Curriculum-Based Reinforcement Learning for Quadrupedal Jumping
A Reference-Free Design

Atanassov, Vassil; Ding, Jiatao; Kober, Jens; Havoutis, Ioannis; Santina, Cosimo Della

DOI
10.1109/MRA.2024.3487325

Publication date
2025

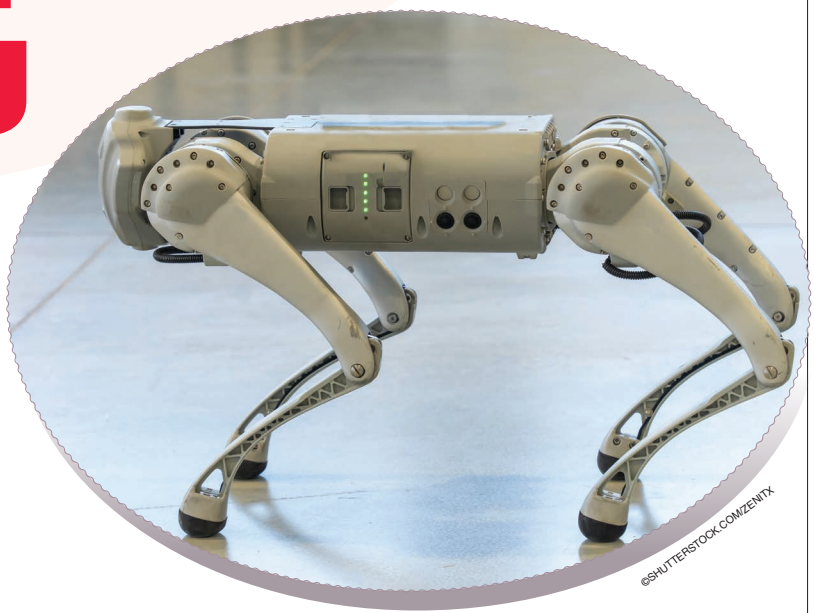
Document Version
Final published version

Published in
IEEE Robotics and Automation Magazine

Curriculum-Based Reinforcement Learning for Quadrupedal Jumping

A Reference-Free Design

By Vassil Atanassov, Jiatao Ding, Jens Kober, Ioannis Havoutis, and Cosimo Della Santina



Deep reinforcement learning (DRL) has emerged as a promising solution to mastering explosive and versatile quadrupedal jumping skills. However, current DRL-based frameworks usually rely on pre-existing reference trajectories obtained by capturing animal motions or transferring experience from existing controllers. This work aims to prove that learning dynamic jumping is possible without relying on imitating a reference trajectory by leveraging a curriculum design. Starting from a vertical in-place jump, we generalize the learned policy to forward and diagonal jumps and, finally, we learn to jump across obstacles. Conditioned on the desired landing location, orientation, and obstacle dimensions, the proposed approach yields a wide range of omnidirectional jumping motions in real-world experiments. In particular, we achieve a 90-cm forward jump, exceeding all previous records for similar robots. Additionally, the robot can reliably execute continuous jumping on soft grassy grounds, which is especially remarkable as such conditions were not included in the training stage.

Digital Object Identifier 10.1109/MRA.2024.3487325
Date of publication 20 November 2024; date of current version 14 June 2025.

INTRODUCTION

Through millions of years of evolution, legged animals have adapted to locomote in highly complex and discontinuous environments that widely exist in nature. Goats, for example, are capable of scaling nearly vertical mountainsides and jumping across chasms several times their body length. While many works have tackled dynamic locomotion [1], [2], [3], achieving such complex controlled behavior is still an open challenge.

Quadrupedal jumping has traditionally been investigated through model-based control, where an accurate model of the dynamical system is needed to generate optimal control inputs [4], [5], [6], [7]. In addition, these methods rely on various heuristics to render the approach feasible, limiting the search space and resulting in conservative performance.

In contrast to model-based optimization, model-free reinforcement learning (RL) has emerged as an effective alternative

that does not require expert knowledge for control engineering and tedious gain tuning. Especially, deep RL (DRL) has shown impressive generalization and robustness capabilities in executing locomotion tasks [1], [2], [8], [9], [10]. For quadrupedal jumping, a series of correct actions need to be taken for the robot to succeed. Paired with an inherently sparse reward structure (the robot has either jumped or not), it is exceptionally hard for the robot to learn, as most of its trials will fail. Current RL approaches tackle this by directly transferring skills from demonstrations [11], [12] or optimal controllers [13], [14], [15]. However, balancing the degree to which the agent should imitate the demonstration and generalize to new tasks is nontrivial.

In this work, we push robots to learn to jump on their own by combining curriculum learning (CL) with DRL, eliminating the reliability of precomputed motion references (Figure 1). By conditioning the policy on the desired landing location and orientation, our approach produces versatile jumping motions with just one single policy. Furthermore, by incorporating partial knowledge of the obstacles surrounding it, the robot learns different maneuvers adapted to complex real-world scenarios.

The main contributions are summarized as follows:

- We propose a curriculum-based DRL framework that is capable of learning jumping motions without requiring motion capture data or a reference trajectory.
- We generalize across a wide range of jumps with a single policy for both indoor and outdoor environments. With our method, the real robot can jump 90 cm forward, which, to the best of our knowledge, is the longest distance achieved on quadrupeds of a similar size. It has been demonstrated that continuous jumping across grassland and robust jumping across uneven terrains can be achieved in a zero-shot manner.
- We achieve jumping over obstacles and more complex terrains of various sizes by conditioning the policy on some partial environmental information in an additional curriculum stage.

In the "Related Work" section, we review the existing RL-based jumping controllers. In the following two sections, we separately present the curriculum design and DRL formulation. After extensively evaluating our method in the "Experimental Validation" section, we conclude this work with a thorough discussion in the final section.

