MSc Robotics Thesis

Curriculum-Based Deep Reinforcement Learning for Explosive Jumping in Quadruped Robots

Vassil Atanassov

Date: 09/08/23

TU Delft

Supervisors: Dr. Jiatao Ding
Dr. Cosimo Della Santina

Curriculum-Based Deep Reinforcement Learning for Explosive Jumping in Quadruped Robots

Master of Science Thesis

For the degree of Master of Science in Robotics at Delft University of Technology

Vassil Atanassov

Student Number.: 5501709

Supervisors:	Dr. C. Della Santina,	TU Delft
	Dr. J. Ding,	TU Delft
Thesis Committee:	Dr. C. Della Santina,	TU Delft
	Dr. J. Ding,	TU Delft
	Dr. J. Kober,	TU Delft
	Dr. A. Sharifi Kolarijani,	TU Delft

Submission Date: August 9, 2023

TU Delft

Department of Cognitive Robotics,
Faculty of Mechanical, Maritime and Materials Engineering,
Delft University of Technology

Acknowledgements

I would first like to thank both of my supervisors Dr. Jiatao Ding and Dr. Cosimo Della Santina for all of their amazing guidance and support throughout my Master's degree, and especially during my Master's project. I would also like to thank my colleagues and friends who not only helped me with numerous experiments throughout the last couple of months, but with whom I shared many pleasant chats and productive discussions over lunch and afternoon coffee. Last, but not least, I am also thankful for all of my friends and my family who supported me throughout this adventure and made my life in the Netherlands truly amazing!

Vassil Atanassov
August 9, 2023

Curriculum-Based Deep Reinforcement Learning for Explosive Jumping in Quadruped Robots

Vassil Atanassov

***Abstract*—Legged animals possess extraordinary agility with which they can gracefully traverse a wide range of environments, from running through grasslands to jumping across cliffs and climbing nearly vertical walls. Inspired by this, in this work, we use Deep Reinforcement Learning to give legged robots the ability to perform a diverse set of highly explosive and agile jumps. Unlike other works, our approach is not constrained to imitating a reference trajectory. We instead use curriculum-based learning to progressively learn more challenging tasks, starting from a vertical high jump and then generalising to forward and diagonal jumps. In the final curriculum stage, the robot learns to leap over barrier-like obstacles or to land on them, conditioned on the desired jumping distance and the object's dimensions. We show that such an approach can produce a wide range of robust and precise motions, which we thoroughly and successfully validated in several indoor and outdoor real-world experiments on the Unitree Go1 robot. In our real-world experiments, we show a forward jump of 90cm, exceeding previous records for similar robots reported in the literature. Additionally, we investigate the effects of incorporating bio-inspired parallel elastic actuators to improve the jumping performance further. This resulted in smoother motions, much softer landings with lower joint velocities and larger jumps. Finally, we present and analyse the limitations of our method and introduce exciting directions for future work to address them.**

I. Introduction

Robotics has the exceptional promise to revolutionise many aspects of the world, from exploring intricate cave systems and scaling complex mountains to reducing the risk to human lives in hazardous environments and benefiting conservation efforts through autonomous environmental monitoring. However, amidst this remarkable variety of possibilities a great challenge still remains - how can such machines seamlessly traverse the complex environments around us?

We can see that through millions of years of evolution, legged animals have adapted to locomote with ease across the highly complex and discontinuous environments that are found in nature. Goats, for example, are capable of scaling nearly vertical mountainsides, and gracefully leaping across chasms several times their body length. Furthermore, traversing stairs, steps and curbs that are commonly found in human-developed environments is also seemingly effortless for most kinds of legged animals. While science and engineering have drawn inspiration from animals in their development of legged robots, the software capabilities to mimic even a fraction of this behaviour have not been seen in robotics to date.

Traditionally, legged locomotion control has been viewed through the lens of optimal model-based control, where a model of the system's dynamics and its interaction with the environment are used to produce desired control inputs. However, achieving a balance between computational efficiency and the accuracy provided by detailed full-order models has proven to be a challenging task. Real-time operation is achieved through linearisations around certain assumptions like zero pitch and roll [1], or by decoupling the complex optimisation problem into two distinct reduced-order dynamics- and kinematics-based components [2].
Furthermore, many existing methods employ various heuristics in the choice of parameters, such as the set of possible contact points, contact phases and durations, in order to make the problem more tractable [3]. These heuristics are used to simplify the problem but can result in conservative and sub-optimal performance that fails to generalise well to a wider range of motions. As is often the case, these parameters can vary strongly, depending on the gait and type of motion, and thus further complicate the problem.

Fig. 1. The Go1 performing an outdoors 50cm long jump across soft grassy terrain under our proposed method.

Recently there has been a drive towards exploring Machine learning, and a subset of it known as Reinforcement Learning (RL) in particular, as an alternative approach. Such methods substitute optimising over a cost function, subject to given mathematical models, for optimising a given reward through trial and error. While manually-defined heuristics can still be part of this reward function, many works instead have tried to learn from nature and incorporate more intuitive and bio-inspired terms, like energy minimisation [4, 5] or directly through imitation of the motion of real animals [6]. Learning-based methods have shown impressive generalisation and robustness capabilities [7–10], and have recently begun fusing internal and external sensor information to seamlessly traverse highly irregular mountainous environments [11–14]. Adaptation modules [8, 15, 16] have further allowed legged robots to exhibit varying behaviours that are conditioned on environmental parameters (like ground friction, restitution, etc.), rather than learning a single conservative policy for every type of terrain. Recent work has drawn more inspiration from animal locomotion and built visuomotor policies [17–19] that directly map environmental visual observations to desired

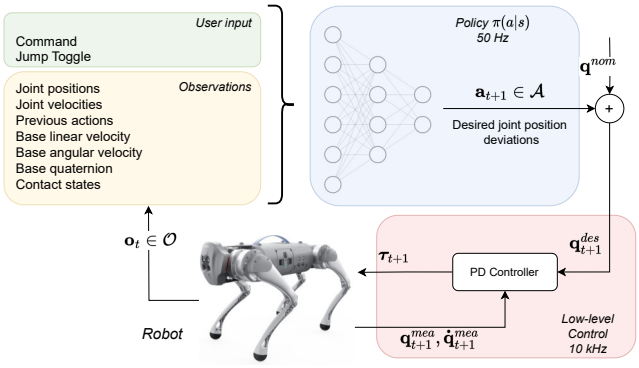

Fig. 2. Control diagram of the system. The observations $\mathbf{o}_t$ include user-controller (in green) and a history of system states (in yellow). The policy is parameterised by a neural network (shown in blue) and outputs desired actions $\mathbf{a}_{t+1}$ which are added to the nominal joint angles $\mathbf{q}^{nom}$. The desired joint angles are then tracked via a low-level PD controller which computes torques for the system to follow.

actions. Such works show that directly linking exteroception to locomotion can greatly improve the capabilities of legged robots acting in the real world.

In this work we tackle the challenge of enriching robots with the motor intelligence and agility to achieve similar feats to those of animals. Through a curriculum-based end-to-end deep reinforcement learning approach, we push the agent towards learning on its own how to best execute a given jump without relying on prior motion references. By conditioning the policy on the desired landing location, our approach can produce a wide range of motions with just a single policy. Furthermore, we incorporate partial knowledge of the obstacles in the environment as a proxy for vision and condition our neural network-based policy on this information. This leads to the robot learning different manoeuvres depending on the location and size of the obstacle, making its jumps much more efficient and robust. To further bridge the gap between robots and animals, we evaluate the effect of adding elastic elements in parallel with the actuators, which can serve a similar function to biological tendons.
Our main contributions can then be summarised as follows:

- We propose a curriculum-based end-to-end Deep Reinforcement Learning (DRL) framework that generalises across a wide range of jumps with a single policy and does not require motion capture data or a reference trajectory.
- We successfully and extensively validate our approach on the Unitree Go1 platform in the real world, across a variety of jumps, both indoors and outdoors. Our policy can produce forward jumps of up to 90cm, which, to the best of our knowledge, is greater than any other state-of-the-art controllers for this category of quadruped robots.
- We incorporate environmental obstacle information into the policy in the final curriculum stage which further enhances the mobility of the robot.
- We evaluate the benefits of adding springs in parallel with the actuators (PEA) to produce smoother and more accurate motions with softer landings.

II. Relevant Literature

A. Model-based control

Model-based control has been the staple for legged locomotion and has pushed the boundaries of what legged robots can do over the last few decades. For standard locomotion and less complex manoeuvres simplified models have shown to be sufficient [1, 20, 21]. The benefit of these models is much faster computation time, in some cases online and even in a low-frequency model-predictive fashion. However, in highly agile locomotion any deviations from the expected behaviour must be handled during the relatively short stance (on-ground) phase, lest they lead to unexpected and potentially catastrophic behaviour throughout the control-limited flight phase. During this aerial phase, the robots cannot adapt their trajectory, due to the lack of contact forces with the environment. While it is possible for them to modify their orientation by modulating their joint angles and therefore shaping their inertia, very few works tackle this [22, 23] due to computational complexity and robot design choices (e.g. most of the mass is concentrated in the body).

This has led to the common decoupling of the locomotion task into two modules - planning and tracking. The large computational overhead of trajectory optimisation of motion plans, based on accurate whole-body models, subject to a large number of kino-dynamic constraints, can then be done offline, and only the tracking of these plans is done in real-time [24, 25]. In order to reduce the computational expense, some choose to manually define the contact phases and durations at the cost of optimality [24] or use a reduced-order model to optimise over these parameters [25]. Even then, such methods can take over several minutes to plan a reference trajectory, depending on the complexity of the problem [25]. Recently Mastalli et al. [26] demonstrated highly agile online locomotion through Differential Dynamic Programming-based model predictive control. However, their approach still relies on a contact schedule produced by a separate planner module. Similarly, [27] has shown contact-rich behaviour, but has only been verified for simpler single-leg systems and requires a predefined set of possible contact points. Contact-implicit optimisation [28, 29] is a promising direction to remedy this, but real-time performance has not yet been achieved.

B. Reinforcement learning-based control

In contrast, goal-conditioned reinforcement learning (RL) presents a promising solution by offloading the computational complexity to offline training, enabling swift and versatile online performance. Several pioneering works in locomotion RL [6–10, 14, 15] have demonstrated outstanding results in achieving high-speed and robust locomotion, capable of adapting to variations in the environment and the dynamics of the system. However, research in realising highly agile behaviours has been more limited. In one of the early works that tackled this, Peng et al. [30] showed that Imitation Learning (IL) [31] can be a powerful tool in leveraging demonstrations to learn to perform various acrobatic motions, like backflips. This has since become one of the de facto

methods of learning more agile motions in literature, due to the relative difficulty in defining reward functions for such behaviours [6, 32–35]. These methods can rely on animal motion capture data [6, 36], on trajectories generated through optimal control [32, 33, 37], or manually created reference motions [34, 38]. To address the challenges associated with imitation learning, such as the selection of states to mimic and managing conflicting objectives, an emerging approach has been proposed called Generative Adversarial Reinforcement Learning (GAIL) [39]. This approach has proven highly effective and versatile at replicating the overall trends observed in the reference set, while also adapting them to real-world systems [40–42], even when dealing with partially incomplete and rough demonstrations [38].
One of the main disadvantages of imitation-based methods is that they have so far shown a limited ability to generalise beyond the imitation domain. In the realm of locomotion, one approach to handle this is to encode these motions in a low-level reusable locomotion module, which can then be controlled by a more general and separate task-specific high-level controller [43–45]. Within agile jumping literature, on the other hand, very few works have demonstrated such generalisation capabilities. Furthermore, many of the aforementioned works rely on learning a separate policy for each unique type of motion, rather than a common task- or goal-conditioned policy.

Several methods have been proposed that do not rely on a reference trajectory or motion prior. In [17], a high-level motion planning module is trained to produce desired centre of mass trajectories for small hops, conditioned on visual inputs and then tracked by a model-based controller. Similarly, in [46] deviations to reference trajectories generated by a non-linear optimal trajectory [25] are learned, providing better generalisation to out-of-training domains. Rudin et al. [23] show cat-like jumping in low gravity by using a more complex reward function but without requiring the imitation of motion clips. However, their approach has not yet been verified on Earth-like gravitational conditions. Concurrent work [34] also used multi-stage training on the Cassie robot to learn imitation-based vertical jumping, and then transferred that knowledge to forward jumping. While similar to our approach, however, there are a couple of significant differences - we do not require any reference trajectories and we learn a single policy for the whole task. Rather than directly imitating a dataset, [47] use a variational auto-encoder (VAE) to encapsulate motion capture data into a latent space for a DRL policy. They combine that with a Bayesian diversity search to discover viable take-off states for athletic vaulting strategies.

C. Curriculum Learning

Within machine learning, curriculum learning (CL) [48] is a training strategy which progressively provides more challenging data or tasks as the model improves. As the name suggests, the idea behind the approach borrows from human education and animal training, where complex tasks are taught by breaking them into simpler parts and then learning them consecutively. From a mathematical perspective, the method can be seen as a form of a continuation method, where the convergence is guided by progressively shifting from a heavily smoothed objective to the true objective function [48, 49]. In legged locomotion, CL has seen wide use, mainly in terms of terrain generation. Xie et al. [11] show how an adaptive curriculum can be used to learn stepping stone skills much more efficiently than other methods like uniform sampling. Similarly, other automatic curriculum learning methods have been proposed to vary environmental parameters based on the performance of the agents [9, 50], rather than using a manually specified curriculum. On the rewards side, Hwangbo et al. [7] employ a curriculum which scales down certain rewards at the start. This allows the policy to first learn how to locomote and only afterwards to be polished to satisfy the additional constraints and limits of the problem.

III. Methodology

Defining and constraining the behaviour of jumping across specific distances is challenging as it combines two distinct behaviours: that of "jumping" and that of reaching a desired spatial point. Furthermore, an easily learnable local optimum exists, where the robot could simply walk (or crawl) towards the target point without actually jumping.
Therefore, to avoid converging to such undesired behaviour we use curriculum-learning to divide the problem into several smaller and simpler sub-tasks that each encapsulate aspects of the full task while simultaneously being easier for the agent to learn. Our implementation is shown in Fig. 3. We combine this with a clever design of state initialisations and perturbations, which serve to guide the agent towards the desired behaviour.

