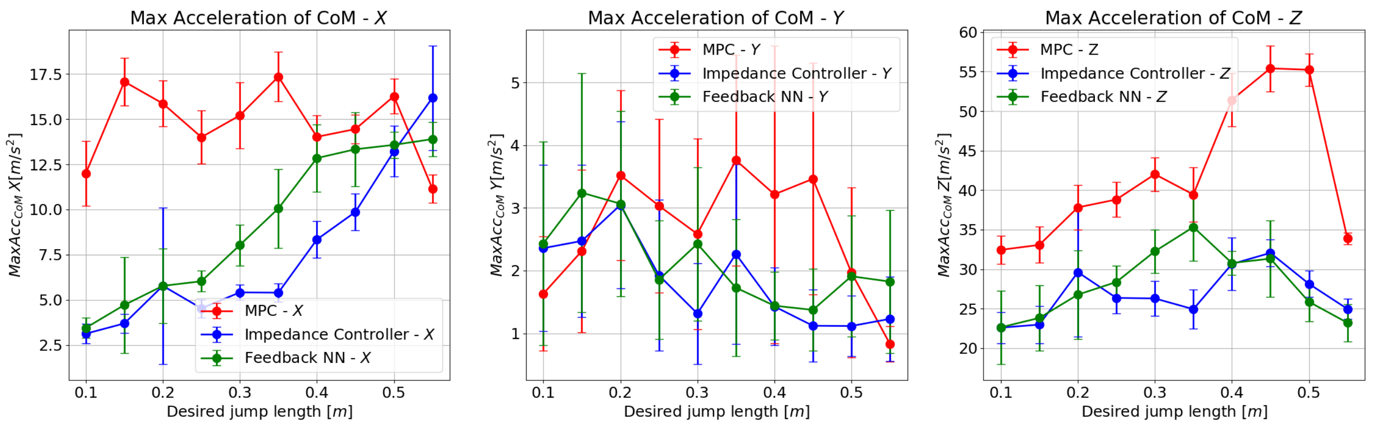


Fig. 14: Three graphs depicting CoM accelerations during landing: in blue, the VIC controller's performance, in red the MPC, and in green the *Feedback NN*. Lower acceleration values indicate higher compliance. The graph shows how the NN can reproduce the performances of the model-based solution.

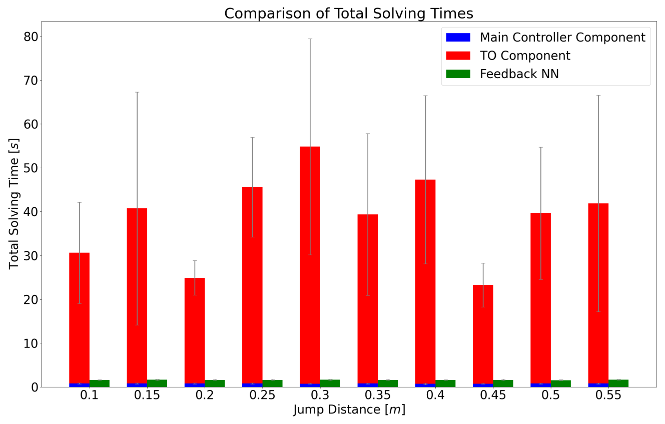


Fig. 15: The bar chart displays a comparison between model-based and model-free approaches to solving times across different jump distances. Red bars show trajectory optimization times, and blue bars represent the main controller time, which includes MPC and then VIC during landing phases. Small green bars adjacent to these indicate the *Feedback NN*'s cumulative prediction time in the behavioral cloning approach.

on these values. To conduct this analysis we implemented also the *Feedback NN* using C++. Table I compares mean, standard deviation, and maximum solving/prediction time computed over 100 random jumps, revealing that the NN substantially outperforms the MPC by at least an order of magnitude in each metric.

Metrics	**MPC**	**Feedback NN**
Mean	951 μs	78 μs
Std	139 μs	3 μs
Max	5325 μs	389 μs

TABLE I: MPC vs. Feedback NN: The NN outperforms MPC by over ten times in all metrics.

This superiority allows for a higher control frequency when implemented on the robot.

The alleviation of the computational demands associated with trajectory optimization enables the quadruped robot to execute continuous jumps without the necessity of pre-computation. This capability is of significant utility in dynamic environments where the distances of subsequent jumps cannot be predetermined. The efficacy of this approach has been validated through simulations, as illustrated in Figure 4.

Lastly, it is crucial to address the *success rate* of the *Feedback NN*. While previous graphs have showcased the network's performance under successful conditions, it is equally important to assess its reliability. Figure 16 presents a heatmap illustrating that the NN's success rate is approximately 97.4%. In the picture the x-axis represents the jump length, the y-axis denotes attempts per length, and the color bar reflects the landing error based on the robot's final x position of the CoM. Notably, the robot exhibits optimal performance in the middle of the jumping range, where landing errors are minimal.