RELATED WORK

RL FOR QUADRUPEDAL JUMPING

DRL is a promising solution for accomplishing jumping tasks by offloading the computational complexity to offline training. One approach to learning quadrupedal jumping is by learning from demonstrations, such as from trajectories generated through optimal control [13], [14], or hand-tuned reference motions [11], [12]. To address the challenges associated with the selection of relevant states to mimic and manage conflicting objectives, generative adversarial imitation learning has recently been widely adopted [16], [17], [18], even when dealing with partially incomplete demonstrations [11]. In [19], transfer learning is used to learn policies capable of diverse agile motions from a database of existing RL and model-based controllers. However, most imitation-based methods have so far shown a limited generalization capability

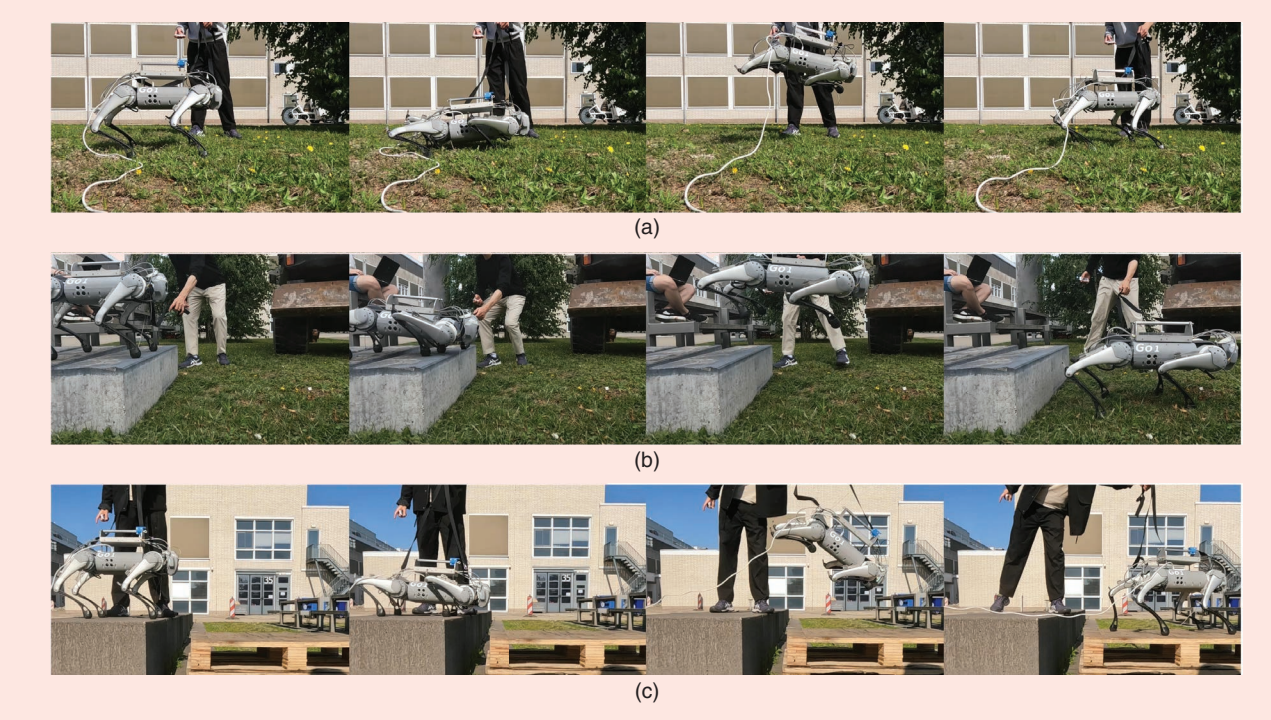


FIGURE 1. The Go1 robot (a) jumps across grassland, (b) jumps down onto grassland, and (c) jumps across a gap onto a lower box.

beyond the imitation domain. 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.

To reduce the dependency on a motion prior, [20] used a variational autoencoder to encapsulate motion capture data into a latent space and then combine it with a Bayesian diversity search to discover viable takeoff states. In [21], Margolis et al. trained a high-level motion-planning module to produce desired center of mass (CoM) trajectories for small hops, conditioned on visual inputs and then tracked by a model-based controller. In [13], deviations to reference trajectories generated by a nonlinear optimal trajectory [5] were learned, providing better generalization to out-of-training domains. Similarly, the researchers in [22] taught action residuals to a model-based controller to achieve continuous jumping. Another work focusing on continuous hopping [23] used a learned centroidal policy to output desired CoM trajectories, which are tracked by a quadratic programming-based ground reaction force controller. Rudin et al. [24] showed cat-like jumping in low gravity by using a more complex reward function without imitating motion clips. However, this approach has not yet been verified on Earth-like gravitational conditions. Recently, Vezzi et al. [25] proposed learning to jump by combining a first-stage evolution strategy with a second-stage DRL. Unlike [25], our approach offers reduced complexity by using proximal policy optimization (PPO) [26] for all curriculum stages, and it is capable of executing a range of jumps conditioned on the desired jumping length and orientation rather than a single distance-maximizing jump.

CL IN DYNAMIC QUADRUPEDAL LOCOMOTION

CL is a training framework that progressively provides more challenging data or tasks as the policy improves. As the name suggests, the idea behind the approach borrows from human education, where complex tasks are taught by breaking them into simpler parts.

In legged locomotion, CL has seen wide use, mainly in terms of terrain adaptation. Xie et al. [27] showed how an adaptive curriculum can be used to learn stepping-stone skills much more efficiently than other methods, like uniform sampling. Similarly, other automatic CL methods have been proposed to vary environmental parameters based on the performance of the agents [10] rather than using a manually specified curriculum. Hwangbo et al. [1] employed a curriculum of adaptive regularization reward scales. In [28], parkour locomotion skills were learned through a well-designed terrain curriculum with a single policy, which was then distilled to a exteroception-conditioned policy. Similar parkour skills were acquired in [29], but the method requires separate policies for each skill as well as a perception and navigation network, which greatly increases the computational complexity. Barkour [30] uses a similar approach but distills the specialist controllers into a single generalist transformer policy. To learn dynamic parkour skills, [31] adopted a two-stage curriculum, transitioning from soft to hard dynamic constraints in the second stage.