Throughout the next several subsections we first provide some background into reinforcement learning, followed by an in-depth look into the curriculum, and the specific choice of observations, action space and rewards.

A. Reinforcement Learning and MDP

Within the field of machine learning, reinforcement learning (RL) is a method of inferring a policy $\pi(a_t|s_t)$ of how to act by constantly observing and interacting with the environment, while receiving a reward for its behaviour as feedback. In this framework, the problem is typically formulated as a Markov Decision Process (MDP), where at each step the agent interacts with the environment by taking an action $\mathbf{a}_t \in \mathcal{A}$. Subsequently, it receives the new states of the environment $\mathbf{s}_{t+1} \in \mathcal{O}$ in the form of an observation, and the associated reward $\mathcal{R}_t$ that it has earned. Based on the observed state s_{t+1} and its policy $\pi(a_{t+1}|s_{t+1})$ the agent can then choose a new action a_{t+1} and repeat the process. In this way, the RL algorithm explores the environment and optimises behaviours that yield high rewards.
This lends itself well to robot control where we often want to encourage a certain behaviour or accomplish a given task but do not explicitly have a set of correct examples to learn

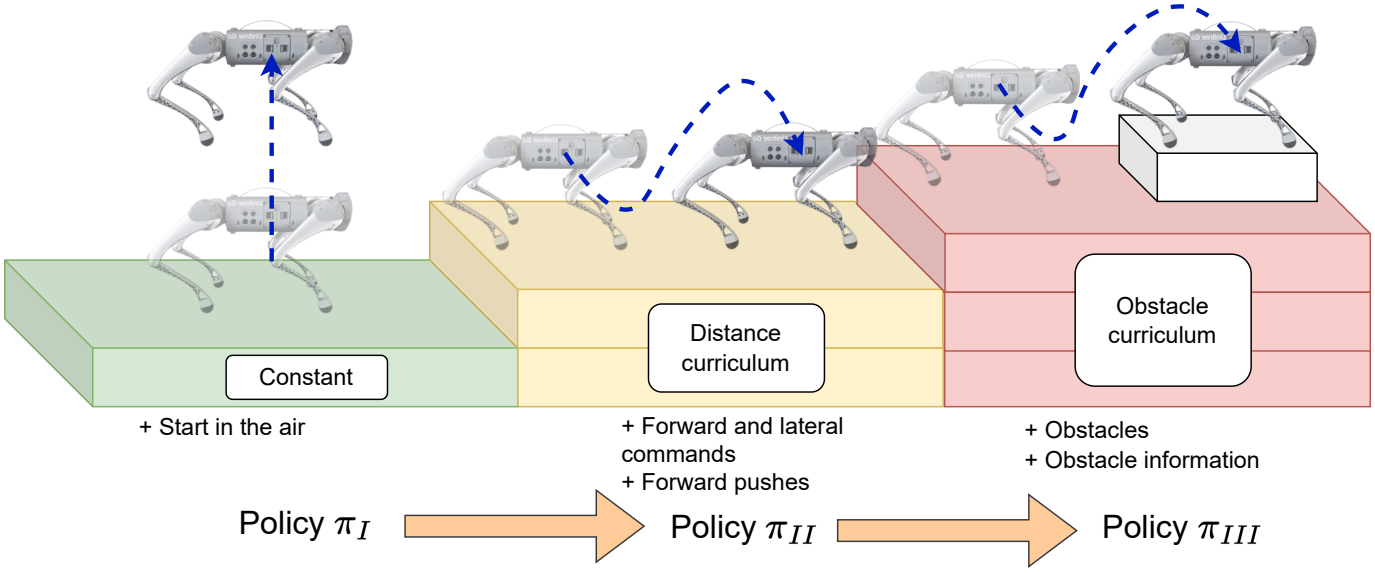

Fig. 3. The curriculum training stages: high jump (left), long-distance jump (middle) and long-distance with obstacles (right). The latter two contain separate difficulty curricula, varying the jump distance and obstacle height, respectively.

from. The policy can also be conditioned on some other goals g, i.e. $\pi(a_t|s_t, g)$, known as goal-conditioned reinforcement learning. Such a policy can then learn to produce a diverse set of behaviours depending on the specific command g that is given. This enables the learning of multiple distinct behaviours under a single policy.

Based on this, we can formulate an objective for the agent: To find a policy $\pi(a|s, g)$ which maximises the cumulative sum of rewards earned over the task duration. As often immediate rewards are more valuable than rewards in the distant future, a discount factor $\gamma \in (0, 1]$ is commonly used. Mathematically, the full objective of maximising the sum of discounted rewards J, known as the return, can be written as:

$$\arg\max_{\pi} \quad J(\pi) = \mathbb{E}_{\tau\sim p^{\pi}(\tau)}[\sum_{t=0}^{T} \gamma^t R_t|s_0 = s], \qquad (1)$$

where R_t is the immediate reward at time t and s_0 is the initial state. The expectation of the return is taken over a trajectory τ sampled by following the policy.

In legged robot control, we are usually interested in continuous action spaces - where each action within the action space $\mathcal{A}$ takes up a continuous value. One method of doing that is to parameterise the policy with a neural network, an approach known as Deep Reinforcement Learning [51]. In our case, the neural network outputs a mean value μ and a standard deviation σ for each of the actions in the action space. To encourage exploration throughout training the applied action a_t is stochastically sampled from its corresponding normal distribution $\mathcal{N}(\mu, \sigma)$, while during evaluation the mean value μ is used to ensure a deterministic behaviour.
The Policy Gradient family of algorithms [52, 53] are commonly used in Deep RL, as they directly optimise the parametric policy based on the gradient of expected return ∇J. Within simulation-based training for quadruped robots, one of the most popular algorithms of this family is Proximal Policy Optimization (PPO) [54], which itself is based on trust-region optimisation [55]. PPO uses a surrogate loss function to approximate and clip the probability ratio between the previous and the new policies at each update step. This clipping is crucial for maintaining stability by preventing excessively large update steps that may deviate from the surrogate approximation of the true objective function. Due to its stability, relative robustness to hyperparameter choice and highly optimised open-source implementations, it is the algorithm we chose for this problem. Our hyperparameter selection is reported in the Appendix in Table VI.

B. *Curriculum-based approach*

In our approach, we adopt two types of curriculum - on a local difficulty level and on a task level, as seen in Fig. 3. The former involves progressively (and automatically) making the environment more challenging as the agent succeeds. In particular, whenever a robot successfully executes several jumps, we increase the range of desired jumping distances (and obstacles heights) that we sample from. The task-level curriculum is, on the other hand, manually selected and consists of training the agent for a certain number of steps at a given task. After this, the policy is loaded onto the next task, which might be defined differently and contains a new set of rewards, and the training is continued. In the remainder of this section, we will describe each of these task-level curricula in the progressive order of training.

Stage I: High jump: The robot should first explore what it means to jump and what actions can achieve such explosive aerial motions. As mentioned, to learn this task we deviate from the common reliance on a reference trajectory

and allow the agent to fully learn on its own how to jump. We do this by rewarding it for being in the air and for the maximum height it has achieved. However, a well-known issue with such relatively sparse rewards is that the agent needs to first learn certain behaviours (for example to squat down and then push hard against the ground to take off) before it can reach the reward-rich states of the environment (i.e. being high in the air). This makes the learning process challenging as the robot will not experience these jumping-specific rewards in the initial stages of training. This often leads to an early convergence to a local maximum, such as standing in place, where the robot can safely collect some rewards.

In imitation-based works a strategy known as reference state initialisation (RSI) is commonly used to handle this issue [30, 32, 33]. RSI initialises the agent at random points of the trajectory which allows it to explore such reward-rich states before it has learned the actions necessary to reach them, and has been shown to boost learning rates significantly. In the spirit of this, we utilise an initialisation strategy that samples a random height and upwards velocity from a predefined range, showing the robot that reaching such aerial states can yield very high rewards.

Stage II: Long-distance jump: Once the robot has converged to a jumping-in-place behaviour, we further train it to perform precise forward and diagonal jumps. The first part of the command vector $\mathbf{g}$ in the observations specifies the desired landing point and orientation to create a goal-conditioned policy. Similarly to the **high jump** sub-task, we push towards exploring such motions by physically pushing a subset of the robots in the direction of the desired landing points once they have taken off. The push consists of setting the planar velocity of the robot to the desired velocity $\mathbf{v}^b_{x,y,des} = \mathbf{\Delta p}_{desx,y}/\Delta t_{jump}$. This value is approximated based on the distance to the target $\mathbf{\Delta p}_{des_{x,y}}$ and the duration of the jump, similarly to [34]. Certain parameters, like joint friction, can greatly affect the jump height and subsequently the flight time duration Δt_{jump}. Therefore we collected a dataset of joint friction values and corresponding flight time durations on the **high jump** stage and fit a line to estimate the slope and intercept. Throughout training, we would then pick a flight time duration Δt_{jump} based on the randomised value of the joint friction (as part of the domain randomisation as mentioned in Section III-E). It is worth mentioning that these pushes are sub-optimal and often result in the robot landing very roughly or even crashing close to the desired point. We also adopt a curriculum-style sampling for desired landing points, where successful agents are progressed to more difficult environments where the desired jumping distance is sampled from a greater range.

Stage III: Long-distance jump with obstacles: Once this behaviour has been mastered, we introduce obstacles in the environment. We have three classes of obstacles - thin barrier-like objects, larger box-shaped obstacles and slopes. Depending on the desired landing pose, the location of the object and its type, the agent might need to either jump onto or lunge over it. While it is possible to learn a general behaviour that can accomplish this without any exteroception, such behaviour will be conservative, sub-optimal and potentially much less robust. We argue that "seeing" and understanding the world around the robot can result in a policy that adapts the motion to better fit the scenario it has encountered. With this in mind, we incorporate information about the distance to the centre of the obstacles and its general dimensions (length, width and height). In the real world, we manually specify these parameters, however, a separate module that estimates them from the RGBD cameras available on the robot could be utilised. A more elegant approach, and a direction for future work, would be to directly link these exteroceptive sensors to the policy and remove the parameterisation of the world around the robot.

Similarly to the previous stage, we start with obstacles of smaller height - successful robots are then progressed towards taller and more challenging environments and failing ones are demoted to easier environments. To ensure that the robot remembers the previously learned behaviour we also randomly send a certain percentage of robots to perform a flat terrain jump, like in Stage 2.

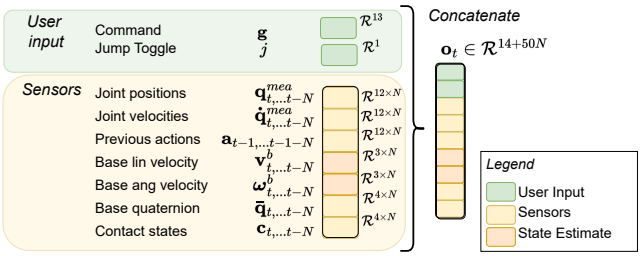

Fig. 4. The observations vector. The command $\mathbf{g}$ and jump toggle j are provided by the user, while the remaining observations are either directly read from the sensors, or estimated from sensor data.

C. Observation and action space

Observation Space: Using a memory of previous observations and actions is believed to allow the agent to implicitly reason about its own dynamics and those of the world around it [35, 56]. Similar to other works in the field [7–10, 14, 16, 56–58] we use a concatenated history of the last N states and actions as input to the policy. Certain works [9, 34] have suggested combining this with a Convolutional Neural Network (CNN) or a Recurrent Neural Network (RNN) [37, 59, 60] to incorporate long-horizon past states. In this case, we found that only using the last 20 states, spanning 0.4 seconds, was sufficient for the task while also being faster and much easier to train.

The observation space $\mathcal{O}$, as illustrated in Fig. 4, consists of the last N steps history of the base linear velocity $\mathbf{v} \in \mathcal{R}^3$, base angular velocity $\boldsymbol{\omega} \in \mathcal{R}^3$ (both in the base frame), joint position $\mathbf{q} \in \mathcal{R}^{12xN}$, joint velocity $\dot{\mathbf{q}} \in \mathcal{R}^{12xN}$, previous actions $\mathbf{a}_{-1} \in \mathcal{R}^{12xN}$, the base orientation (as a quaternion) $\bar{\mathbf{q}} \in \mathcal{R}^{4xN}$ and the foot contact states $\mathbf{c} \in \mathcal{R}^{4xN}$.

The policy is also conditioned on the command $\mathbf{g} \in \mathcal{R}^{13}$ and

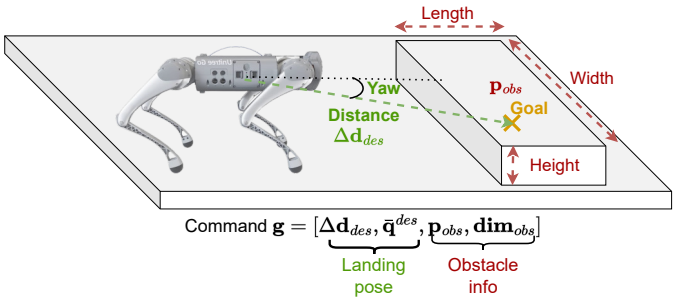

Fig. 5. An example of the forward jump onto an obstacle task, including the command vector $\mathbf{g}$. For the first two training stages (π_I and π_{II}) where there are no obstacles, the obstacle information part of the command is set to 0.

jump toggle $j \in \{0, 1\}$. Due to the lack of long-term memory in the feed-forward neural network, we use the jump state j to indicate whether the robot has already jumped, similarly to the approach in [30]. However, in our case, the jump toggle also serves as a control switch, where the robot will remain standing until its value has been changed, after which it will jump.

The command $\mathbf{g} \in [\Delta \mathbf{p}_{des}, \Delta \bar{\mathbf{q}}_{des}, \mathbf{p}_{obs}, \mathbf{dim}_{obs}]$ is used to condition the policy to achieve different jumps, and contains the desired Cartesian landing position and orientation relative to the base frame, the centre of the obstacle (if present), also relative to the base frame, and its dimensions (height, width, length). An illustration of this is shown in Fig. 5. Throughout training, we sample the landing pose and obtain the obstacle parameters from the simulation. In the real world, the command vector is specified by the user prior to the jump.