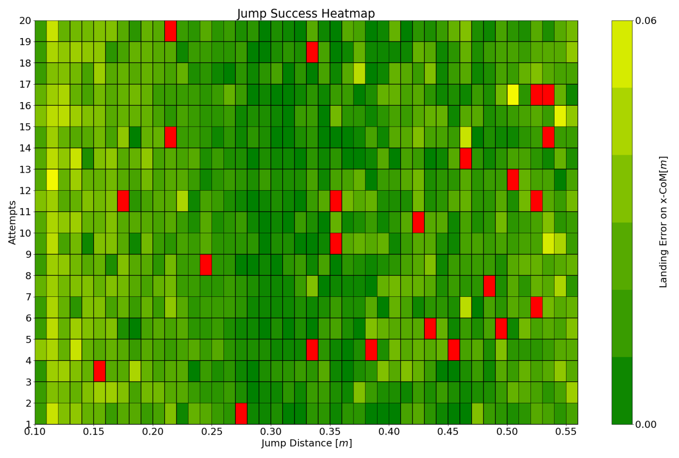


Fig. 16: The heatmap depicting the jumping performances of the *Feedback NN*. The graph illustrates the relationship between the jump length, the number of attempts per length, and the landing error calculated at the final x position of the robot's CoM.

E. Hardware Results

Prior sections of this research utilized simulations to explore the efficacy of behavioral cloning in facilitating robotic jumps. A pivotal aspect of this study involves validating these findings on actual hardware. Among the neural network architectures examined in Section IV-D, the *Feedback NN* was found superior in performance. Conversely, the *Feedforward NN*, while not as effective in simulations, offers essential benefits for hardware applications, particularly in the interpretability of its outputs. This feature is vital for the pre-assessment of joint movements and center of mass (CoM) trajectories, crucial for ensuring safety during operations.

Given the essential focus on safety and the upcoming paper submission deadline, the study proceeded with the *Feedforward NN* for empirical validation of the behavioral cloning approach, despite its comparative under-performance in simulations relative to the *Feedback NN*. To adapt the *Feedforward NN* for hardware execution, two modifications were implemented:

- Introduction of a low-pass filter on the predicted torques to mitigate jerky movements.
- Network weight refinement using an additional dataset comprising 3000 jumps, aimed at addressing discrepancies between simulation and physical execution.

The necessity for fine-tuning arose from the realization that the robot's initial stance in the synthetic dataset differed markedly from its actual starting position, leading to unsuccessful jump executions. To rectify this, 25 actual starting positions were recorded and used to synthesize a new dataset for training. Besides this difference, the refinement dataset has been collected following the approach described in section IV-D. Following these adjustments, the robot was tested with jumps ranging between $0.1m$ and $0.5m$, as depicted in Figure 1 and 17.

VI. Discussion & Conclusion

This work shows the potential of merging model-based with model-free techniques. Often these two techniques show complementary strong points, and combining them is a good way to exploit the best of both worlds. In particular, we started improving the *trajectory optimization* algorithm, introducing the single rigid body model in the dynamics equations. We proved how using a more accurate model in the TO leads to more feasible trajectories, hence lower tracking errors. This improvement comes at the expense of a longer computational time. Building on this improved framework, we introduced an innovative *variable impedance controller* during the landing phase, to automatically tune the stiffness and damping of the robot and achieve a compliant solution without allowing big tracking errors. We then proved that a VIC outperforms a traditional MPC when it comes to soft landing, not only reducing the accelerations on the CoM but also reducing the stress on the motors.

Once proved the superiority of the model-based solution compared to the starting framework, we solved the computational burden generated by the TO by training a behavioral cloning algorithm. This algorithm aimed to generate a neural network capable of doing the job of the TO and the controller at the same time. We structured it in the form of a *Feedback NN* and a *Feedforward NN*. To obtain reliable results, we collected a synthetic dataset of $11,000$ jumps ranging from $0.10m$ to $0.55m$ with several types of noise, which allowed us to explore the state space as much as possible and obtain a NN able to generalize when faced with unforeseen scenarios. The *Feedback NN*, featuring two hidden layers of 1024 neurons each, demonstrated superior performance in simulations, achieving a remarkable average landing precision of $0.02m$ and a success rate of 97.4%, effectively mimicking the VIC's soft landing capabilities.

We then proceeded to validate our approach on hardware, but this time using the *Feedforward NN* for safety and timing reasons. The real-world tests proved the feasibility of the approach showing good sim-to-real transferability.

The results of the *Feedback NN* demonstrate how it is possible to have advanced results by exploiting behavioral cloning techniques, usually limited to complex model-based solutions [10][11][12], without renouncing on-the-fly performances. When compared with reinforcement learning [23][24][25][26][27], relying on a BC inspired by a model-based solution is a straightforward way to define one's desired goal, rather than crafting a complex reward function.

Looking at the main limitation of this work, we recommend extending this research to include hardware implementation of the *Feedback NN* and direct comparison with model-based results. Another possible shortcoming of this study lies in the jumping range achievable, an interesting follow-up paper may be to investigate a reformulation of the trajectory optimization to extend the maximum jumping distance.