In contrast to all of these works, we propose a task-based curriculum, which alters the task structure and objectives rather than the reward scales or environment properties. Recently, [12] used multistage training 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 unified policy for versatile jumping motions.

CURRICULUM DESIGN

Defining and constraining the behavior of jumping across specific distances is challenging, as it combines two distinct behaviors: 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) toward the target point without actually jumping. To avoid converging to such undesired behavior, we use CL to decompose the problem into several simpler subtasks.

In our approach, we adopt two types of curricula—on a local difficulty level and on a task level, as can be seen in Figure 2. The former involves progressively (and automatically) making the environment more complex as the agent succeeds. In particular, upon successful jumps, we increase the range of desired jumping distances and obstacle 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 mastering the easier jumping skill, the policy is loaded onto the next task, which might be defined differently and contains a new set of rewards.

In the remainder, we describe each of these task-level and difficulty curricula in the progressive order of training.

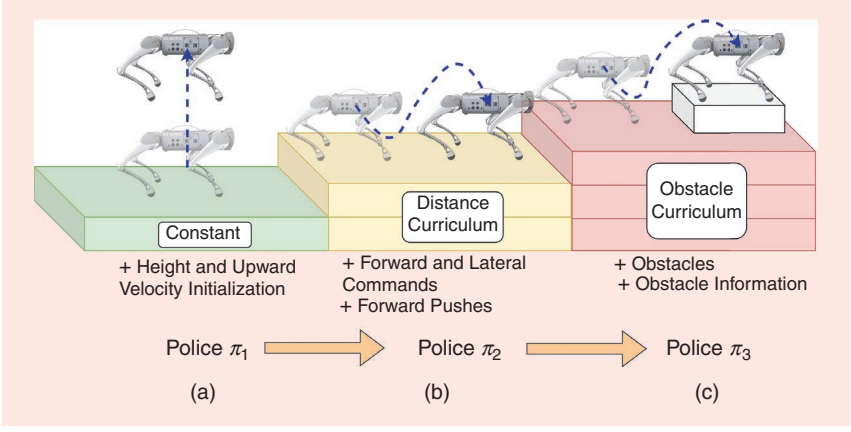


FIGURE 2. The curricula: (a) jumping in place, (b) long-distance jump, and (c) long-distance jump with obstacles. The latter two vary the jump distance/orientation and obstacle height, respectively.

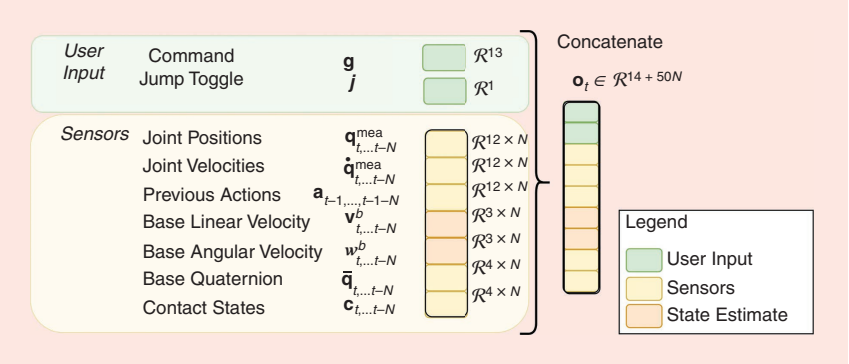


FIGURE 3. The definition of observations. The command $\mathbf{g}$ and jump toggle j are provided by the user, while the remaining observations are either directly read from the sensors or estimated using sensory data.

STAGE 1: JUMPING IN PLACE

Vertical jumping without traversing a certain horizontal distance, i.e., jumping in place, is the basic component of agile jumping. However, the lack of reference results in a learning problem with sparse rewards, given that the agent needs to first learn certain behaviors (e.g., squatting down and then pushing hard against the ground to take off) before it can reach the reward-rich states (i.e., being high in the air). As the robot does not experience these jumping-specific rewards initially, it is prone to converging to a local optimum, such as standing in place, where small rewards are collected safely.

To avoid getting stuck in this local optimum behavior, we adopt a modified form of the reference state initialization (RSI) technique [32]. In imitation learning, RSI initializes the agent at random points of the reference trajectory, allowing the agent to explore such reward-rich states before it has learned the actions necessary to reach them. As we do not use a reference trajectory, we instead modify RSI to sample a random height and upward velocity from a predefined range.

STAGE 2: LONG-DISTANCE JUMP

Once the robot has converged to a jumping-in-place behavior, we further train it to perform precise forward and diagonal jumps. The first part of the command vector $\mathbf{g} \in \mathcal{R}^{13}$ (see Figure 3) in the observations specifies the desired landing point and orientation to create a goal-conditioned policy. Similarly to the jumping-in-place subtask, 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 and landing yaw are sampled from a greater range.

STAGE 3: LONG-DISTANCE JUMP ACROSS OBSTACLES

Finally, we introduce obstacles in the environment. Without loss of generality, we choose three classes of obstacles, including thin barrier-like objects, box-shaped obstacles, and slopes. Depending on the desired landing pose and the obstacle location and type, the agent needs to either jump onto or over it. While it is possible to learn a general behavior that can accomplish this without any exteroception, such a behavior will be conservative, suboptimal, and potentially much less robust. With this in mind, we incorporate information about the distance to the center of the obstacles and its general dimensions (length, width, and height). In the real world, we manually specify these parameters.[1]

Similar to the previous stage, we start with obstacles of smaller height. Then, successful robots progress toward more challenging terrains, whereas failing ones are demoted to easier environments. To ensure that the robot remembers the previously learned behavior, we also randomly send a certain percentage of robots to jump on flat ground, as in stage 2.

DRL FORMULATION

This section details the DRL formulation, as illustrated in Figure 4. First, preliminaries are introduced. Then, we define the key components of goal-conditioned RL, including observations, actions, and reward functions. Finally, we introduce our domain randomization scheme to mitigate the simulation-to-reality (sim2real) gap.