Action Space: For the action space $\mathcal{A} \in \mathcal{R}^{12}$, similarly to other works [7, 34] we used position control, in the form of learning desired deviations from the nominal joint positions $\mathbf{q}^{nom}$ [16, 58, 61, 62]. The benefit of this formulation is that producing a zero action results in a standing pose, making the learning process easier. To smooth the output actions we used an Exponential Moving Average (EMA) low-pass filter with a cut-off frequency of 5 Hz. The filtered actions are then scaled down and added to the nominal joint position $\mathbf{q}^{nom}$ to generate a desired joint position $\mathbf{q}^{des}$ for the motor to track, i.e. $\mathbf{q}^{des} = \mathbf{a} + \mathbf{q}^{nom}$. A simple feedback PD controller then produces a desired torque at a higher frequency to be sent to the motor, as shown in Fig. 2.
It is important to note that the policy can generate desired joint position deviations outside the physical joint limits. This allows us to use very low stiffness for the PD controller ($K_p = 20, K_d = 0.5$) while still ensuring that large torques can quickly be generated for the explosive motion at take-off. To ensure the safety of the robot, however, we clip the desired joint positions to the physical range **only when** the real joint angles approach the limits.

Policy: The actor policy π and the critic are parameterised by a shared MLP with 3 hidden layers of dimensions $[256, 128, 64]$, with Exponential Linear Unit (ELU) activations after each layer.

D. Rewards

All of the rewards used for training the desired behaviour are shown in Table I, where the task-based rewards are shown in orange and the auxiliary regularisation rewards - in violet. For conciseness, the short-hand notation $e(|x|^2)$ is used to represent passing the squared error $|x|^2$ through an exponential kernel of the form $exp(-|x|^2/\sigma)$. This ensures the reward is positive and scales it between 0 and 1. The rewards are divided into *Single* and *Continuous*, where the former are given once per episode (typically at the end), and the latter - once per each simulation step that satisfies the conditions. Three general phases of the jump are used to describe when each reward is given. Stance (or pre-jump) indicates that the robot has been given a command to jump but is still on the ground. The flight phase follows the stance and is triggered when the robot is in mid-air and has no contact with the ground. Finally, the landing (or post-jump) begins upon landing and lasts until the end of the episode. We consider the event where the robot must remain standing until it receives a jumping command as being in the post-jump phase.

Task rewards: We use several rewards to encourage the general behaviour associated with the task of jumping, namely those of detecting contact (landing) after several steps of no contact (flight), the maximum height the agent reached, and whether it has landed at the desired position and orientation. These rewards are only given once at the end of the episode, which further incentivises the agent to survive until the episode ends. Due to the sparsity of these rewards, we introduce several other task-related objectives that bring continuous rewards to the agent and simplify the learning problem. These are:

- Tracking the desired linear velocity ($\mathbf{v}^b_{x,y,des}$) and yaw angular velocity while in flight, and tracking zero angular velocity after landing.
- Squatting down to a height of 0.2m while on the ground, tracking a height of 0.8m in the air, and maintaining a stance height (0.32m) after landing.
- Maintaining a constant base position and tracking the desired orientation after landing.

In order to ensure enough clearance when jumping forward and over obstacles, we introduce a feet clearance reward. This reward tracks the nominal foot position (i.e. at the nominal joint angles q^{nom}) on the xy-plane, while simultaneously minimising the z-distance between each foot and the centre of mass. This results in the robot tucking its legs in close to its body while in the air which can help prevent them from hitting the ground during longer jumps.

Regularisation rewards: As we do not imprint any reference motions onto the agent, we incorporate several regularisation rewards to make sure the motion is smooth, feasible and will not damage the robot. To this end, we penalise the action rate and its derivative, together with any

TABLE I
LIST OF REWARDS USED FOR THE TRAINING. TASK REWARDS ARE GIVEN ONLY ONCE AT THE END OF THE EPISODE, WHILE CONTINUOUS REWARDS ARE EVALUATED AT EVERY TIME STEP. THE LIGHT ORANGE COLOUR INDICATES TASK-BASED REWARDS, WHILE THE LIGHT PURPLE SHADE DESCRIBED REGULARISATION REWARDS.

Name	Type	Stance	Flight	Landing
Landing position	Single	0	0	$w_p(e(\sum \lvert\mathbf{p}_{land} - \mathbf{p}_{des}\rvert^2))$
Landing orientation	Single	0	0	$w_{ori}(e(\lvert log(\bar{\mathbf{q}}_{land}^{-1} * \bar{\mathbf{q}}_{des}\rvert^2))$
Max height	Single	0	0	$w_h(e(\lvert h_{max} - 0.9\rvert^2))$
Jumping	Single	0	0	w_{jump}
Position Tracking	Continuous	0	0	$w_{p.l}(e(\sum \lvert\mathbf{p} - \mathbf{p}_{land}\rvert^2))$
Orientation Tracking	Continuous	$w_{ori,st}(e(\lvert log(\bar{\mathbf{q}}_{base}^{-1} * \bar{\mathbf{q}}_{des}\rvert^2))$	0	$w_{ori,l}(e(\lvert log(\bar{\mathbf{q}}_{base}^{-1} * \bar{\mathbf{q}}_{des}\rvert^2))$
Base height	Continuous	$w_{p_z,sq}(e(\lvert p_z - 0.20\rvert^2)$	$w_{p_z,fl}(e(\lvert p_z - 0.8\rvert^2)$	$w_{p_z,st}(e(\lvert p_z - 0.32\rvert^2))$
Base lin. velocity	Continuous	0	$w_{\mathbf{v}_{x,y}}(e(\sum \lvert\mathbf{v}_{x,y} - \mathbf{v}_{des}\rvert^2)$	0
Base ang. velocity	Continuous	0	$w_{\boldsymbol{\omega}}(e(\sum \lvert\boldsymbol{\omega} - \boldsymbol{\omega}_{des}\rvert^2)$	$0.1w_{\boldsymbol{\omega}}(e(\sum \lvert\boldsymbol{\omega} - \mathbf{0}\rvert^2)$
Feet clearance	Continuous	0	$w_{feet}e(\lvert p_{feet} - p_{feet}^0 + [0.0, 0.0, -0.15]\rvert^2)$	0
Symmetry	Continuous	$w_{sym}(\sum_{joint} \lvert\mathbf{q}_{left} - \mathbf{q}_{right}\rvert^2)$		
Nominal pose	Continuous	$w_q(e(\sum_{joint} \lvert\mathbf{q}_j - \mathbf{q}_{j,nom}\rvert^2)$	$0.1w_q(e(\sum_{joint} \lvert\mathbf{q}_j - \mathbf{q}_{j,nom}\rvert^2)$	$w_q(e(\sum_{joint} \lvert\mathbf{q}_j - \mathbf{q}_{j,nom}\rvert^2)$
Energy	Continuous	$w_{energy}(\boldsymbol{\tau}^T \dot{\mathbf{q}})$		
Base acceleration	Continuous	$w_{acc}\lvert\dot{\mathbf{v}}\rvert^2$		
Contact forces	Continuous	$w_{F_c} \sum_{i=0}^{n_f} \lvert F_i - \bar{F}\rvert$		
Action rate	Continuous	$w_a \sum_{joint} \lvert\mathbf{a}(t) - \mathbf{a}(t-1)\rvert^2$		
Action rate 2nd order	Continuous	$w_{\dot{a}} \sum_{joint} \lvert\mathbf{a}_j(t) - 2\mathbf{a}_j(t-1) + \mathbf{a}_j(t-2)\rvert^2$		
Joint acceleration	Continuous	$w_{\ddot{q}} \sum_{joint} \lvert\ddot{\mathbf{q}}_j\rvert^2$		
Joint limits	Continuous	$w_{q_{lim}} \sum_{joint} \lvert\mathbf{q}_j - \mathbf{q}_{j,lim}\rvert^2$		

violations of predefined soft limits for the joint position. The instantaneous energy, computed as the dot product between actuator torque and joint velocity, is penalised to produce a more energy-efficient motion [4, 5]. Many quadrupedal jumps seen in nature exhibit high left- and right-side symmetry - we push the robot towards maintaining this symmetry with an additional reward but do not strictly enforce it. Finally, we noticed an emergent behaviour where the robot would stomp its feet rapidly and in place during the squat-down phase. A likely explanation is that the agent has begun to overfit and learned to exploit the dynamics of the simulator to accelerate faster towards the ground. To discourage this, we added a small reward for maintaining the same contact states for each foot between simulation steps.

Termination: We implement several termination conditions, which when met reset the episode and yield a large penalty to the agent:

- Collision between any part (other than the foot) of the robot and the environment.
- Base height lower than 0.10m.
- Orientation error larger than 3.0 rad.
- Landing position error bigger than 15cm.
- Moving the base after landing or when told to stand in place.

These termination conditions help constrain the agent away from highly undesirable states, from which the agent either cannot recover or are too distant from the desired task goal, in order to improve the learning process.

TABLE II
DOMAIN RANDOMISATION VARIABLES AND THEIR RANGES.

Name	Randomisation range
Ground friction	[0.01, 3.0]
Ground restitution	[0.0, 0.4]
Additional payload	[-1.0, 4.0] kg
Link mass factor	[0.7,1.3] x
Centre of mass displacement	[-0.1, 0.1] m
Episodic Latency	[0.0, 40.0] ms
Extra per-step latency	[-5.0, 5.0] ms
Motor Strength factor	[0.9, 1.1] x
Joint offsets	[-0.02, 0.02] rad
PD Gains factor	[0.9, 1.1] x
Spring Stiffness/Damping factor	[0.7,1.3] x
Spring rest position	[-0.1, 0.1] rad
Joint friction	[0.0, 0.04]
Joint damping	[0.0, 0.01] $\mathrm{N\,m\,s\,rad^{-1}}$

E. Domain Randomisation

In reinforcement learning (and more generally in machine learning) it is commonly known that discrepancies between the domain that the agent is trained on and the one it is deployed on can lead to sub-optimal performance or even critical failure, depending on the degree of mismatch. Domain randomisation is a method of addressing this issue, based on the idea that by introducing a large amount of variation into the training domain the agent can become more robust to domain distribution differences [59, 63, 64]. Robotics

has widely adopted this approach to handle inaccuracies in the environmental and system parameters, as well as in the model and its dynamics. To ease the domain transfer between simulation and the real world, we implement several common and lesser-known domain randomisation techniques. The ground friction, restitution, and the mass of the base and each link are sampled at random at the start of every episode. In addition, to deal with calibration discrepancies between the hardware and simulated motors we add a random offset to the joint encoder values in the low-level PD controller, randomise its proportional and derivative gains to simulate the effect of unknown motor friction and damping, and randomise the strength of the motors for every episode [16, 33, 62].
Unmodelled communication delays and latencies have been shown to strongly affect the performance of learning-based policies by reducing the stability region of the feedback controller when deployed in the real world [4], which is amplified in the case of high-frequency policies [65]. For this reason, at the beginning of each episode, we sample an observation to action latency from the range of $l \in [0, 40]$ ms. At each step, we add a small random value to the episodic latency to mirror the effect of stochastic communication delays. Implementation-wise, we follow a similar approach to [4], where we continuously store the past M observations $\mathcal{O}_{t,\dots t-M}$. At every step we pick the two observations $\mathbf{o}_{t-l}$ and $\mathbf{o}_{t-l+1}$ from $\mathcal{O}_{t,\dots t-M}$ between which the sampled latency l lies. Then we linearly interpolate between them to obtain the delayed observation which can be added to the observation history of the agent.

Randomising the friction and damping of the actuated joints in the simulation proved to be crucial for successful deployment. Policies that did not incorporate this would result in much lower joint velocities throughout the take-off phase and smaller jumps when deployed onto the real robot.

Finally, for the parallel elastic version we randomise all of the parameters of the parallel springs - including their stiffness, damping and rest position. While this could make the robot rely less on storing energy in the elastic elements, it is beneficial in terms of robustness.

IV. Experimental Validation

In the following sections, we first show the performance of the policy, trained on the first two curriculum stages (i.e. policy π_{II}, shown in Fig. 3), across several experiments - forward and diagonal jumps, several continuous back-to-back jumps, and jumping in the presence of environmental disturbances and uneven terrain. We then compare the performance of the policy after the final training stage (policy π_{III}) when jumping onto and over obstacles. Finally, we train a separate policy to investigate and show the effects of parallel elastic actuators. To make the simulation results more realistic, during the validation we used a constant joint friction value of 0.04, joint damping of 0.01 $Nmsrad^{-1}$ and a constant latency of 30 ms.

A. Hardware setup

We performed all of the experiments on the Unitree Go1 [66], which is a quadrupedal robot platform with three actuators per leg that control the hip abduction/adduction, thigh flexion/extension and calf flexion/extension. The hip and thigh motors are capable of producing instantaneous torques of up to 23.7 Nm and reaching a joint velocity of 30.1 rad/s. The calf motors are more powerful and can produce torques of up to 33.5 Nm and a maximum velocity of 20.06 rad/s. The lightweight nature and powerful motors of the robot make it a good potential platform for learning highly agile motions, such as jumping.

Nevertheless, animals are capable of producing very impressive motions without necessarily being able to achieve high instantaneous forces with their muscles. To mimic the biological energy storage mechanisms, we added springs in parallel with each actuator - in a so-called Parallel Elastic Actuator (PEA) arrangement. These uni-directional springs allow the robot to store energy as it squats down and then release the energy at once to achieve a more explosive motion. Through our DRL framework, we aim to investigate how the robot can leverage such elastic elements to jump higher, further and land more softly. The engagement mechanism allows us to easily enable the springs, specify their rest positions and swap between springs with different stiffnesses. In this work, the spring stiffness was heuristically chosen, but other recent works [67] have shown that RL help in generating an optimal set of parameters and it is something we are considering as future work. Here we used $K_{hip} = 10$, $K_{thigh} = 16$, $K_{calf} = 20$ Nm and rest positions of $0, 0.7220, -1.4471$ rad for the hip, thigh and calf joint respectively. In section IV-H we investigate and record the effect of the PEAs.

B. Training environment

We train our policy in the Isaac Gym simulator [68], where we use 4096 agents and 24 environmental steps per agent per update step, resulting in a batch size of 98304 steps. For the vertical jump, we train for 3k iterations, while for the forward jump without and with obstacles we train for 10k steps each. The policy operates at a frequency of 50Hz and the simulation runs at 200Hz. Our implementation is based on the open-source Legged Gym by ETH Zurich [61] and partially on some additional functionality introduced in [16].