Since the literature, up to the author's knowledge, is very scarce about BC solutions to jump, future compelling works may include applying similar techniques to obtain omnidirectional and obstacle-aware jumps. Some effort could go into exploring network architectures and dataset compositions to achieve a nearly perfect success rate. Another possibility is improving the VIC, in particular trying to implement the algorithm without the assumption of a diagonally dominant inertia matrix. Avoiding this simplification allows us to find the values of the impedance controller that updates with more accuracy based on the configuration of the robot. In summary, our study marks a notable advancement in robotic locomotion, highlighting the benefits of merging model-based and model-free strategies using behavioral cloning. This paper introduces a versatile method applicable across various locomotion tasks, laying a foundation for broad applications in the field.

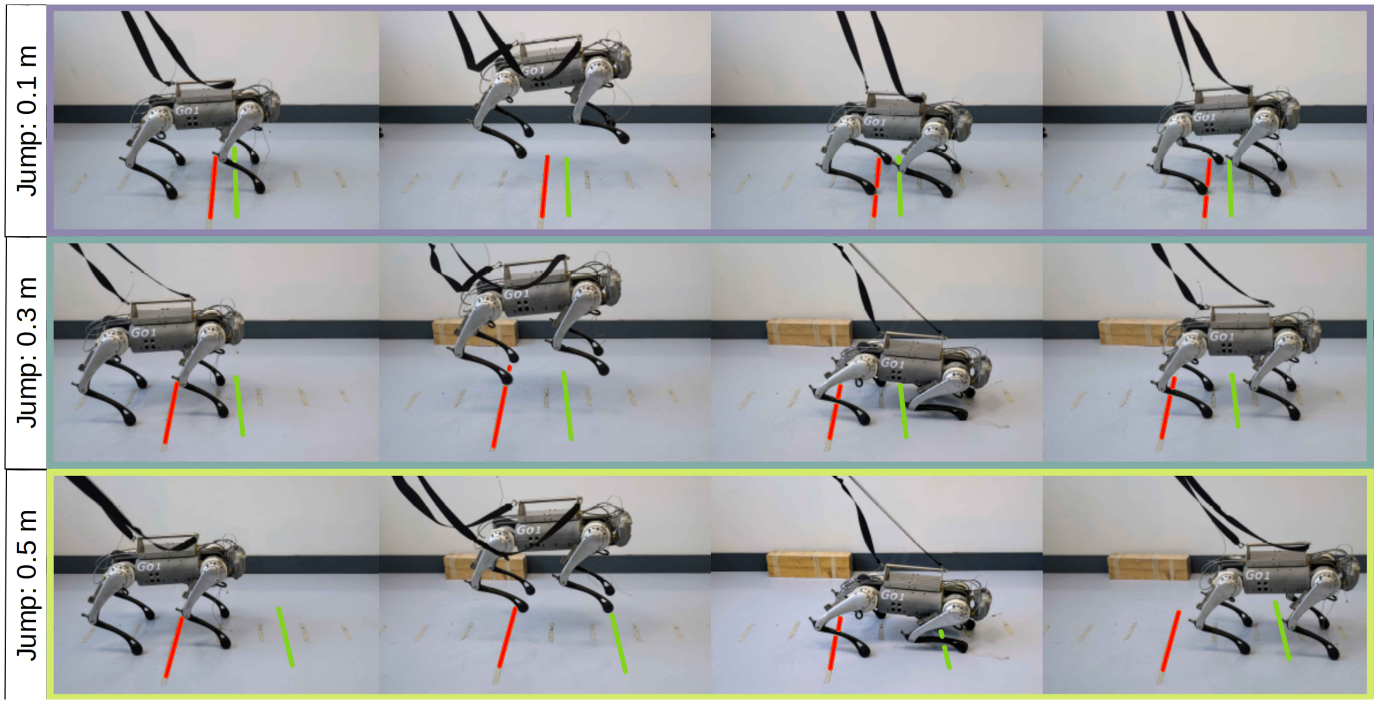


Fig. 17: Each row of this picture illustrates a jump of a different length. For each one four steps are reported: the *starting* position, the *highest* point of the trajectory, the *lowest* point of the landing, and the *final* position. In each picture, the red line indicates the initial position of the CoM, and the green one is the desired end position.

References

[1] V. Hugel and P. Blazevic, "Towards efficient implementation of quadruped gaits with duty factor of 0.75," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 3, Detroit, MI, USA: IEEE, 1999, pp. 2360–2365.

[2] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," in *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA: IEEE, May 2008, pp. 811–818.

[3] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Fast, robust quadruped locomotion over challenging terrain," in *2010 IEEE International Conference on Robotics and Automation*, Anchorage, AK: IEEE, May 2010, pp. 2665–2670.

[4] G. K. K. and P. M. Pathak, "Dynamic modelling & simulation of a four legged jumping robot with compliant legs," *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 221–228, Mar. 2013.

[5] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2261–2268, Jul. 2018.

[6] H.-W. Park, P. M. Wensing, and S. Kim, "Online planning for autonomous running jumps over obstacles in high-speed quadrupeds," *Park*, 2015.