PRELIMINARIES

RL infers a policy $\pi(a_t \mid s_t)$ of how to act by constantly interacting with the environment. The RL problem is typically formulated as a Markov decision process, 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 observation and the associated reward $\mathcal{R}_t$ that it has earned. Based on the observed state s_{t+1} and its policy $\pi(a_{t+1} \mid s_{t+1})$, the agent can then choose a new action a_{t+1}. In this way, the RL algorithm optimizes behaviors that yield high rewards. In goal-conditioned RL, the action policy can also be conditioned on specific goals, i.e., $\pi(a_t \mid s_t, g)$. Such a policy can be used to produce diverse behaviors depending on the specific command g, enabling the learning of multiple distinct behaviors under a single policy.

In this work, we formulate the following objective: finding a policy $\pi(a \mid s, g)$ that maximizes 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 maximizing the sum of discounted rewards J, known as the *return*, can be written as

$$\underset{\pi}{\text{argmax}} \quad J(\pi) = \mathbb{E}_{\tau \sim p^{\pi}(\tau)}\left[\sum_{t=0}^{T} \gamma^t R_t \,\middle|\, s_0 = s\right] \tag{1}$$

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. We optimize this objective

[1] A separate module that estimates obstacle dimensions could be utilized. One future work would be linking exteroceptive sensors to the policy and removing the parameterization of the world around the robot.

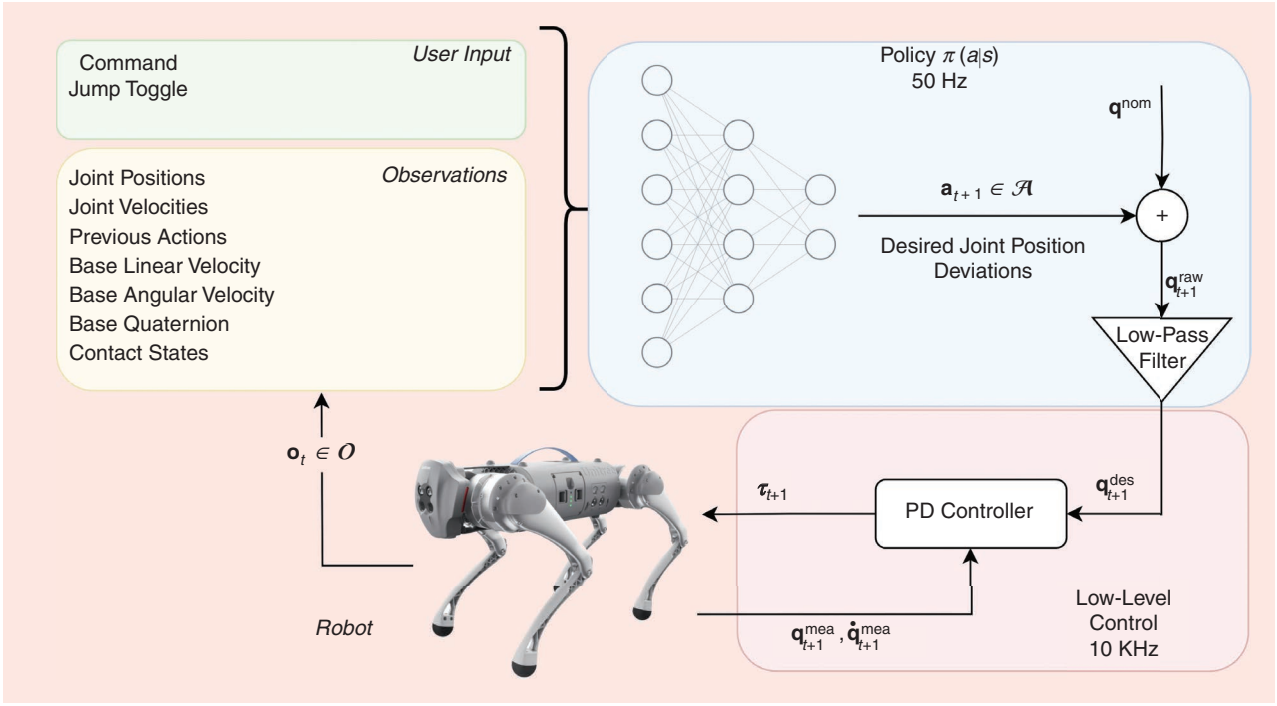


FIGURE 4. A control diagram of the system. The observations $\mathbf{o}_t$ include the user command (in green) and a history of system states (in yellow). The policy is parameterized by a neural network (shown in blue). The output actions $\mathbf{a}_{t+1}$ are added to the nominal joint angles $\mathbf{q}^{nom}$ and passed through a low-pass filter. The desired joint angles are then tracked via a PD controller, which computes torque commands. PD: proportional derivative.

using the PPO algorithm, with both the actor and the critic parameterized by a multilayer perceptron (MLP) network.

OBSERVATION AND ACTION SPACE

OBSERVATION SPACE

Using a memory of previous observations and actions allows the agent to implicitly reason about its own dynamics and the interaction with the environment [1], [10]. Here, we use a concatenated history of the last N steps as input to the policy.[2] As illustrated in Figure 4, the observation space consists of the historical base linear velocity $\mathbf{v} \in \mathbb{R}^{3\times N}$, base angular velocity $\boldsymbol{\omega} \in \mathbb{R}^{3\times N}$ (both in the base frame), joint position $\mathbf{q} \in \mathbb{R}^{12\times N}$, joint velocity $\dot{\mathbf{q}} \in \mathbb{R}^{12\times N}$, previous actions $\mathbf{a}_{t-1} \in \mathbb{R}^{12\times N}$, base orientation (as a quaternion) $\bar{\mathbf{q}} \in \mathbb{R}^{4\times N}$, and foot contact states $\mathbf{c} \in \mathbb{R}^{4\times N}$.