The training was done on a desktop workstation equipped with a single RTX 3090 GPU, and the three highly parallelised training stages took approximately 1.4 hours, 4.1 hours and 4.8 hours, respectively.

C. Forward jumping

First, we evaluate the policy on a variety of forward jumps. In Fig. 6 we first show a 60cm forward jump, i.e. $\Delta p_{des} = [0.6, 0.0, 0.0]$. We compare the quantitative results from the simulation to those recorded on the real robot for

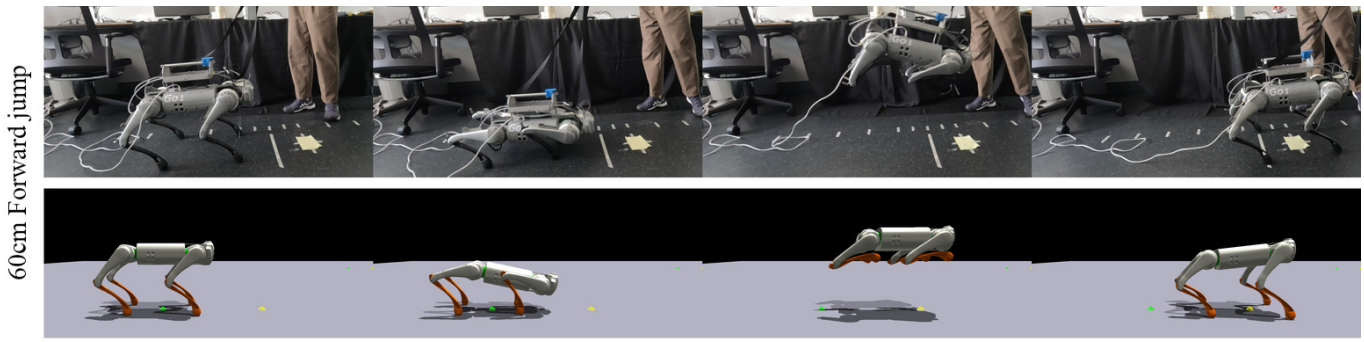

Fig. 6. Real world (top) and simulation (bottom) execution of a forward jump with the yellow marker indicating the desired 60cm jumping distance.

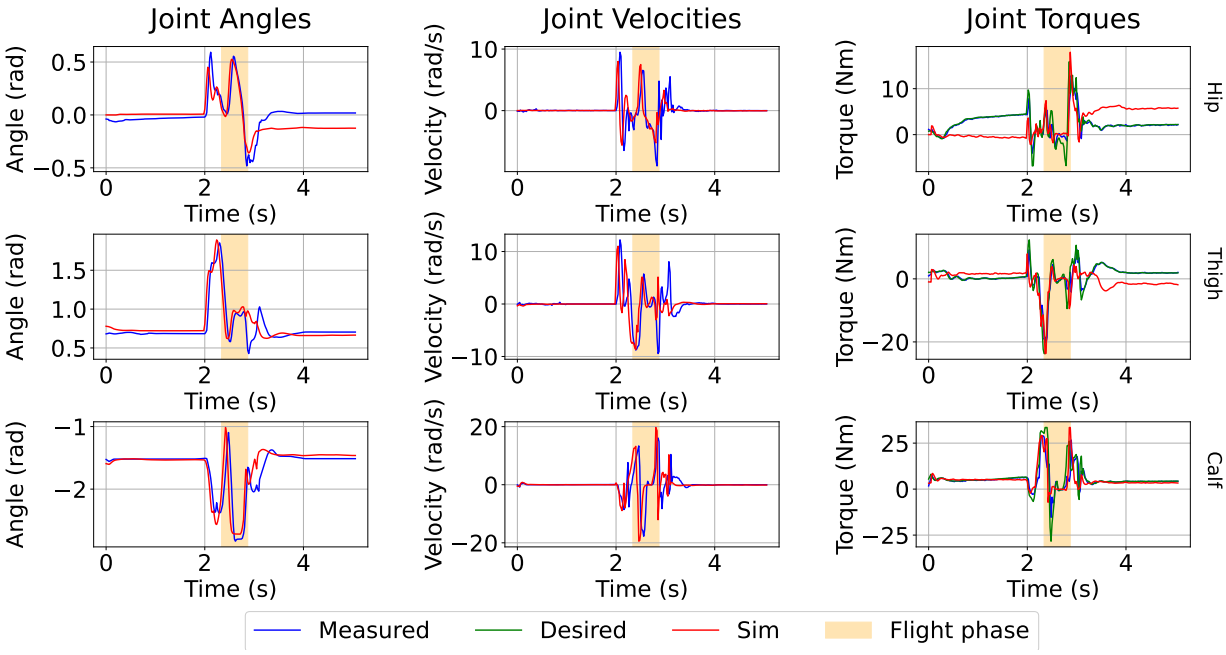

Fig. 7. Joint angles, velocities and torques for the front right (FR) leg during the 60cm forward jump. The measurements are shown in blue, the desired values (corresponding to the ideal PD control law for the torques) are shown in green, and the simulation results for the same jump are overlayed in red. The flight phase for the hardware experiment is indicated by the yellow-shaded region.

the former in Figures 7 and 8. As can be seen, the real-world behaviour closely matches the simulated prediction. One noticeable deviation is in the peak torques at take-off - where the measured torques deviate from both the desired torques (based on the PD control law on the desired joint angles) and the simulation torques. This could partially be attributed to inaccurate torque measurements, as they are estimated from the motor current draw, rather than dedicated torque sensors at the joints. Furthermore, additional joint-level friction and motor saturation could also explain the difference in torques. We also noticed that larger joint angles for the hip and thigh are measured upon landing in real-world tests, likely due to poor impact modelling in the simulation. Finally, the Euler angles show a slight variation between simulation and hardware. We hypothesise that this mismatch is mainly due to the aforementioned motor modelling inaccuracies, coupled with the weight of the additional mass, which contains the springs, shifting the centre of mass of the robot. Nevertheless, despite these small deviations, the distance is well-tracked and the velocity profiles of the robot match the expected behaviour, as seen in Fig. 8, showing a good sim-to-real adaptation. We then further pushed the robot to determine the maximum distance it can jump across. Figure 9 illustrates a 90cm forward jump, with the target landing point shown by the yellow marker. Interestingly, we observe a slightly more accurate landing in the hardware compared to the simulation. In addition, despite slipping on the soft pads as it lands, the robot recovers quickly, showing additional disturbance robustness. To the best of our knowledge, this is the largest jumping distance shown by robots of similar size shown in the literature. It is worth noting that in our approach we reward the position of the base upon landing, rather than the feet. In the trial, the base cleared the 90cm distance but the rear left foot landed slightly before the white line, after which it was moved forward when the robot stabilised itself.

In Table III we compare the maximum forward distance our

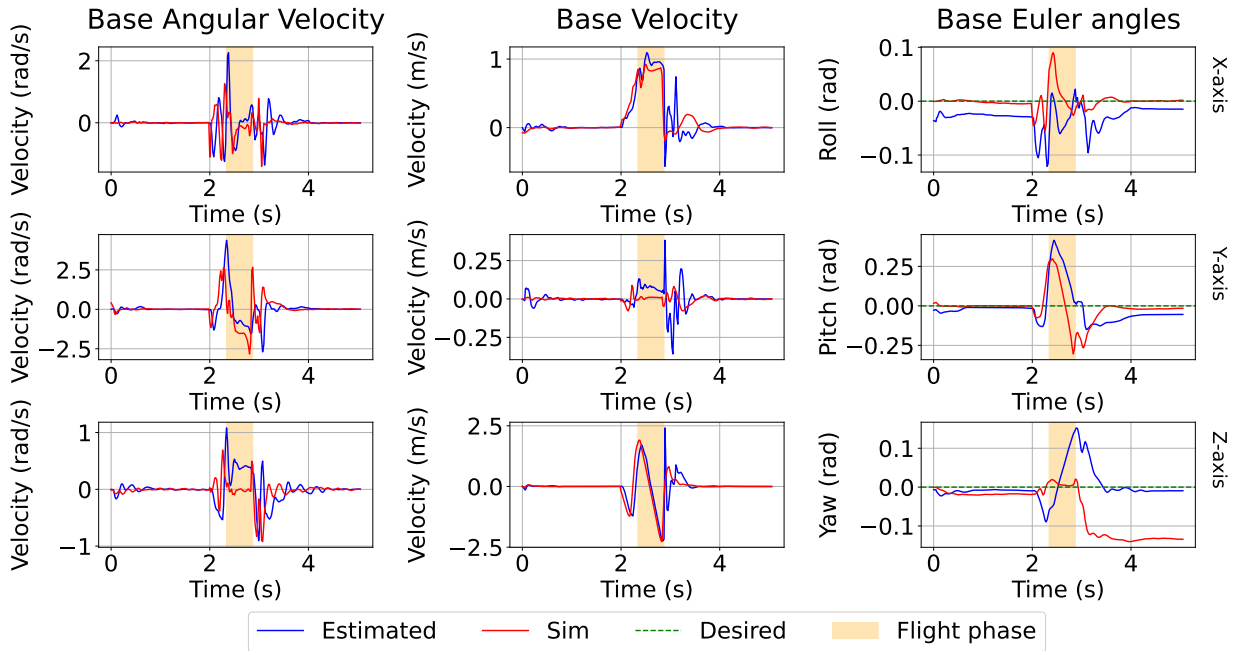

Fig. 8. Base angular and linear velocity estimates during the 60cm forward jump. The estimated values (via kinematics and the IMU) are shown in blue and are compared to the simulation results for the same jump (in red). The flight phase for the hardware test is indicated in light yellow.

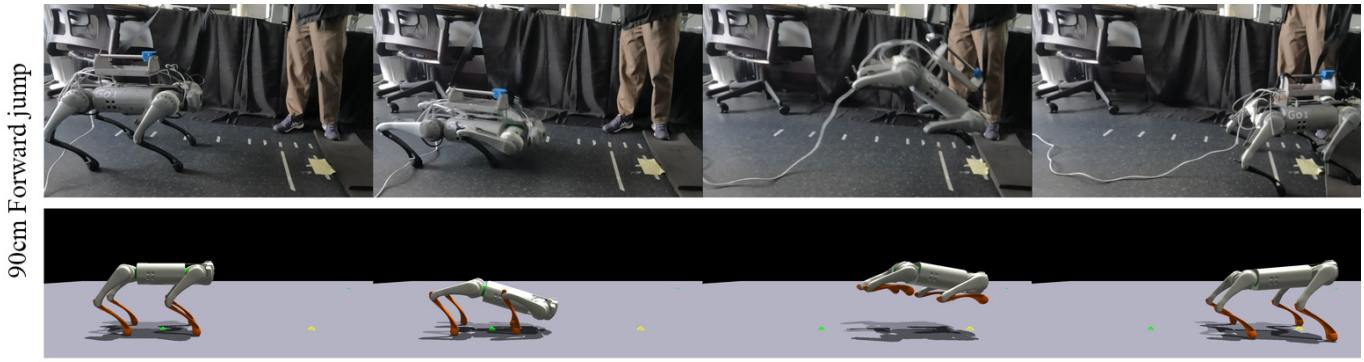

Fig. 9. Real world (top) and simulation (bottom) execution of a forward jump with the yellow marker indicating the desired 90cm jumping distance. Note that the rear left leg was slightly behind the white line upon landing, indicating a travel distance of around 85cm.

method achieved to that of other state-of-the-art learning-based approaches. Most of these works are either deployed on Unitree robots (A1/Go1) or other quadrupeds of similar size and capabilities.

D. Diagonal jumping

Next, in Fig. 10, we show the robot performing a diagonal jump of 50cm x 30 cm with a desired yaw of 30°. The same jump is shown from a frontal perspective in Fig. 11 (top), together with the same jump but with 0° desired yaw (bottom). Interestingly, for the most part, the two jumps are nearly identical, with the exception of the flight yaw velocity. Visually, the landing precision of both tasks is very similar, although the latter exhibits a slightly larger yaw error right after the landing impact.

We further evaluated the performance of the robot when prompted to jump in place and land with a desired yaw of ±60°, as shown in Figure 13. Despite some small error in the landing position, the robot could accurately track the

TABLE III
COMPARISON BETWEEN THE LARGEST JUMP LENGTH REPORTED BY OTHER WORKS AND OURS, ON ROBOTS OF SIMILAR SIZE. ADAPTED FROM [69].

Method	Largest jump length
TWiRl [70]	0.2m
Margolis et al. [17]	0.26m
Barkour [71]	0.5m
Yang et al. [72]	0.6m
CAJun [69]	0.7m
Ours	**0.9m**

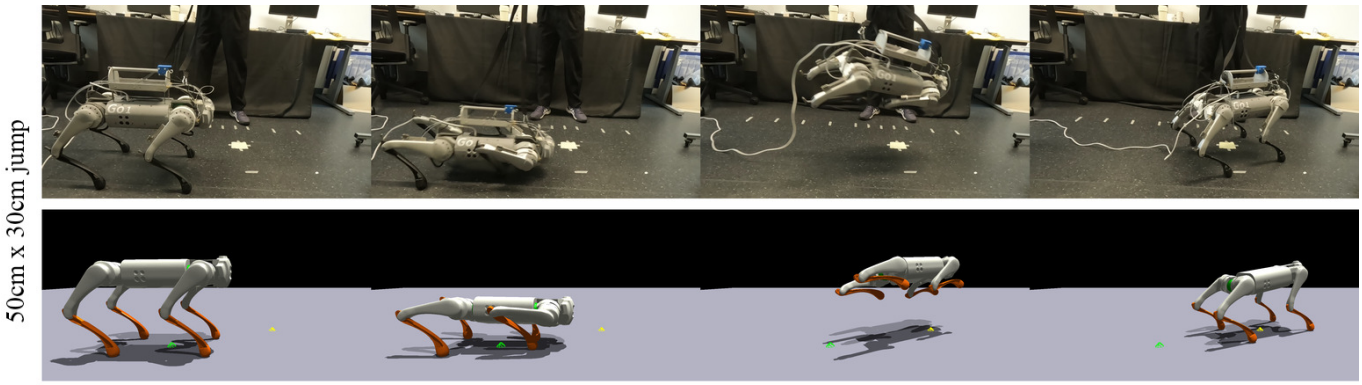

Fig. 10. Diagonal jump of 0.5m forward and 0.3m to the left with a desired yaw of 30°, comparing the real-world (top) and simulation (bottom) results.