[7] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," in *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, ser. Mechanisms and Machine Science, G. Carbone and F. Gomez-Bravo, Eds., Cham: Springer International Publishing, 2015, pp. 3–27.

[8] S. H. Jeon, S. Kim, and D. Kim, "Online optimal landing control of the MIT mini cheetah," in *2022 International Conference on Robotics and Automation (ICRA)*, May 2022, pp. 178–184.

[9] M. J. Pollayil, F. Angelini, G. Xin, *et al.*, "Choosing stiffness and damping for optimal impedance planning," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1281–1300, Apr. 2023.

[10] S. Gilroy, D. Lau, L. Yang, *et al.*, "Autonomous navigation for quadrupedal robots with optimized jumping through constrained obstacles," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, Lyon, France: IEEE, Aug. 23, 2021, pp. 2132–2139.

[11] C. Nguyen, L. Bao, and Q. Nguyen, *Continuous jumping for legged robots on stepping stones via trajectory optimization and model predictive control*, Sep. 16, 2022.

[12] M. Chignoli, S. Morozov, and S. Kim, "Rapid and reliable quadruped motion planning with omnidirectional jumping," in *2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA: IEEE, May 23, 2022, pp. 6621–6627.

[13] Q. Nguyen, M. J. Powell, B. Katz, J. D. Carlo, and S. Kim, "Optimized jumping on the MIT cheetah 3 robot," in *2019 International Conference on Robotics

and Automation (ICRA), Montreal, QC, Canada: IEEE, May 2019, pp. 7448–7454.

[14] C. Nguyen and Q. Nguyen, "Contact-timing and trajectory optimization for 3d jumping on quadruped robots," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan: IEEE, Oct. 23, 2022, pp. 11 994–11 999.

[15] M. Chignoli and S. Kim, "Online trajectory optimization for dynamic aerial motions of a quadruped robot," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China: IEEE, May 30, 2021, pp. 7693–7699.

[16] C. Mastalli, W. Merkt, G. Xin, *et al.*, *Agile maneuvers in legged robots: A predictive control approach*, Jul. 18, 2022.

[17] S. H. Jeon, S. Kim, and D. Kim, *Real-time optimal landing control of the MIT mini cheetah*, Oct. 6, 2021.

[18] N. Heess, D. TB, S. Sriram, *et al.*, *Emergence of locomotion behaviours in rich environments*, Jul. 10, 2017.

[19] J. Hwangbo, J. Lee, A. Dosovitskiy, *et al.*, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, Jan. 16, 2019.

[20] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "DeepGait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, Apr. 2020.

[21] A. Kumar, Z. Fu, D. Pathak, and J. Malik, *RMA: Rapid motor adaptation for legged robots*, Jul. 8, 2021.

[22] N. Rudin, H. Kolvenbach, V. Tsounis, and M. Hutter, "Cat-like jumping and landing of legged robots in low gravity using deep reinforcement learning," *IEEE Transactions on Robotics*, vol. 38, no. 1, pp. 317–328, Feb. 2022.

[23] M. Bogdanovic, M. Khadiv, and L. Righetti, "Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization," *Frontiers in Robotics and AI*, vol. 9, 2022.

[24] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, *Robust and versatile bipedal jumping control through reinforcement learning*, May 31, 2023.

[25] X. Huang, Z. Li, Y. Xiang, *et al.*, "Creating a dynamic quadrupedal robotic goalkeeper with reinforcement learning," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Detroit, MI, USA: IEEE, Oct. 1, 2023, pp. 2715–2722.

[26] G. Bellegarda, C. Nguyen, and Q. Nguyen, *Robust quadruped jumping via deep reinforcement learning*, Aug. 11, 2023.

[27] J. Qi, H. Gao, H. Su, *et al.*, "Reinforcement learning-based stable jump control method for asteroid-exploration quadruped robots," *Aerospace Science and Technology*, vol. 142, p. 108 689, Nov. 2023.

[28] S. Schaal, "Learning from demonstration," in *Advances in Neural Information Processing Systems*, vol. 9, MIT Press, 1996.

[29] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," 2020.

[30] Q. Yao, J. Wang, S. Yang, *et al.*, "Imitation and adaptation based on consistency: A quadruped robot imitates animals from videos using deep reinforcement learning," in *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Jinghong, China: IEEE, Dec. 5, 2022, pp. 1414–1419.

[31] X. B. Peng, P. Abbeel, S. Levine, and M. Van De Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–14, Aug. 31, 2018.

[32] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning agile skills via adversarial imitation of rough partial demonstrations," in *Proceedings of The 6th Conference on Robot Learning*, PMLR, Mar. 6, 2023, pp. 342–352.

[33] Y. Fuchioka, Z. Xie, and M. Van De Panne, "OPT-mimic: Imitation of optimized trajectories for dynamic quadruped behaviors," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, London, United Kingdom: IEEE, May 29, 2023, pp. 5092–5098.