Note that our policy is also conditioned on the command $\mathbf{g} \in \mathbb{R}^{13}$ and jump toggle $j \in \{0, 1\}$ (see the green block in Figure 4). As illustrated in Figure 5, the command $\mathbf{g} \in [\Delta\mathbf{p}_{des}, \bar{\mathbf{q}}_{des}, \mathbf{p}_{obs}, \mathbf{dim}_{obs}]$ contains the desired landing position ($\Delta\mathbf{p}_{des} \in \mathbb{R}^3$), the desired landing orientation ($\bar{\mathbf{q}}_{des} \in \mathbb{R}^4$), the center of the obstacle ($\mathbf{p}_{obs} \in \mathbb{R}^3$ if present), and its dimensions ($\mathbf{dim}_{obs} \in \mathbb{R}^3$ including height, width, and length).[3] Due to the lack of long-term memory in the feedforward neural network, we use the jump toggle j to indicate whether the robot has already jumped, similar to [32]. However, in our case, the jump toggle also serves as a control switch, where the robot remains standing until its value is changed.

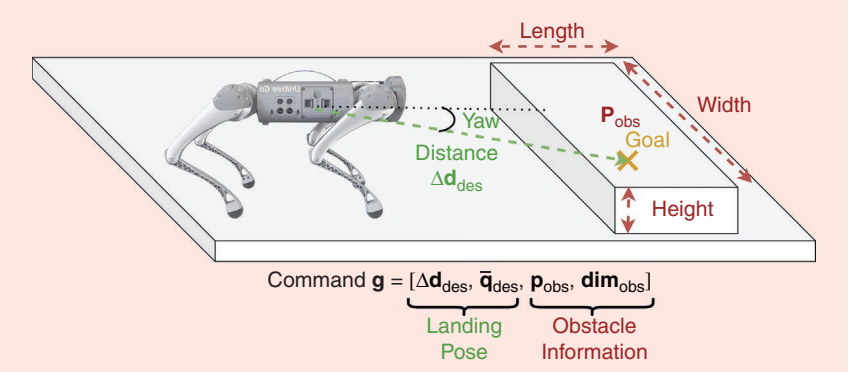


FIGURE 5. The command vector $\mathbf{g}$ for a forward jump onto an obstacle. In the first two training stages (π_I and π_{II}), where no obstacles are considered, the information of the obstacle is set to zero.

ACTION SPACE

Our policy generates the 12 actuated joint angles ($\mathbf{q}^{des} \in \mathbb{R}^{12}$) for jumping control. Particularly, we learn the deviations

[2] In practice, we found that using the last 20 steps is sufficient for the task while also being fast for training.

[3] In the training process, we sample the landing pose and obtain the obstacle parameters from the simulator. In the real world, the command vector is specified by the user.

TABLE 1. Rewards definition.

NAME	TYPE	STANCE	FLIGHT	LANDING
Landing position	Sparse	0	0	$w_{\mathbf{p}}(\mathrm{e}(-\Sigma\|\mathbf{p}_{\text{land}}-\mathbf{p}_{\text{des}}\|^2)/\sigma_{p,\text{land}})$
Landing orientation	Sparse	0	0	$w_{\text{ori}}(\mathrm{e}(-\|\log(\bar{\mathbf{q}}_{\text{land}}^{-1}*\bar{\mathbf{q}}_{\text{des}}\|^2)/\sigma_{\text{ori,land}})$
Maximum height	Sparse	0	0	$w_h(\mathrm{e}(\|h_{\max}-0.9\|^2)/\sigma_{p_z,\max}))$
Jumping	Sparse	0	0	w_{jump}
Base position	Dense	$w_{p_z,\text{st}}(\mathrm{e}(-\|p_z-0.20\|^2/\sigma_{p_z,\text{st}}))$	$w_{p_z,\text{fl}}(\mathrm{e}(-\|p_z-0.7\|^2/\sigma_{p_z,\text{fl}}))$	$w_{\mathbf{p},\text{l}}(\mathrm{e}(-\Sigma\|\mathbf{p}-\mathbf{p}_{\text{des}}\|^2/\sigma_{p,l}))$
Orientation tracking	Dense	$w_{\text{ori,st}}(\mathrm{e}(-\|\log(\bar{\mathbf{q}}_{\text{base}}^{-1}*\bar{\mathbf{q}}_{\text{des}}\|^2/\sigma_{\text{ori,st}}))$	0	$w_{\text{ori,l}}(\mathrm{e}(-\|\log(\bar{\mathbf{q}}_{\text{base}}^{-1}*\bar{\mathbf{q}}_{\text{des}})\|^2/\sigma_{\text{ori,l}}))$
Base linear velocity	Dense	0	$w_{\mathbf{v}_{x,y}}(-\mathrm{e}(\Sigma\|\mathbf{v}_{x,y}-\mathbf{v}_{\text{des}}\|^2/\sigma_v))$	0
Base angular velocity	Dense	0	$w_{\omega}(\mathrm{e}(-\Sigma\|\boldsymbol{\omega}-\boldsymbol{\omega}_{\text{des}}\|^2/\sigma_{\omega}))$	$0.1w_{\omega}(\mathrm{e}(-\Sigma\|\boldsymbol{\omega}\|^2/\sigma_{\omega}))$
Feet clearance	Dense	0	$w_{\text{feet}}(\|p_{\text{feet}}-p_{\text{feet}}^0+[0.0,0.0,-0.15]\|^2)$	0
Symmetry	Dense		$w_{\text{sym}}(\Sigma_{\text{joint}}\|(\mathbf{q}_{\text{left}}-\mathbf{q}_{\text{right}}\|^2)$	
Nominal pose	Dense	$w_{\mathbf{q}}(\mathrm{e}(-\Sigma_{\text{joint}}\|\mathbf{q}_j-\mathbf{q}_{j,\text{nom}}\|^2/\sigma_q)$	$0.1w_{\mathbf{q}}(\mathrm{e}(-\Sigma_{\text{joint}}\|\mathbf{q}_j-\mathbf{q}_{j,\text{nom}}\|^2/\sigma_q)$	$w_{\mathbf{q}}(\mathrm{e}(-\Sigma_{\text{joint}}\|\mathbf{q}_j-\mathbf{q}_{j,\text{nom}}\|^2/\sigma_q)$
Energy	Dense		$w_{\text{energy}}(\boldsymbol{\tau}^T\dot{\mathbf{q}})$	
Base acceleration	Dense		$w_{\text{acc}}\|\dot{\mathbf{v}}\|^2$	
Contact change	Dense		$w_c\Sigma_{\text{feet}}(c_{\text{foot}}(t)-c_{\text{foot}}(t-1))$	
Maintain contact	Dense	$w_{\text{contact}}\Sigma_{\text{feet}}\,c_{\text{foot}}(t)$	0	0
Contact forces	Dense		$w_{F_c}\Sigma_{i=0}^{n_f}\|F_i-\bar{F}\|$	
Action rate	Dense		$w_a\Sigma_{\text{joint}}\|\mathbf{a}(t)-\mathbf{a}(t-1)\|^2$	
Joint acceleration	Dense		$w_{\ddot{q}}\Sigma_{\text{joint}}\|\ddot{\mathbf{q}}_j\|^2$	
Joint limits	Dense		$w_{q_{lim}}\Sigma_{\text{joint}}\|\mathbf{q}_j-\mathbf{q}_{j,lim}\|^2$	