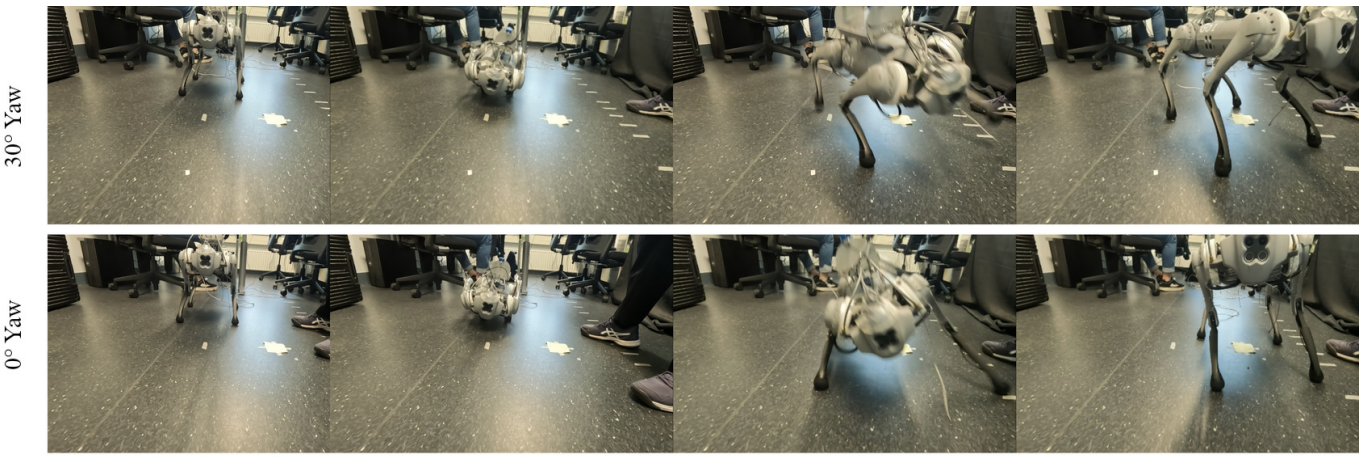

Fig. 11. Diagonal jump of 0.5m forward and 0.3m to the left with a desired yaw of 30° (top) and desired yaw of 0° (bottom), respectively.

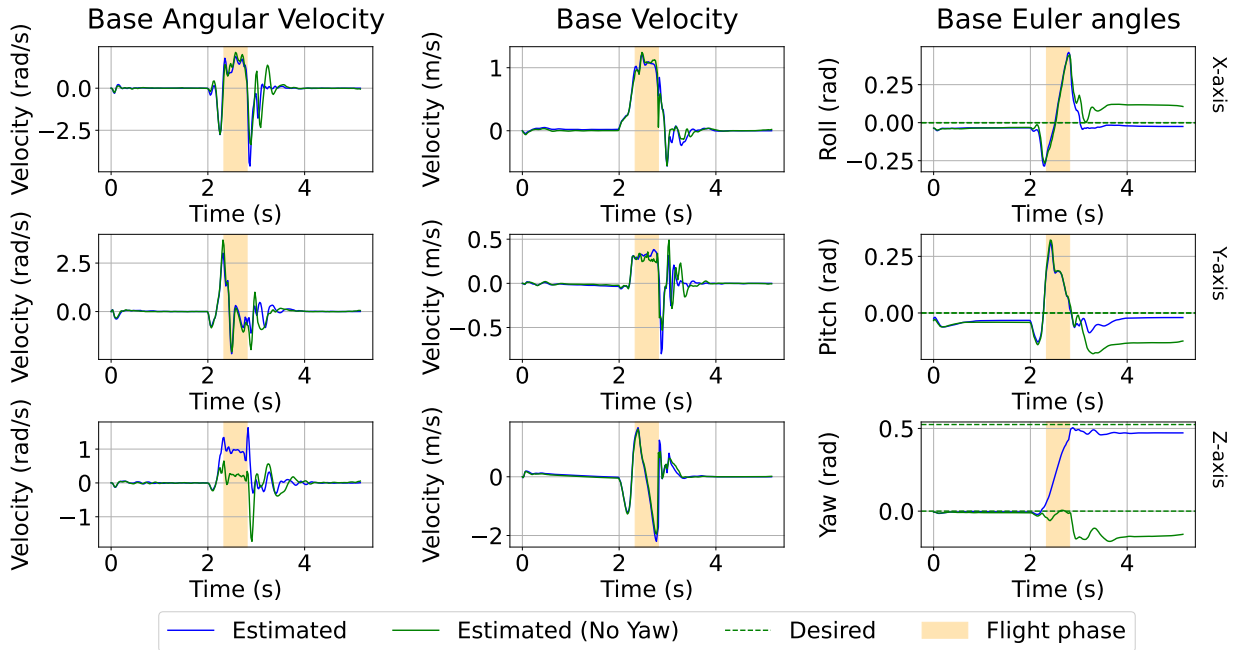

Fig. 12. Comparison of the base angular velocity, base linear velocity and the Euler angles of the 30° yaw (blue) and 0° yaw (in green) diagonal jumps.

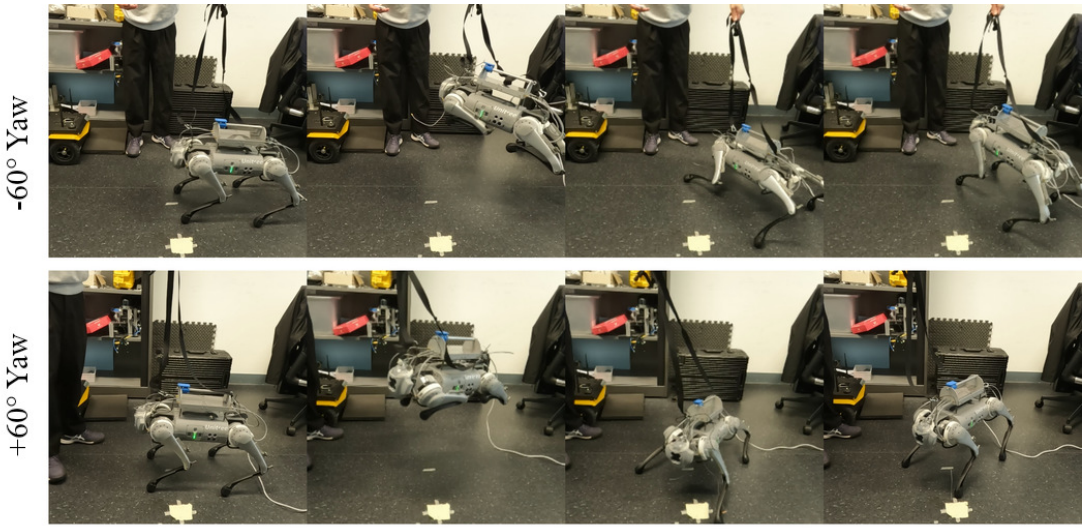

Fig. 13. Experiments showing in-place hops with desired $\pm 60°$ yaw change.

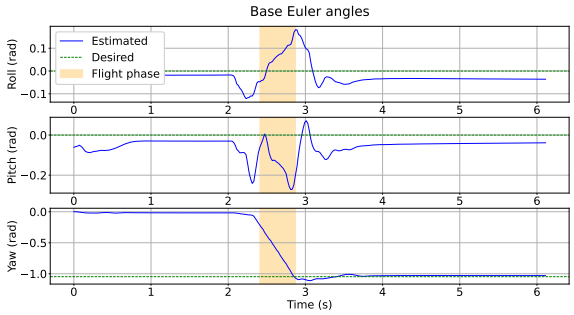

Fig. 14. Roll, pitch, yaw angles for the $-60°$ in-place turn. The measured values are shown in blue and the desired - with a green dashed line.

TABLE IV
SIMULATION RESULTS FROM THE 4000 TRIALS ACROSS THE WHOLE JUMPING RANGE. THE QUANTITATIVE DATA IS PRESENTED FOR PURELY FORWARD JUMPS AND FOR DIAGONAL JUMPS.

Forward Distance	0.00 - 0.24	0.25 - 0.49	0.50 - 0.74	0.75 - 0.99
Success Rate Forward	1	1	1	1
Success Rate Diagonal	1	1	0.999	0.981
Mean Error Forward (cm)	2.853	4.171	9.270	17.815
Mean Error Diagonal (cm)	6.388	3.306	8.527	17.423

desired yaw orientation, as indicated in the Euler angles plots in Fig. 14. Finally, we evaluated the policy across the whole jumping range in simulation and recorded the success rate and tracking metrics, as shown in Fig. 15. As can be seen from the left plot, the tracking error is lowest for narrow jumps of forward distance up to 50cm. As both the longitudinal and lateral distances increase, so does the final landing error. Interestingly, the majority of failed environments asymmetrically occur in the lower right corner of the plot. We consider failed those environments that have been terminated due to contact of any body part (other than the foot) with the ground. The right plot in Fig. 15 shows the same data but grouped by total desired distance vs actual achieved distance. From this view, it can be seen that the data closely follow the $45°$ line (i.e. ideal performance) for the smaller jumps with the gradient slowly decreasing after 50cm. It is worth mentioning that we recorded better tracking performance on the real hardware for the larger jumps, and more undershooting for the shorter distances. We hypothesise that this is due to a sim-to-real gap between the simulated joint friction model and real friction. We summarise these results in Table IV, grouped by the type of jump and the forward distance. As we sampled from the same lateral range regardless of the distance, the policy exhibited a larger error when performing short-distance diagonal jumps. Such jumps are more challenging and would often require the robot to turn a large yaw angle while also jumping to the side. In Fig. 16 we show the maximum height achieved during the jump as a function of the distance of the jump. A noticeable trade-off can be observed, where upwards momentum is traded for forward momentum.

E. Rough terrain

We were interested in evaluating how well the policy performs in the presence of environmental disturbances, despite not being trained on uneven or rough ground. To this end, we ran several experiments where small obstacles were scattered around the robot, blindly jumping from and onto a box, and jumping from asphalt onto a soft grassy terrain. As shown by the top two time-lapses in Fig. 17, the policy showed strong robustness to both soft and stiff objects that could (and did) slip under the feet of the robot. The quadruped easily executed the desired jump starting from the 4cm box, as well as landing on it. It is worth emphasising that such scenarios were not seen in simulation, as the second-stage curriculum policy had only been trained on flat and non-discrete terrain. The policy is also robust to changes in the ground friction, hardness and restitution, as can be seen from the final graphic in Fig. 17

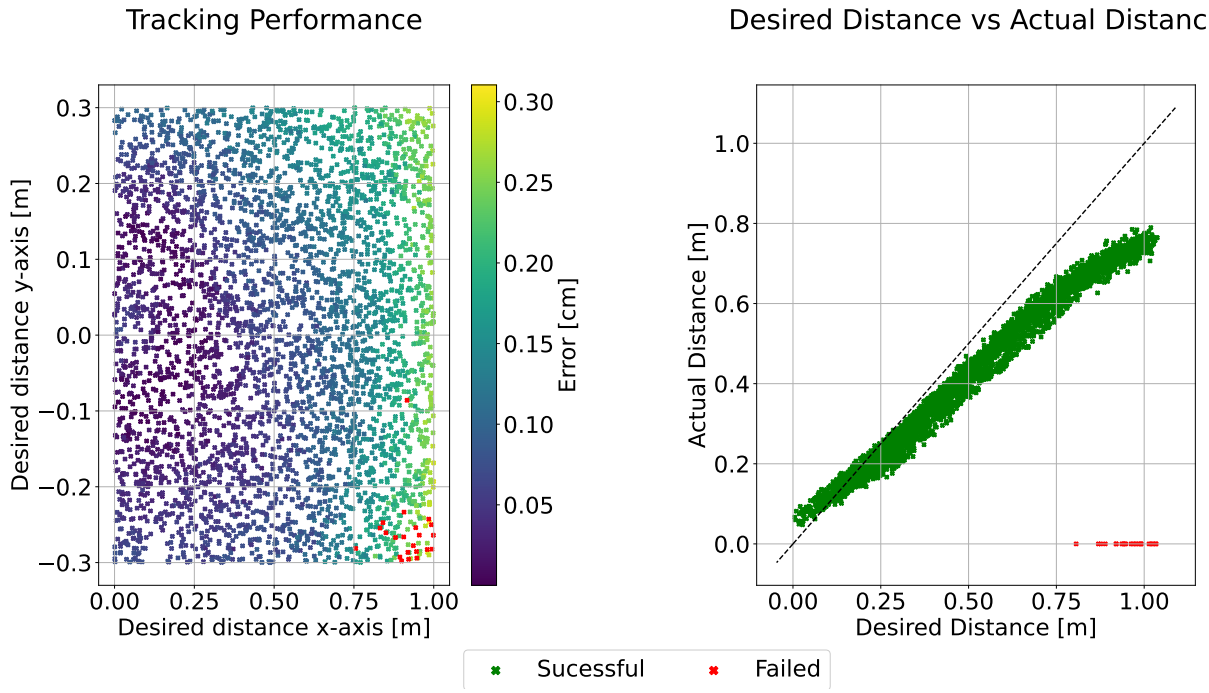

Fig. 15. Tracking performance as a function of the desired X- and Y-axis jumping distances, with the error (in cm) shown by the colour gradient (left); and the tracking performance in terms of overall desired vs actual jumping distance (right). The environments that have been terminated are shown in red, and the black $45°$ dashed line indicates the ideal tracking performance. Data is gathered from 4000 trials across the whole jumping range.

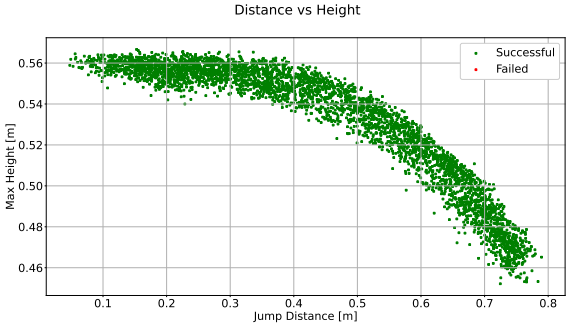

Fig. 16. Trade-off between maximum height during the jump and the final landing distance.

where the robot performs a 40cm jump from hard asphalt onto soft earth.