[34] D. A. Pomerleau, "Knowledge-based training of artificial neural networks for autonomous robot driving," in *Robot Learning*, J. H. Connell and S. Mahadevan, Eds., Boston, MA: Springer US, 1993, pp. 19–43.

[35] J. Choi, H. Kim, Y. Son, C.-W. Park, and J. H. Park, "Robotic behavioral cloning through task building," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, Korea (South): IEEE, Oct. 21, 2020, pp. 1279–1281.

[36] Q. Wang, Z. He, J. Zou, H. Shi, and K.-S. Hwang, "Behavior cloning and replay of humanoid robot via a depth camera," *Mathematics*, vol. 11, no. 3, p. 678, Jan. 29, 2023.

[37] V. Kurtz, H. Li, P. M. Wensing, and H. Lin, "Mini cheetah, the falling cat: A case study in machine learning and trajectory optimization for robot acrobatics," in *2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA: IEEE, May 23, 2022, pp. 4635–4641.

[38] Z. Hu, K. Wan, X. Gao, and Y. Zhai, "A dynamic adjusting reward function method for deep reinforcement learning with adjustable parameters," *Mathematical Problems in Engineering*, vol. 2019, pp. 1–10, Nov. 23, 2019.

[39] J. Zhang, M. Li, J. Cao, Y. Dou, and X. Xiong, "Research on bionic jumping and soft landing of single leg system in quadruped robot," *Journal of Bionic Engineering*, vol. 20, no. 5, pp. 2088–2107, Sep. 2023.

[40] J. Qi, H. Gao, H. Yu, M. Huo, W. Feng, and Z. Deng, "Integrated attitude and landing control for quadruped robots in asteroid landing mission scenarios using reinforcement learning," *Acta Astronautica*, vol. 204, pp. 599–610, Mar. 2023.

[41] J. Ding, V. Atanassov, E. Panichi, J. Kober, and C. della Santina, *Robust quadrupedal jumping with impact-aware landing: Exploiting parallel elasticity*, Dec. 10, 2023.

[42] A. Spröwitz, A. Tuleu, M. Vespignani, M. Ajallooeian, E. Badri, and A. J. Ijspeert, "Towards dynamic trot gait locomotion: Design, control, and experiments with cheetah-cub, a compliant quadruped robot," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 932–950, Jul. 2013.

[43] R. Sato, I. Miyamoto, K. Sato, A. Ming, and M. Shimojo, "Development of robot legs inspired by bi-articular muscle-tendon complex of cats," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany: IEEE, Sep. 2015, pp. 1552–1557.

[44] L. Wang, F. Meng, H. Liu, *et al.*, "Design and implementation of jumping robot with multi-springs based on the coupling of polyarticular," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia: IEEE, Dec. 2018, pp. 287–292.

[45] E. Kazama, R. Sato, I. Miyamoto, A. Ming, and M. Shimojo, "Development of a small quadruped robot with bi-articular muscle-tendon complex," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Zhuhai: IEEE, Dec. 2015, pp. 1059–1064.

[46] M. Ernst, H. Geyer, and R. Blickhan, "SPRING-LEGGED LOCOMOTION ON UNEVEN GROUND: A CONTROL APPROACH TO KEEP THE RUNNING SPEED CONSTANT," in *Mobile Robotics*, Istanbul, Turkey: WORLD SCIENTIFIC, Aug. 2009, pp. 639–644.

[47] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the MIT cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid: IEEE, Oct. 2018, pp. 1–9.

[48] N. Hogan, "Impedance control: An approach to manipulation," in *1984 American Control Conference*, San Diego, CA, USA: IEEE, Jul. 1984, pp. 304–313.

[49] A. Albu-Schaffer, C. Ott, U. Frese, and G. Hirzinger, "Cartesian impedance control of redundant robots: Recent results with the DLR-light-weight-arms," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, Taipei, Taiwan: IEEE, 2003, pp. 3704–3709.

[50] F. Ficuciello, A. Romano, L. Villani, and B. Siciliano, "Cartesian impedance control of redundant manipulators for human-robot co-manipulation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, USA: IEEE, Sep. 2014, pp. 2120–2125.

[51] A. Ghorbanpour and H. Richter, "Energy-optimal impedance control of cooperative robot manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 144, no. 12, p. 121 002, Dec. 1, 2022.

[52] F. Angelini, G. Xin, W. J. Wolfslag, *et al.*, "Online optimal impedance planning for legged robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China: IEEE, Nov. 2019, pp. 6028–6035.

[53] A. O. Ly and M. Akhloufi, "Learning to drive by imitation: An overview of deep behavior cloning methods," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 195–209, Jun. 2021.