The light orange color indicates task-based rewards, while the light purple shade describes regularization rewards. $w_{\times}$ is the weight, $\sigma_{\times}$ is a scaling factor for the exponential kernel, and $\mathrm{e}(\cdot)$ and $\log(\cdot)$ separately denote the exponent and logarithm operation.

from the nominal joint positions $\mathbf{q}^{\text{nom}} \in \mathbb{R}^{12}$. To smooth the output actions, we used an exponential moving average low-pass filter with a cutoff frequency of 5 Hz. The filtered actions are then scaled and added to $\mathbf{q}^{\text{nom}}$ to generate $\mathbf{q}^{\text{des}}$ for the motor servos, i.e., $\mathbf{q}^{\text{des}} = \mathbf{a} + \mathbf{q}^{\text{nom}}$. A proportional derivative (PD) feedback controller then produces the desired torque at a higher frequency, as shown in Figure 4. To guarantee safety, we clip $\mathbf{q}^{\text{des}}$ within the feasibility range when the real joint angles approach the limits.

REWARDS

Ideally, we expect the agent to accomplish the task while maximizing the rewards it receives. However, a poor choice of reward scaling could lead the agent to converge to the local minima, e.g., standing behavior without jumping, where only certain penalties, like energy cost and joint acceleration, are minimized. To avoid this, instead of naively summing them, we multiply the positive component of the reward by the exponent of the squared negative component, i.e., $r_{\text{total}} = r^{+}\mathrm{e}(-\|r^{-}\|^2/\sigma)$.[4] This allows the agent to always receive a strictly positive reward, scaled down by the amount of penalties, which improves the learning stability.

As listed in Table 1, three phases are used to describe when each reward is given. In particular, *stance* indicates that the robot has been given a command to jump but is still on the ground. Then, flight is triggered when the robot is in midair

[4]For conciseness, the notation $\mathrm{e}(-\|x\|^2/\sigma)$ 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 zero and one.

and has no contact with the ground. Finally, the landing begins upon landing and lasts until the end of the episode. In each phase, task-based rewards (in orange) and regularization rewards (in violet) are considered. On the other hand, the rewards items can be divided into sparse type and dense type, where the former is given once per episode (typically at the end), and the latter is given once per each simulation step that satisfies the conditions.

TASK REWARDS

First, sparse rewards are introduced to encourage the general behavior for accomplishing the desired jumping task, including 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 with the desired orientation. These rewards are only given once at the end of the episode, marked by "sparse" in Table 1. In addition, dense task-related objectives are also defined to simplify the exploration, including

- tracking the desired linear velocity ($\mathbf{v}^{b}_{x,y,\mathrm{des}}$) and yaw angular velocity while in flight and tracking zero angular velocity after landing
- squatting down to a height of 0.2 m while on the ground and tracking a certain height in the air
- maintaining a constant base position and tracking the desired orientation after landing.

Notably, to ensure enough clearance when jumping forward and over obstacles, we introduce a foot clearance reward that tracks the nominal foot position (i.e., at the nominal joint angles q^{nom}) on the xy-plane and, simultaneously, minimizes the z-distance between each foot and the CoM. This objective encourages the robot to tuck its legs in close to its body while in the air.

REGULARIZATION REWARDS

As we do not imprint any reference motions onto the agent, auxiliary regularization rewards are needed to achieve smooth, feasible, and safe behavior. Specifically, we penalize the action rate, together with any violations of predefined soft limits for the joint position. Furthermore, the instantaneous energy power, computed as the dot product between the actuator torque and joint velocity, is penalized for generating an energy-efficient motion. Considering that various quadrupedal jumps seen in nature exhibit high left- and right-side symmetry, we drive the robot toward maintaining this symmetry with an additional reward. Finally, we noticed that the robot often stomped its feet rapidly during the squat-down stage in the training process. To eliminate this unnecessary behavior, we add a small reward for maintaining contact in the first few steps of the episode as well as a penalty on frequent contact state changes.

TERMINATION

We terminate each episode when the following events occur:

- collision between body links and the environment
- base height lower than 0.12 m
- orientation error larger than 3 rad
- landing position error bigger than 0.15 m.

DOMAIN RANDOMIZATION

To bridge the gap between simulation and real-world scenarios, we implement zero-shot domain randomization. The ground friction, restitution, and link mass are sampled at random at the start of every episode. In addition, we add a random offset to the joint encoder values, randomize proportional and derivative gains of the PD controller, and randomize the strength of the motors for every episode. The range of each randomized variable is listed in Table 2.

For hardware control, unmodeled communication delays and latencies strongly weaken the performance of learning-based policies. To tackle this issue, at the beginning of each episode, we sample a latency value from the range of $l \in [0, 40]$ ms. Then, at each step, we add a small random value to reflect the effect of stochastic communication delays.

EXPERIMENTAL VALIDATION

In this section, we first validate the policy trained on the first two curriculum stages (i.e., policy π_2, shown in Figure 2), through various experiments—forward and diagonal jumps, continuous jumps, and robust jumping in the presence of environmental disturbances and uneven terrains. Then, we validate the policy after the final training stage (policy π_3) when jumping onto and over obstacles. For the deployment, we used a naive velocity state estimator that uses the inverse kinematics (IK) to estimate the body velocity while in contact with the ground and integrates the inertial measurement unit acceleration data in midair.