F. Continuous jumping

Next, we tested the policy on a continuous jumping task outdoors, where a new command of a 40cm forward jump is given following each jump without resetting the robot states. This is occasionally simulated throughout training (10% of robots that landed successfully do not get reset before the next episode), but not targeted by any particular rewards. This experiment, as shown in Fig. 18, shows that the policy is robust enough to execute a jump from a variety of different initial states. Despite the fact that the soft ground causes some hip angle deviation upon landing, the robot was able to execute at least nine consecutive jumps.

G. Forward jumping with obstacles

In this subsection, we validated the performance after completing the last curriculum stage, namely training in the presence of obstacles and conditioning the policy on the general dimensions and location of these obstacles. After the last training stage, the policy was robust enough to jump on or over obstacles of up to 10cm. In Fig. 19 we first show the performance of the forward jumping policy π_{II} (top row) when prompted to jump on a box-like obstacle of 10cm. As can be seen, the robot does not have enough height to clear the obstacle and only partially lands on it, after which it falls backwards. Similarly, in row 2, the robot hits the thin barrier and crashes. On the other hand, the final policy π_{III} fully lands on the box and returns to the nominal configuration (third row), and successfully jumps over the barrier (fourth row). In most of the jumps, the robot lands quite closely to the ground but does not register any body contact with it. Overall the policy was less smooth than π_{II}, which we attribute to the agent prioritising jump height over smoothness. It is interesting to note that the robot adopts a more downward pitch style of jumping in this stage. We deployed the policy on the hardware and tested two of the scenarios - jumping over a 10cm tall thin obstacle and landing on a 10cm box. The sim-to-real gap here was stronger than for the second stage policy. The robot needs to leap over 70cm due to the size of the obstacles while also maintaining a large height to avoid hitting them. As seen in the top row of Fig. 20 this was especially difficult for the barrier-like obstacle where the robot had to jump 80cm. The robot succeeded in crossing the barrier, however, its legs did touch it after landing. Better performance was observed when jumping onto the box, as a shorter forward distance meant the robot

Fig. 17. Several experiments showcasing the robustness of the policy π_{II} to variations in the terrain: jumping across discrete hard and soft objects (rows 1 and 2), blindly jumping from and onto a small box (rows 3 and 4), and asphalt-to-grass jump (row 5).

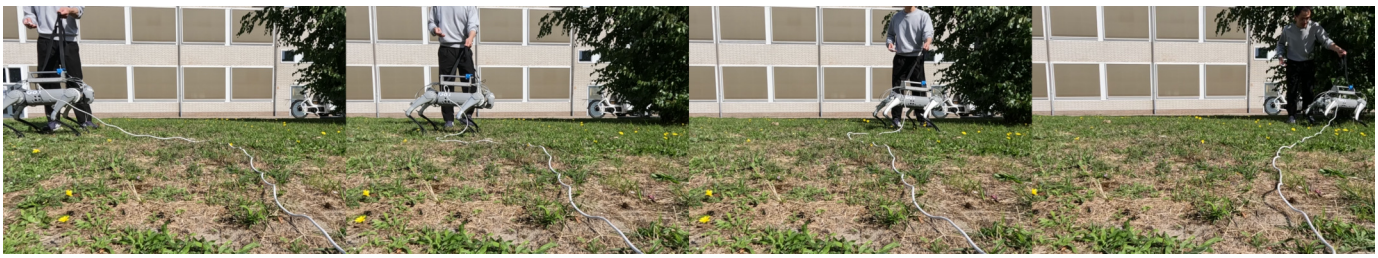

Fig. 18. Outdoors continuous jumping, consisting of 9 consecutive jumps of 0.4m each.

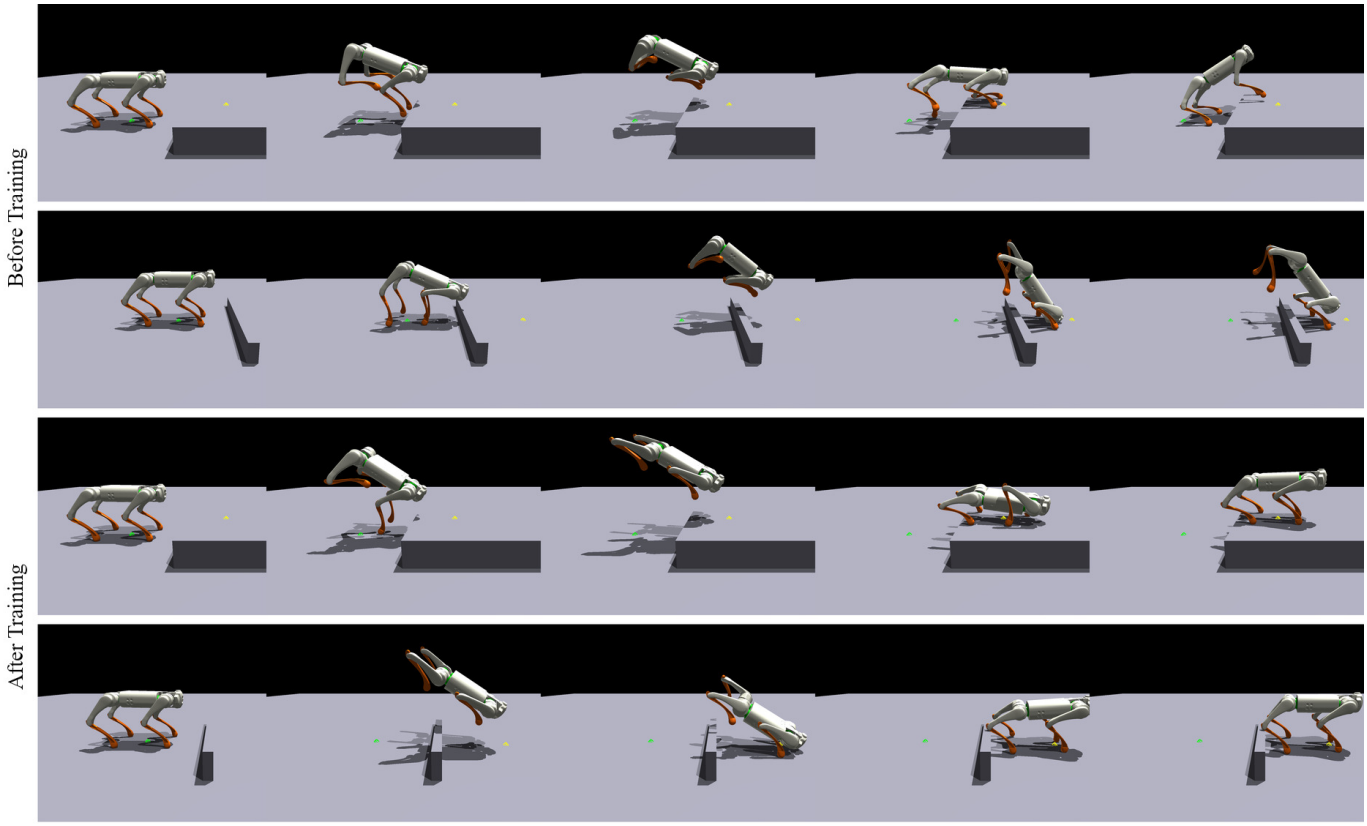

Fig. 19. Jumping onto and over obstacles, using Stage 2 policy π_{II} (first two row) and Stage 3 policy π_{III} (last two rows).

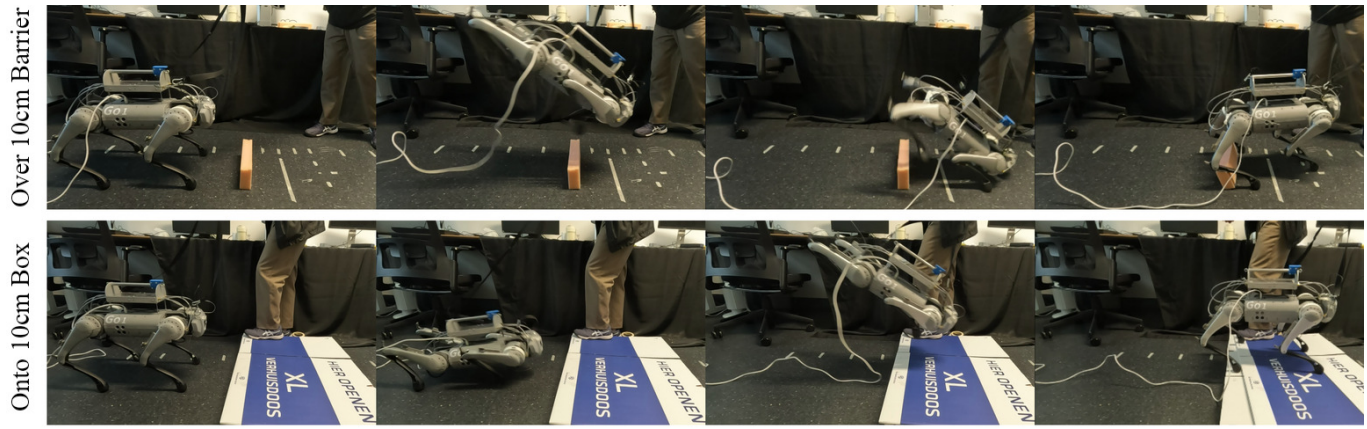

Fig. 20. Jumping over a 10cm tall, 5cm wide obstacle (top row) and jumping onto a 10cm tall box (bottom).

could achieve a larger height throughout flight. This trade-off between flight height and desired distance was shown in Fig. 16.

H. Jumping with springs

We trained a separate policy that incorporates the parallel elastic actuators, defined in Section IV-A. Some of the rewards were slightly tuned in order to accommodate the presence of springs, but the majority of them remained the same. In Fig. 21 and 22 we show the quantitative results for an 80cm forward jump. As can be seen in Fig. 21, the thigh and calf torques and joint velocities are much lower at the moment of landing, as some of the energy is absorbed by the springs. This is beneficial, as large jumps and falls can cause wear and damage to the motors. On the other hand, the PEA version requires higher torques during the squat-down phase (to compress the springs). Another disadvantage of the springs is that more energy is required to keep the legs close to the body in mid-air, as the springs are fully compressed in that pose. The base position and velocity profiles throughout the jump were very similar, however, the elastic version landed closer to the desired target while maintaining the same flight height.

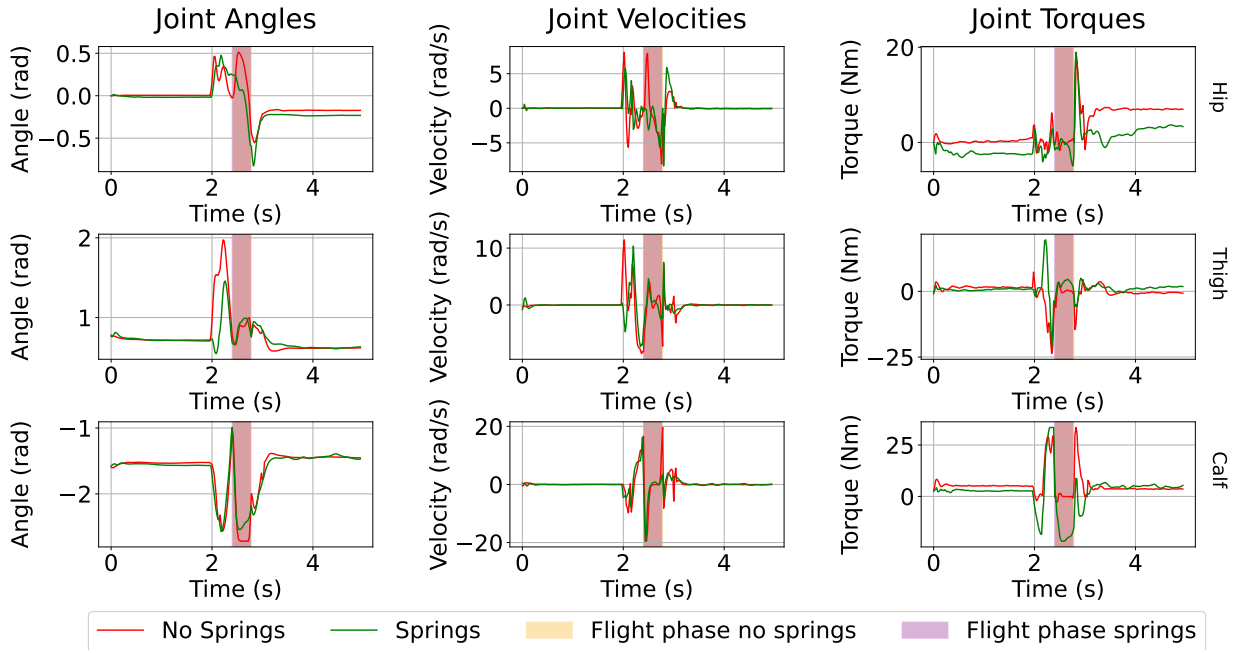

Fig. 21. Simulation comparison of the joints angles, velocities and torques between an 80cm forward jump with (green) and without (red) springs in parallel with the actuators.