APPENDIX A
TO OF THE STARTING FRAMEWORK

We express the problem formulation using the following set of equations:

$$\begin{aligned}
&\underset{\mathbf{x},\mathbf{u},\mathbf{dt}}{\operatorname{argmin}} && J_{\text{cost}}, && \textit{Cost Function} && (15a)\\
&\text{s.t.} && \textbf{Kinematic constraints:} && && \\
& && \mathbf{x}(0) = \mathbf{x}_0, && \textit{Initial condintions} && (15b)\\
& && \dot{\mathbf{x}}(0) = 0, && && (15c)\\
& && \mathbf{x}(N) = \mathbf{x}_f, && \textit{Final condintions} && (15d)\\
& && \dot{\mathbf{x}}(N) = 0, && && (15e)\\
& && \mathbf{x}_z(N_s-1) = \mathbf{x}_{z,\text{takeoff}}, && \textit{Takeoff condintions} && (15f)\\
& && 0 \le \mathbf{dt} \le \mathbf{dt}_{max}, && \textit{Step-size} && (15g)\\
& && \textbf{Dynamics constraints:} && && \\
& && \forall k \in [1,2,\ldots,N_s+N_f]: && && \\
& && \mathbf{x}(k+1) = Taylor(\cdot(k)), && \textit{Discrete Dynamics} && (15h)\\
& && \forall k \le N_s: && && \\
& && \mathbf{x}_x(k) \in conv[\mathbf{p}_{\text{feet}}], && \textit{Support Polygon} && (15i)\\
& && \ddot{\mathbf{x}}(k) = \frac{\mathbf{F}_s(\mathbf{x}(k))}{m} + \mathbf{g} + \mathbf{u}, && \textit{Stance Dynamics} && (15j)\\
& && \mathbf{u}_z + \frac{\mathbf{F}_{s.z}(\mathbf{x}(k))}{m} \ge 0, && \textit{Gravity Constraint} && (15k)\\
& && \forall k > N_s: && && \\
& && \ddot{\mathbf{x}}(k) = \mathbf{g}, && \textit{Flight Dynamics} && (15l)
\end{aligned}$$

the constraints (15b)(15c)(15d)(15e) distinctly define the initial and final states of the system, where $\mathbf{x} \in \mathbb{R}^2 \in [x, z]$ represents the state vector. The system operates under hybrid dynamics: during the stance phase, the spring force ($\mathbf{F}_s \in \mathbb{R}$) and control inputs ($\mathbf{u} \in \mathbb{R}^{2\times N_s}$, representing additional CoM acceleration) are the primary dynamics, while in the flight phase, the system follows a gravity-dominated parabolic trajectory and becomes uncontrollable. The discretization step-sizes for both stance and flight phases are contained within $\mathbf{dt} \in \mathbb{R}^2 = [dt_s, dt_f]$. The state position $\mathbf{X}(k+1) = Taylor(\cdot(k))$ is approximated through a second-order Taylor expansion, using the respective component of the step-size $\mathbf{dt}$ during stance and flight phase, and the state acceleration (constraints (15j) or (15l)) for the respective phase, i.e.:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \dot{\mathbf{x}}(k)dt + \frac{1}{2}\ddot{\mathbf{x}}(k)dt^2.$$

Additionally, constraints (15j) and (15k) ensure that the maximum acceleration in the negative Z-direction during stance does not exceed gravitational acceleration. Constraint (15f) specifies a mandatory z-position at iteration N_s. Simultaneously, constraint (15i) guarantees that the x-component of the CoM during stance remains within the support polygon defined by the robot's feet, assuming no movement of the feet during this phase. See Figure 2 for a graphical representation of the jump.

The cost function (15a) comprises components penalizing control input and jerk during stance, rewarding vertical velocity at takeoff, and penalizing deviation from the takeoff height during flight. These components are:

$$J_{\text{cost}} = J_{\text{stance}} + J_{\text{takeoff}} + J_{\text{flight}}, \tag{16a}$$

$$J_{\text{stance}} = \sum_{k=1}^{N_s} \omega_u \|\mathbf{u}(k)\|_2 + \omega_{\text{jerk}} \|\dddot{\mathbf{x}}(k)\|_2, \tag{16b}$$

$$J_{\text{takeoff}} = -\omega_{to}\dot{\mathbf{x}}_z(N_s-1), \tag{16c}$$

$$J_{\text{flight}} = \sum_{k=N_s}^{N_s+N_f} \omega_f \|\mathbf{x}_z(k) - \mathbf{x}_{z,\text{takeoff}}\|_2. \tag{16d}$$

APPENDIX B
VARIABLE IMPEDANCE CONTROLLER: IMPLEMENTATION

The foundation of the variable impedance controller (VIC) is an optimization problem [9]:

$$\begin{aligned}
&\underset{D,K}{\operatorname{argmin}} && h(D,K), \qquad\qquad (17)\\
&\text{s.t.} && l_{d_{i,j}} \le d_{i,j} \le u_{d_{i,j}},\\
& && l_{k_{i,j}} \le k_{i,j} \le u_{k_{i,j}},\\
&\text{s.t.} && \max_{\tilde{x}_0,\dot{\tilde{x}}_0,F_{ext}} |\tilde{x}_i(t)| \le b_i \quad \forall t \in [0,+\infty),\\
&\text{s.t.} && l_{\tilde{x}_{0_i}} \le \tilde{x}_{0_i} \le u_{\tilde{x}_{0_i}},\\
& && l_{\dot{\tilde{x}}_{0_i}} \le \dot{\tilde{x}}_{0_i} \le u_{\dot{\tilde{x}}_{0_i}},\\
& && l_{F_{ext_i}} \le F_{ext_i} \le u_{F_{ext_i}},\\
& && \Lambda(q)\ddot{\tilde{x}} + D\dot{\tilde{x}} + K\tilde{x} = F_{ext}.
\end{aligned}$$