TRAINING SETUP

The implementation is based on the open source Legged gym environment [9]. Specifically, we use 4,096 agents and 24 environmental steps per agent per update step. For the vertical jump, we train for 3,000 iterations, while, for the

TABLE 2. Randomized variables and their ranges.

NAME	RANDOMIZATION RANGE
Ground friction	[0.01, 3]
Ground restitution	[0, 0.4]
Additional payload	[–1, 3] kg
Link mass factor	[0.7, 1.3] ×
CoM displacement	[–0.1, 0.1] m
Episodic latency	[0, 40] ms
Extra per-step latency	[–5, 5] ms
Motor strength factor	[0.9, 1.1] ×
Joint offsets	[–0.02, 0.02] rad
PD gains factor	[0.9, 1.1] ×
Joint friction	[0.0, 0.04]
Joint damping	[0, 0.01] $\mathrm{N \cdot m \cdot s \cdot rad^{-1}}$

forward jump without and with obstacles, we train for 10,000 steps each. The actor policy and the critic are parameterized by a shared MLP with three hidden layers of dimensions [256, 128, 64], with exponential linear unit activations after each layer. Using a single RTX 3090 GPU, the three highly parallelized training stages took approximately 1.4, 4.1, and 4.8 h, respectively.

The policy operates at a frequency of 50 Hz, and the simulation runs at 200 Hz. We performed all of the experiments on the Unitree Go1. During the simulation validations, we use a constant joint friction value of 0.03, joint damping of 0.01 $\mathrm{N \cdot m \cdot s \cdot rad^{-1}}$ and a constant latency of 40 ms.

VERSATILE JUMPING ON FLAT GROUND

FORWARD JUMPING

First, we evaluate the policy on a variety of forward jumps. Figure 6 compares hardware and simulation motions of a 60-cm forward jump, while Figure 7 presents the quantitative results. As can be seen, the real-world behavior closely matches the simulated prediction. One noticeable deviation is in the peak torques at takeoff—where the measured torques deviate from both the desired torques (computed by the PD control law using the desired joint angles) and the simulation torques. Furthermore, larger joint angles for the hip and thigh are measured upon landing in real-world tests, likely due to poor impact modeling in the simulation. Finally, the Euler angles show a slight variation between the simulation and hardware. We believe that this mismatch is mainly due to the motor modeling inaccuracies, coupled with the weight of the additional mass on top of the robot, shifting its CoM. Despite these state deviations, the jumping distance is well tracked, and the base velocity matches the expected behavior, showing a good sim-2real adaptation. The controller was also robust to the particularly noisy contact observations and velocity state estimates, which we attribute to both the extensive domain randomization and the large amounts of noise that we simulate. As our curriculum gradually guides the robot toward accomplishing the task, it is easier for it to converge even in the presence of greater domain randomization and observation noise.

We then tested the maximum distance it could jump across. Figure 8(a) illustrates a 90-cm forward jump, with the target landing point shown by the yellow marker. Despite slipping on the soft pads as it lands, the robot recovers quickly, demonstrating its robustness against uncertainties.[5] To the best of our knowledge, this is the largest jumping distance achieved by robots of similar size and similar actuators (see Table 3).

DIAGONAL JUMPING

Figure 8(b) shows a diagonal jump of 50×30 cm with a desired yaw of 30°. Both the landing position and yaw are tracked accurately.

Furthermore, we evaluated the policy across the whole jumping range in simulation, the success rate and tracking metrics of which are presented in Figure 9. As can be seen from Figure 9(a), the tracking error is lowest for narrow jumps of a forward distance up to 50 cm. As both the longitudinal and lateral distances increase, so does the final landing error. Interestingly, most failed environments asymmetrically occur in the lower right corner of the plot. Figure 9(b) shows the same data but grouped by total desired distance versus actual achieved distance. We found that the data closely follow the 45° line (i.e., ideal performance) for the smaller jumps, with the gradient slowly decreasing after 50 cm.

JUMPING ONTO/ACROSS ROUGH TERRAIN

Here, we evaluate how well the policy performs in the presence of environmental disturbances, despite not being trained on uneven or rough ground. In this section, we ran several experiments, including jumping with obstacles surrounding the robot, blindly jumping from and onto a box, and jumping from asphalt onto a soft grassy terrain. As shown by the time

[5]It is worth mentioning that we reward the position of the base upon landing, rather than the feet. As a result, in the trial, the base cleared the 90-cm distance, but the rear left foot landed a bit behind.

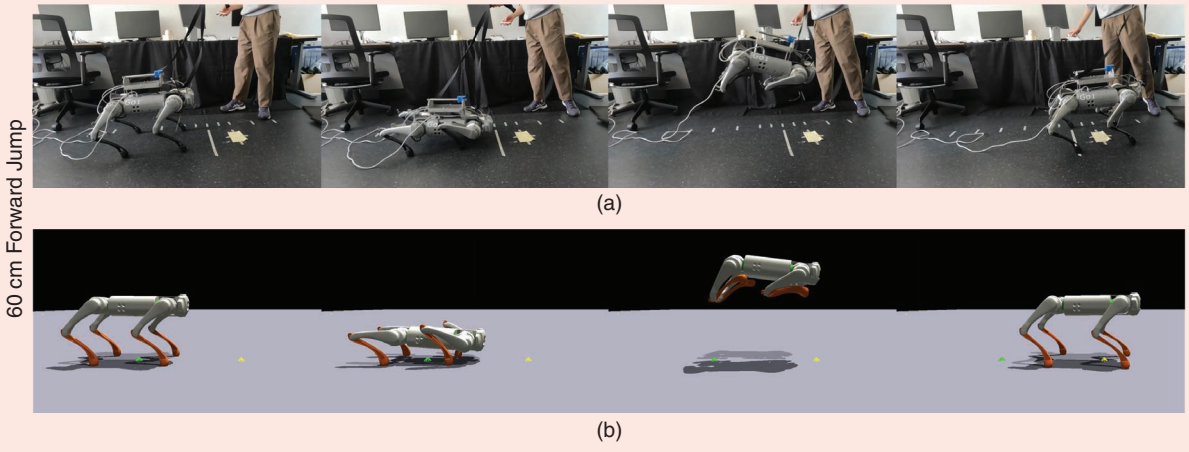


FIGURE 6. The (a) real-world and (b) simulation execution of a forward jump. The yellow marker indicates the desired 60-cm jumping distance.

lapses in Figure 10(a) and (b), the policy enables robust jumping onto both soft and stiff objects that could (and did) slip under the feet of the robot. The third row demonstrates that the robot could jump from hard asphalt onto soft grass, despite training on flat ground only.