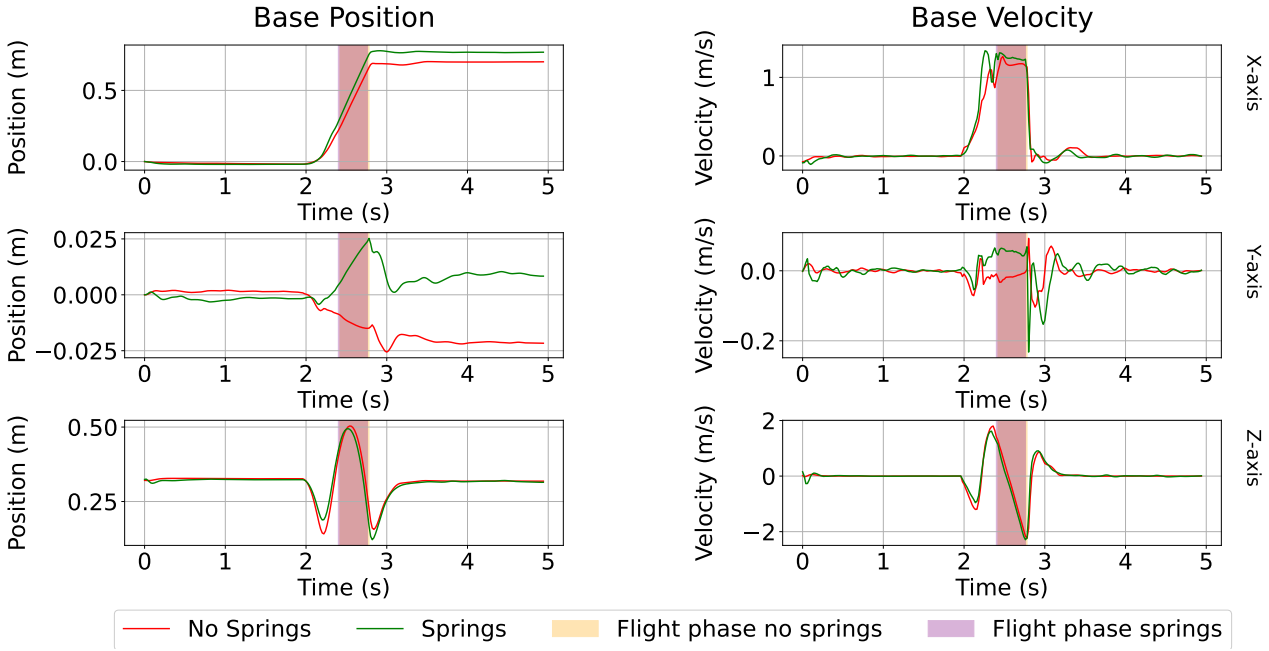

Fig. 22. Simulation comparison of the base position and velocity between an 80cm forward jump with (green) and without (red) springs in parallel with the actuators.

V. Discussion and Future work

A. Motor modelling

We observed some limitations caused by less accurate motor modelling of the system, as there was a significant discrepancy between the simulated and observed behaviour of the actuators, as was shown in Fig. 7. As we operate at the limits of the hardware, any such discrepancies can be exaggerated and drastically change the behaviour of the system. Furthermore, unlike general locomotion, jumping exhibits a long uncontrollable flight phase where these deviations accumulate and cannot be corrected. This was seen in our results, as the pitch angle across longer jumps deviated from the simulation results. The simulated motors do not represent phenomena like motor saturation, backlash and the effect of the transmission and gear reduction ratios. We estimated the joint friction and damping values experimentally by comparing the simulated and real behaviours. In the future, we would like to perform a more in-depth estimation of these parameters. Building a more complex and accurate model of the actuators can further improve the performance. However, this can be challenging, especially in the case of the Go1 where most of the motor specifications are not provided in detail by the manufacturer.

Several works have suggested learning the motor mapping between actions and output torque with a neural network instead, the so-called actuator networks. In our experiments, we tried such an approach but in our experience, it led to a worse performance. One explanation could be that the actuator network overfits to the motor dynamics at the operating range of the specific task. We found that training a network on data collected through locomotion or on only a couple of jumps did not capture the motor dynamics of jumping well. To successfully learn the relationship between desired and realised control inputs, capable of executing a wide range of jumps, we might need a more complex network or much more data, captured across a similarly wide range of jumps. To this end, it is a direction we would like to explore further.

B. Domain randomisation

One of the disadvantages of excessive domain randomisation is that it can lead to more general, but sub-optimal, behaviours [4, 8]. For example, walking on slippery surfaces like ice requires a different gait than walking on paved roads. A robot could learn a common locomotion pattern that can traverse both such environments, but that would come at the cost of speed or energy efficiency. On the other hand, if the policy of the agent is conditioned on certain environmental parameters, such as ground friction, it can then learn a suite of locomotion patterns and choose the best one depending on the type of terrain. This notion has been explored in literature, with some works using adaptation modules that modulate the behaviour depending on some perceived environmental conditions [8]; or by training a policy that learns to estimate these parameters given a history of proprioceptive information [9, 10, 14, 16].

Our method behaves similarly to the latter, as the policy is conditioned on a history of states. The robot could then implicitly reason about the state of its environment based on how its own states have evolved (and how they were expected to change). In contrast to other locomotion works, our approach consists of short episodes with a single jump. This means that the robot does not get the opportunity to explore its environment and (implicitly or explicitly) estimate its parameters. For this reason, we believe that there will not be much benefit in explicitly training a student-teacher network [8, 10, 14, 16] in the learning by cheating framework [73]. Li et al. [34] show experimental validations to the same result in their comparison study for bipedal jumping. Nevertheless, in the future, we would like to verify this experimentally by comparing the two methods.

C. Jumping with momentum

If we look at the behaviour of four-legged animals when executing a long-distance jump we can notice that they exhibit a four-legged contact phase, followed by an upward pitch and pure rear-leg contact at take-off. During landing a mirrored behaviour is observed - the body is pitching downwards and contact is first gained with the front legs. Previous model-based control works [24, 25] have incorporated this contact schedule into their optimisers. It would be interesting to investigate how such behaviour can be learned through DRL without supplying a reference trajectory, and furthermore, validate its benefits compared to the style of jumping exhibited here.

In addition, one of the limitations when jumping onto obstacles was caused by the fixed initial position - starting too close or too far from the obstacle might result in the robot being unsuccessful. In this work, we only consider jumping from a standstill, but as future work, it would be interesting to investigate the effect of allowing the robot to reposition itself prior to the jump or even transitioning from a walk or a run into a jump.

D. Parallel elastic actuators

In the comparison between the robot with parallel elastic actuators and the one without, we observed higher jumping accuracy and a smoother and softer landing for the former, at the cost of higher energy consumption during the squat-down and flight phases. However, we would like to further investigate the effect of the elastic elements. One direction of future work would be to optimise the stiffness of the springs and their rest positions throughout training. This can give better insights into the optimal placement of the springs to make them more energy efficient. Furthermore, it would be interesting to validate these results on the hardware. Due to the impact modelling mismatches mentioned earlier, the springs could be even more beneficial in the real world where the impacts were observed to be larger.

VI. Conclusion

In this work, we presented a curriculum-based end-to-end deep reinforcement learning approach, capable of successfully

learning a variety of precise short- and long-distance jumps, while also reaching the desired yaw upon landing. Unlike many existing methods, we have achieved this through a single policy, without the need for reference trajectories and additional imitation rewards.

Furthermore, through extensive domain randomisation we successfully deployed the policy onto the real system and closely matched the expected behaviour from the simulation. The system was robust to the noisy sensor data, especially the foot contact sensors and the velocity state estimates. The jumps exhibited high accuracy, both in simulation and on the hardware, in terms of tracking the desired landing position and orientation. Furthermore, our policy achieved a 90cm forward jump on the Unitree Go1 robot, a distance greater than those reported by other model- and learning-based controllers. We demonstrated additional outdoor tests, where the robot successfully performed nine consecutive jumps on soft grass, without previously encountering such environments in its training. In addition, we showed that simulating obstacles throughout training and conditioning the policy on their properties can enhance the mobility of the robot, allowing it to safely leap over or land on objects of up to 10cm.

We presented the limitations of our work, mainly caused by the discrepancy between simulated and real motors. In the future, we would like to improve upon this aspect, for example by training a more general actuator network. Having demonstrated that knowledge of the environment can improve the performance of agile locomotive controllers in the presence of obstacles, we would like to directly link exteroceptive sensing to the policy in the future, instead of manually specifying the obstacle parameters. Our preliminary work in using elastic elements in parallel with the actuators has shown promising results and it is a direction we plan to further explore, both for performing larger, more explosive jumps but also for softening the landing and reducing any potential damage to the robot.

REFERENCES

[1] 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," en, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid: IEEE, Oct. 2018, pp. 1–9, ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8594448. [Online]. Available: https://ieeexplore.ieee.org/document/8594448/.

[2] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *2014 IEEE-RAS International Conference on Humanoid Robots*, ISSN: 2164-0580, Nov. 2014, pp. 295–302. DOI: 10.1109/HUMANOIDS.2014.7041375.

[3] J. Carius, R. Ranftl, V. Koltun, and M. Hutter, "Trajectory Optimization With Implicit Hard Contacts," en, *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3316–3323, Oct. 2018, ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2018.2852785. [Online]. Available: https://ieeexplore.ieee.org/document/8403260/.

[4] J. Tan *et al.*, *Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*, arXiv:1804.10332 [cs], May 2018. DOI: 10.48550/arXiv.1804.10332. [Online]. Available: http://arxiv.org/abs/1804.10332.

[5] Z. Fu, A. Kumar, J. Malik, and D. Pathak, *Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots*, arXiv:2111.01674 [cs], Oct. 2021. DOI: 10.48550/arXiv.2111.01674. [Online]. Available: http://arxiv.org/abs/2111.01674.

[6] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, *Learning Agile Robotic Locomotion Skills by Imitating Animals*, arXiv:2004.00784 [cs], Jul. 2020. DOI: 10.48550/arXiv.2004.00784. [Online]. Available: http://arxiv.org/abs/2004.00784.

[7] J. Hwangbo *et al.*, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, Jan. 2019, Publisher: American Association for the Advancement of Science. DOI: 10.1126/scirobotics.aau5872. [Online]. Available: https://www.science.org/doi/full/10.1126/scirobotics.aau5872.

[8] A. Kumar, Z. Fu, D. Pathak, and J. Malik, *RMA: Rapid Motor Adaptation for Legged Robots*, arXiv:2107.04034 [cs], Jul. 2021. DOI: 10.48550/arXiv.2107.04034. [Online]. Available: http://arxiv.org/abs/2107.04034.

[9] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, eabc5986, Oct. 2020, Publisher: American Association for the Advancement of Science. DOI: 10.1126/scirobotics.abc5986. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abc5986.

[10] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, *Rapid Locomotion via Reinforcement Learning*, arXiv:2205.02824 [cs], May 2022. DOI: 10.48550/arXiv.2205.02824. [Online]. Available: http://arxiv.org/abs/2205.02824.

[11] Z. Xie, H. Y. Ling, N. H. Kim, and M. van de Panne, "ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills," en, *Computer Graphics Forum*, vol. 39, no. 8, pp. 213–224, 2020, ISSN: 1467-8659. DOI: 10.1111/cgf.14115. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14115.

[12] N. Rudin, D. Hoeller, M. Bjelonic, and M. Hutter, *Advanced Skills by Learning Locomotion and Local Navigation End-to-End*, arXiv:2209.12827 [cs], Sep. 2022. DOI: 10.48550/arXiv.2209.12827. [Online]. Available: http://arxiv.org/abs/2209.12827.

[13] Z. Fu, A. Kumar, A. Agarwal, H. Qi, J. Malik, and D. Pathak, *Coupling Vision and Proprioception for Navigation of Legged Robots*, arXiv:2112.02094 [cs], Jul. 2022. DOI: 10.48550/arXiv.2112.02094. [Online]. Available: http://arxiv.org/abs/2112.02094.

[14] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, eabk2822, Jan. 2022, Publisher: Ameri-

can Association for the Advancement of Science. DOI: 10.1126/scirobotics.abk2822. [Online]. Available: https://www.science.org/doi/full/10.1126/scirobotics.abk2822.

[15] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "RLOC: Terrain-Aware Legged Locomotion Using Reinforcement Learning and Optimal Control," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 2908–2927, Oct. 2022, Conference Name: IEEE Transactions on Robotics, ISSN: 1941-0468. DOI: 10.1109/TRO.2022.3172469.

[16] G. B. Margolis and P. Agrawal, *Walk These Ways: Tuning Robot Control for Generalization with Multiplicity of Behavior*, arXiv:2212.03238 [cs, eess], Dec. 2022. DOI: 10.48550/arXiv.2212.03238. [Online]. Available: http://arxiv.org/abs/2212.03238.

[17] G. B. Margolis *et al.*, *Learning to Jump from Pixels*, arXiv:2110.15344 [cs], Oct. 2021. DOI: 10.48550/arXiv.2110.15344. [Online]. Available: http://arxiv.org/abs/2110.15344.

[18] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, *Legged Locomotion in Challenging Terrains using Egocentric Vision*, arXiv:2211.07638 [cs, eess], Nov. 2022. DOI: 10.48550/arXiv.2211.07638. [Online]. Available: http://arxiv.org/abs/2211.07638.

[19] W. Yu *et al.*, "Visual-Locomotion: Learning to Walk on Complex Terrains with Vision," en, in *Proceedings of the 5th Conference on Robot Learning*, ISSN: 2640-3498, PMLR, Jan. 2022, pp. 1291–1302. [Online]. Available: https://proceedings.mlr.press/v164/yu22a.html.

[20] M. Chignoli and S. Kim, *Online Trajectory Optimization for Dynamic Aerial Motions of a Quadruped Robot*, arXiv:2110.06330 [cs], Oct. 2021. DOI: 10.48550/arXiv.2110.06330. [Online]. Available: http://arxiv.org/abs/2110.06330.

[21] H.-W. Park, P. M. Wensing, and S. Kim, "Jumping over obstacles with MIT Cheetah 2," en, *Robotics and Autonomous Systems*, vol. 136, p. 103 703, Feb. 2021, ISSN: 0921-8890. DOI: 10.1016/j.robot.2020.103703. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889020305431.

[22] 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)*, May 2022, pp. 4635–4641. DOI: 10.1109/ICRA46639.2022.9812120.

[23] 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, arXiv:2106.09357 [cs], ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2021.3084374. [Online]. Available: http://arxiv.org/abs/2106.09357.

[24] 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)*, ISSN: 2577-087X, May 2019, pp. 7448–7454. DOI: 10.1109/ICRA.2019.8794449.

[25] C. Nguyen and Q. Nguyen, *Contact-timing and Trajectory Optimization for 3D Jumping on Quadruped Robots*, 2022. [Online]. Available: https://arxiv.org/abs/2110.06764.

[26] C. Mastalli *et al.*, *Agile Maneuvers in Legged Robots: A Predictive Control Approach*, arXiv:2203.07554 [cs, eess], Jul. 2022. DOI: 10.48550/arXiv.2203.07554. [Online]. Available: http://arxiv.org/abs/2203.07554.