This optimization seeks to find matrices D and K that minimize the cost function $h(D,K)$. The solution ensures boundedness of the tracking error $\tilde{x}$ over time. Specifically, the optimization constrains the peaks of each component $\tilde{x}_i(t)$ to remain within predefined bounds b_i, regardless of the duration. This constraint holds under the assumption that external forces F_{ext} and initial conditions $\tilde{x}_0$ and $\dot{\tilde{x}}_0$ are within set boundaries.

In the optimization problem, $\tilde{x}$ denotes the tracking error computed between the desired trajectory and $\mathbf{X}^+_{1:6}$. The matrix $\Lambda(q) \in \mathbb{R}^{6\times 6}$ represents the positive-definite Cartesian inertia and the vector $F_{ext} \in \mathbb{R}^6$ encapsulates the external force/torque acting on the robot's CoM.

The elements of the damping matrix D and the stiffness matrix K are denoted by $d_{i,j}$ and $k_{i,j}$, respectively. These matrices are subject to bilateral constraints, which are represented using the lower bound operator $l(\cdot)$ and the upper bound operator $u(\cdot)$. These constraints apply to the variables $d_{i,j}$, $k_{i,j}$, $\tilde{x}_0$, $\dot{\tilde{x}}_0$, and F_{ext}, ensuring that the system's physical and operational limits are respected.

The optimization problem outlined in Equation (17) is not directly implemented on the robot. Instead, as detailed in [9], several simplifications enable the closed-form solution of the problem. This leads to the following expression for the damping coefficient d_i:

$$d_i = \min\left(\max\left(l_{d_i}, \frac{2m_i\dot{\tilde{x}}_{0_i,\max}}{(b_i - \tilde{x}_{0_i,\max})e}\right), u_{d_i}\right). \tag{18}$$

Here, m_i represents the i_{th} diagonal element of the matrix Λ. Whereas $\tilde{x}_{0_i,\max}$ and $\dot{\tilde{x}}_{0_i,\max}$, are defined respectively as:

$$\tilde{x}_{0_i,\max} \triangleq \max\left(|l_{\tilde{x}_{0_i}}|, u_{\tilde{x}_{0_i}}\right),$$
$$\dot{\tilde{x}}_{0_i,\max} \triangleq \max\left(|l_{\dot{\tilde{x}}_{0_i}}|, u_{\dot{\tilde{x}}_{0_i}}\right).$$

Key to deriving the solution (18) are two assumptions: firstly, treating $\Lambda(q)$ as a *diagonally dominant* matrix, which allows for decoupling the system's dynamics. This enables solving each i_{th} element of the optimization problem (17) individually. Secondly, a *critically damped* behavior is imposed to the controlled system, which relates the damping coefficient d_i to the stiffness coefficient k_i as per the following relationship:

$$k_i = \frac{d_i^2}{4m_i}. \tag{20}$$

After determining the appropriate values for damping and stiffness using Equations (18) and (20), it is imperative to ensure the stability of the closed-loop system with time-varying gains. This is verified using the stability condition demonstrated in [9]:

$$d_i(t+T) > d_i(t) - \frac{d_i^2(t)T}{m_i(t)} + \frac{d_i(t)\dot{m}_i(t)T}{m_i(t)}. \tag{21}$$

If the calculated d_i values satisfy Equation (21), d_i and k_i can be applied to the VIC as new damping and stiffness parameters. If not, d_i must be adjusted to the right-hand side of Equation (21) plus a small increment, and k_i recalculated using Equation (20).

APPENDIX C

In Table II, we present a comprehensive list of parameters associated with both the *Feedback NN* and the *Feedforward NN*.

	Feedback NN	**Feedforward NN**
Epochs	400	1000
Batch Size	500	2
Input Size	36	35
Output Size	12	5700
Normalization Layer	yes	yes
Learning Rate	0.001	0.001
Hidden Layer	2×1024 neurons	2×128 neurons
Activation Function	ReLU	eLU
Optimizer	Adam	Adam
Landing error on x-position of CoM		
Mean	$2.0cm$	$5.7cm$
Std	$1.2cm$	$3.5cm$
Landing error on z-position of CoM		
Mean	$1.7cm$	$1.8cm$
Std	$0.7cm$	$0.6cm$

TABLE II: Comparison of Feedback and Feedforward Neural Networks.

This delineation facilitates a direct comparison of the two methodologies, particularly in terms of landing precision, as illustrated at the end of the table, and in terms of soft landing as illustrated in the Figures 18, 19, and 20. Upon evaluation, it is evident that the performance metrics of the *Feedforward NN* are inferior when compared to the *Feedback NN*. Consequently, for the specified task of this work, the *Feedback NN* is identified as the more viable and effective approach.