[27] I. Chatzinikolaidis and Z. Li, "Trajectory Optimization of Contact-rich Motions using Implicit Differential Dynamic Programming," English, *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2626–2633, Apr. 2021, Publisher: Institute of Electrical and Electronics Engineers Inc., ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3061341. [Online]. Available: https://www.research.ed.ac.uk/en/publications/trajectory-optimization-of-contact-rich-motions-using-implicit-di.

[28] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," en, *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, Jan. 2014, ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913506757. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364913506757.

[29] I. Chatzinikolaidis, Y. You, and Z. Li, "Contact-Implicit Trajectory Optimization using an Analytically Solvable Contact Model for Locomotion on Variable Ground," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6357–6364, Oct. 2020, arXiv:2007.11261 [cs], ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.3010754. [Online]. Available: http://arxiv.org/abs/2007.11261.

[30] 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, 143:1–143:14, Jul. 2018, ISSN: 0730-0301. DOI: 10.1145/3197517.3201311. [Online]. Available: https://doi.org/10.1145/3197517.3201311.

[31] S. Schaal, "Learning from Demonstration," in *Advances in Neural Information Processing Systems*, vol. 9, MIT Press, 1996. [Online]. Available: https://papers.nips.cc/paper/1996/hash/68d13cf26c4b4f4f932e3eff990093ba-Abstract.html.

[32] 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, ISSN: 2296-9144. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.854212.

[33] Y. Fuchioka, Z. Xie, and M. van de Panne, *OPT-Mimic: Imitation of Optimized Trajectories for Dynamic Quadruped Behaviors*, arXiv:2210.01247 [cs], Nov. 2022. DOI: 10.48550/arXiv.2210.01247. [Online]. Available: http://arxiv.org/abs/2210.01247.

[34] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, *Robust and Versatile Bipedal Jumping Control through Multi-Task Reinforcement Learning*, en, arXiv:2302.09450 [cs, eess], Feb. 2023. [Online]. Available: http://arxiv.org/abs/2302.09450.

[35] X. Huang *et al.*, *Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning*, arXiv:2210.04435 [cs, eess], Oct. 2022. DOI: 10.48550/arXiv.2210.04435. [Online]. Available: http://arxiv.org/abs/2210.04435.

[36] Q. Yao *et al.*, *Imitation and Adaptation Based on Consistency: A Quadruped Robot Imitates Animals from Videos Using Deep Reinforcement Learning*, arXiv:2203.05973 [cs], Mar. 2022. DOI: 10.48550/arXiv.2203.05973. [Online]. Available: http://arxiv.org/abs/2203.05973.

[37] F. Yu, R. Batke, J. Dao, J. Hurst, K. Green, and A. Fern, *Dynamic Bipedal Maneuvers through Sim-to-Real Reinforcement Learning*, arXiv:2207.07835 [cs], Jul. 2022. DOI: 10.48550/arXiv.2207.07835. [Online]. Available: http://arxiv.org/abs/2207.07835.

[38] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, *Learning Agile Skills via Adversarial Imitation of Rough Partial Demonstrations*, arXiv:2206.11693 [cs], Nov. 2022. DOI: 10.48550/arXiv.2206.11693. [Online]. Available: http://arxiv.org/abs/2206.11693.

[39] J. Ho and S. Ermon, *Generative Adversarial Imitation Learning*, arXiv:1606.03476 [cs], Jun. 2016. DOI: 10.48550/arXiv.1606.03476. [Online]. Available: http://arxiv.org/abs/1606.03476.

[40] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, *AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control*, arXiv:2104.02180 [cs], May 2022. DOI: 10.1145/3450626.3459670. [Online]. Available: http://arxiv.org/abs/2104.02180.

[41] A. Escontrela *et al.*, *Adversarial Motion Priors Make Good Substitutes for Complex Reward Functions*, arXiv:2203.15103 [cs], Mar. 2022. DOI: 10.48550/arXiv.2203.15103. [Online]. Available: http://arxiv.org/abs/2203.15103.

[42] E. Vollenweider, M. Bjelonic, V. Klemm, N. Rudin, J. Lee, and M. Hutter, *Advanced Skills through Multiple Adversarial Motion Priors in Reinforcement Learning*, arXiv:2203.14912 [cs], Mar. 2022. DOI: 10.48550/arXiv.2203.14912. [Online]. Available: http://arxiv.org/abs/2203.14912.

[43] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics*, vol. 36, no. 4, 41:1–41:13, Jul. 2017, ISSN: 0730-0301. DOI: 10.1145/3072959.3073602. [Online]. Available: https://doi.org/10.1145/3072959.3073602.

[44] X. B. Peng and M. van de Panne, "Learning locomotion skills using DeepRL: Does the choice of action space matter?" In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, ser. SCA '17, New York, NY, USA: Association for Computing Machinery, Jul. 2017, pp. 1–13, ISBN: 978-1-4503-5091-4. DOI: 10.1145/3099564.3099567. [Online]. Available: https://doi.org/10.1145/3099564.3099567.

[45] S. Bohez *et al.*, *Imitate and Repurpose: Learning Reusable Robot Movement Skills From Human and Animal Behaviors*, arXiv:2203.17138 [cs], Mar. 2022. DOI: 10.48550/arXiv.2203.17138. [Online]. Available: http://arxiv.org/abs/2203.17138.

[46] G. Bellegarda and Q. Nguyen, *Robust Quadruped Jumping via Deep Reinforcement Learning*, arXiv:2011.07089 [cs, eess], Mar. 2021. DOI: 10.48550/arXiv.2011.07089. [Online]. Available: http://arxiv.org/abs/2011.07089.

[47] Z. Yin, Z. Yang, M. van de Panne, and K. Yin, *Discovering Diverse Athletic Jumping Strategies*, arXiv:2105.00371 [cs], May 2021. DOI: 10.48550/arXiv.2105.00371. [Online]. Available: http://arxiv.org/abs/2105.00371.

[48] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09, New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 41–48, ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553380. [Online]. Available: http://doi.org/10.1145/1553374.1553380.

[49] X. Wang, Y. Chen, and W. Zhu, *A Survey on Curriculum Learning*, arXiv:2010.13166 [cs], Mar. 2021. DOI: 10.48550/arXiv.2010.13166. [Online]. Available: http://arxiv.org/abs/2010.13166.

[50] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, *Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions*, arXiv:1901.01753 [cs], Feb. 2019. DOI: 10.48550/arXiv.1901.01753. [Online]. Available: http://arxiv.org/abs/1901.01753.

[51] V. Mnih *et al.*, *Playing Atari with Deep Reinforcement Learning*, arXiv:1312.5602 [cs], Dec. 2013. DOI: 10.48550/arXiv.1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602.

[52] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems*, vol. 12, MIT Press, 1999. [Online]. Available: https://papers.nips.cc/paper/1999/hash/464d828b85b0bed98e80ade0a5c43b0f-Abstract.html.

[53] S. M. Kakade, "A Natural Policy Gradient," in *Advances in Neural Information Processing Systems*, vol. 14, MIT Press, 2001. [Online]. Available: https://papers.nips.cc/paper/2001/hash/4b86abe48d358ecf194c56c69108433e-Abstract.html.

[54] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, arXiv:1707.06347 [cs], Aug. 2017. DOI: 10.48550/

arXiv.1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[55] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust Region Policy Optimization*, arXiv:1502.05477 [cs], Apr. 2017. DOI: 10.48550/arXiv.1502.05477. [Online]. Available: http://arxiv.org/abs/1502.05477.

[56] Z. Li *et al.*, "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 2577-087X, May 2021, pp. 2811–2817. DOI: 10.1109/ICRA48506.2021.9560769.

[57] L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine, *Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World*, arXiv:2110.05457 [cs], Oct. 2021. DOI: 10.48550/arXiv.2110.05457. [Online]. Available: http://arxiv.org/abs/2110.05457.

[58] L. Smith, I. Kostrikov, and S. Levine, *A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning*, arXiv:2208.07860 [cs], Aug. 2022. DOI: 10.48550/arXiv.2208.07860. [Online]. Available: http://arxiv.org/abs/2208.07860.

[59] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, ISSN: 2577-087X, May 2018, pp. 3803–3810. DOI: 10.1109/ICRA.2018.8460528.

[60] J. Siekmann *et al.*, *Learning Memory-Based Control for Human-Scale Bipedal Locomotion*, arXiv:2006.02402 [cs], Jun. 2020. DOI: 10.48550/arXiv.2006.02402. [Online]. Available: http://arxiv.org/abs/2006.02402.

[61] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning," en, in *Proceedings of the 5th Conference on Robot Learning*, ISSN: 2640-3498, PMLR, Jan. 2022, pp. 91–100. [Online]. Available: https://proceedings.mlr.press/v164/rudin22a.html.

[62] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent Training of a Control Policy and a State Estimator for Dynamic and Robust Legged Locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4630–4637, Apr. 2022, Conference Name: IEEE Robotics and Automation Letters, ISSN: 2377-3766. DOI: 10.1109/LRA.2022.3151396.

[63] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World*, arXiv:1703.06907 [cs], Mar. 2017. DOI: 10.48550/arXiv.1703.06907. [Online]. Available: http://arxiv.org/abs/1703.06907.

[64] F. Sadeghi and S. Levine, *CAD2RL: Real Single-Image Flight without a Single Real Image*, arXiv:1611.04201 [cs], Jun. 2017. DOI: 10.48550/arXiv.1611.04201. [Online]. Available: http://arxiv.org/abs/1611.04201.

[65] S. Gangapurwala, L. Campanaro, and I. Havoutis, *Learning Low-Frequency Motion Control for Robust and Dynamic Robot Locomotion*, arXiv:2209.14887 [cs], Sep. 2022. DOI: 10.48550/arXiv.2209.14887. [Online]. Available: http://arxiv.org/abs/2209.14887.

[66] *Unitree Go1 – UnitreeRobotics*. [Online]. Available: https://shop.unitree.com/products/unitreeyushutechnologydog-artificial-intelligence-companion-bionic-companion-intelligent-robot-go1-quadruped-robot-dog.

[67] F. Bjelonic *et al.*, "Learning-based Design and Control for Quadrupedal Robots with Parallel-Elastic Actuators," en, *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1611–1618, Mar. 2023, arXiv:2301.03509 [cs], ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2023.3234809. [Online]. Available: http://arxiv.org/abs/2301.03509.

[68] V. Makoviychuk *et al.*, *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*, arXiv:2108.10470 [cs], Aug. 2021. DOI: 10.48550/arXiv.2108.10470. [Online]. Available: http://arxiv.org/abs/2108.10470.

[69] Y. Yang *et al.*, *CAJun: Continuous Adaptive Jumping using a Learned Centroidal Controller*, arXiv:2306.09557 [cs], Jun. 2023. DOI: 10.48550/arXiv.2306.09557. [Online]. Available: http://arxiv.org/abs/2306.09557.

[70] L. Smith *et al.*, *Learning and Adapting Agile Locomotion Skills by Transferring Experience*, en, arXiv:2304.09834 [cs], Apr. 2023. [Online]. Available: http://arxiv.org/abs/2304.09834.

[71] K. Caluwaerts *et al.*, *Barkour: Benchmarking Animal-level Agility with Quadruped Robots*, en, May 2023. [Online]. Available: https://arxiv.org/abs/2305.14654v1.

[72] Y. Yang, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Continuous Versatile Jumping Using Learned Action Residuals," en, in *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*, ISSN: 2640-3498, PMLR, Jun. 2023, pp. 770–782. [Online]. Available: https://proceedings.mlr.press/v211/yang23b.html.

[73] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, *Learning by Cheating*, arXiv:1912.12294 [cs], Dec. 2019. DOI: 10.48550/arXiv.1912.12294. [Online]. Available: http://arxiv.org/abs/1912.12294.

APPENDIX A
REWARD SCALES

Table V shows the scales for each term in the reward function, for each of the three training stages respectively.

APPENDIX B
PPO HYPERPARAMETERS

In this section we present the hyperparameters for the PPO reinforcement algorithm (see Table VI).

TABLE V
REWARD SCALES FOR THE THREE CURRICULUM STAGES.

Name	Scale Stage 1	Scale Stage 2	Scale Stage 3	Kernel σ
Landing position w_p	4.0	30.0	30.0	0.05
Landing orientation w_{ori}	4.0	30.0	30.0	0.05
Max height w_h	40.0	100.0	100.0	0.05
Termination w_{term}	-0.04	-0.04	-0.04	0.0
Jumping w_{jump}	1.0	4.0	4.0	0.05
Position Tracking $w_{p,tr}$	0.06	0.06	0.06	0.001
Orientation Tracking $w_{ori,tr}$	0.12	0.12	0.12	0.05
Base height squat $w_{p_z,sq}$	0.24	0.24	0.24	0.005
Base height flight $w_{p_z,fl}$	2.0	2.0	2.0	0.1
Base height landing w_{h_st}	0.4	0.4	0.4	0.005
Tracking Linear vel $w_{v_{x,y}}$	0.1	0.6	0.6	0.05
Tracking Angular vel w_ω	0.01	0.1	0.1	0.05
Feet clearance w_{feet}	-0.4	-0.4	-0.4	0.0
Symmetry w_{sym}	-0.06	-0.06	-0.06	0.05
Nominal pose w_q	0.16	0.24	0.24	0.1
Energy usage w_{energy}	-2e-4	-2e-4	-2e-4	0.0
Change of contact $w_{contact}$	0.0	0.4	0.4	0.0
Feet contact forces w_{F_c}	-0.1	-0.1	-0.1	0.0
Action rate w_a	-2e-3	-2e-3	-2e-3	0.0
Action rate second order $w_{\dot{a}}$	-2e-3	-2e-3	-2e-3	0.0
Joint acceleration $w_{\ddot{q}}$	-2e-8	-2e-8	-2e-8	0.0
Joint limits $w_{q_{lim}}$	-0.2	-0.2	-0.2	0.0

TABLE VI
PPO HYPERPARAMETER CHOICE.

Hyperparameter	Value
Discount Factor	0.99
Value Loss Coefficient (α_1)	1.0
Entropy Coefficient (α_2)	0.01
Ratio Clip	0.2
Learning rate	1e-3
GAE parameter	0.95
Num of epochs	5
Num of mini batches	4
Num of envs	4096
Num of steps per env	30