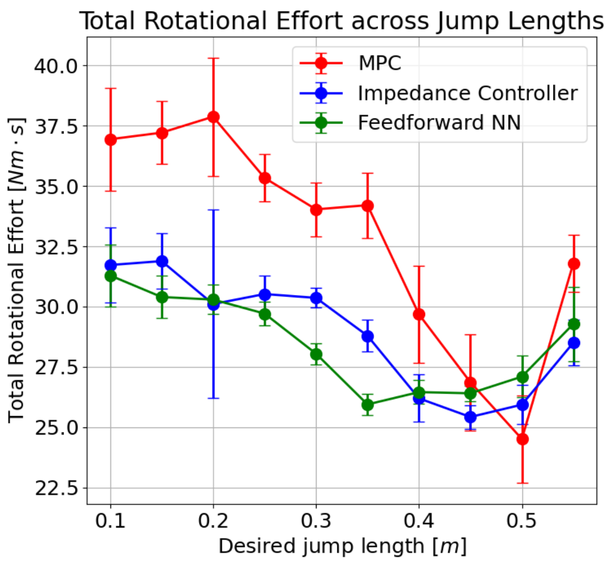


Fig. 18: Comparative analysis of *rotational effort* between the MPC (in red), the VIC (in blue), and the *Feedforward NN* (in green). For this metric, the network can properly clone the behavior of the VIC.

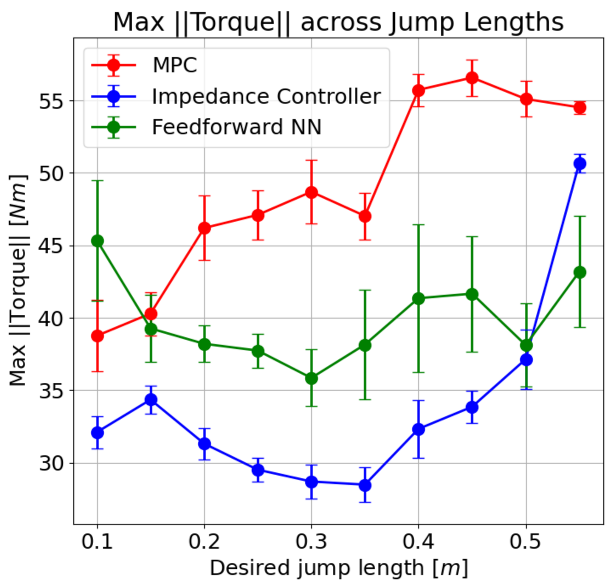


Fig. 19: Comparative analysis of *peak effort* between the MPC (in red), the VIC (in blue), and the *Feedforward NN* (in green). The network's results are worse than both the VIC but better than the MPC.

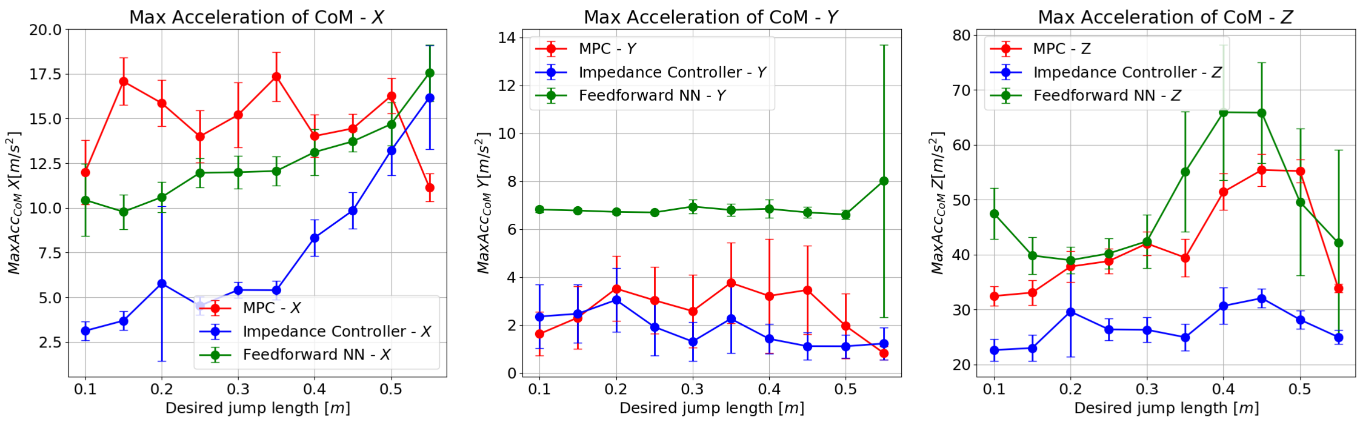


Fig. 20: Three graphs depicting CoM accelerations during landing: in blue, the VIC controller's performance, in red the MPC, and in green the *Feedforward NN*. The graph shows how the NN results are the least compliance solution along y and z, and it is only partially able to reproduce the VIC's results along x.