Next, we tested the policy on a continuous jumping task, where a new command of a 40-cm forward jump is given following each jump without resetting the robot states. As seen in the fourth row of Figure 10, the policy is robust enough to execute a jump from a variety of different initial states.

FIGURE 7. Hardware and simulation quantitative results for the 60-cm forward jump. We show the desired values in green—which, for torques, are the ideal PD law torques and, for the base Euler angles, are zero. In blue, we show the value measurements and estimates from the hardware experiment, and, in red, we show the values from the simulator for the same task. (a) Joint angles, velocities, and torques for the front right leg during the 60-cm forward jump. The flight phase for the hardware experiment is indicated by the yellow-shaded region. (b) Base angular and linear velocity during the 60-cm forward jump. The flight phase of the hardware test is indicated in light yellow. Sim: simulated.

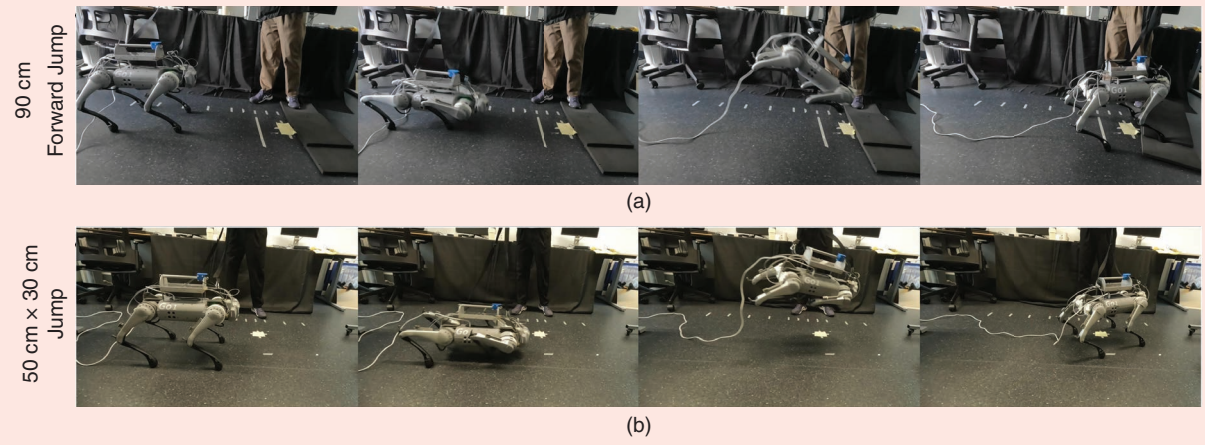


FIGURE 8. Hardware results for (a) a 90-cm forward jump and (b) a 50 × 30-cm diagonal jump with desired yaw of 30°.

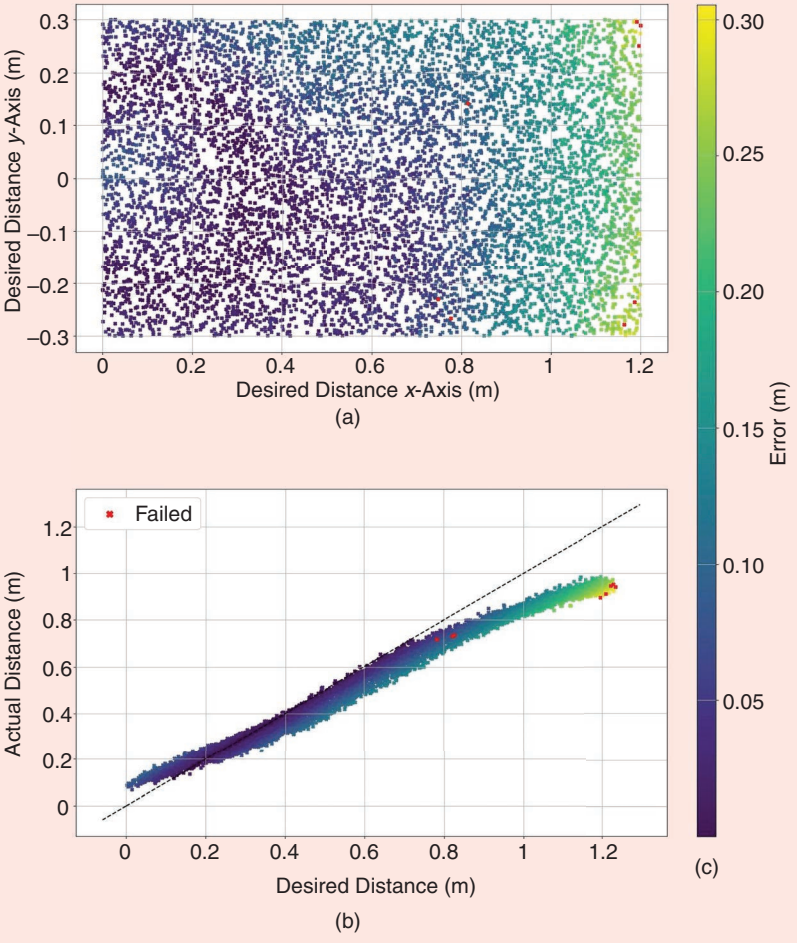


FIGURE 9. The simulation tracking performance as a function of the desired x- and y-axis jumping distances: (a) the tracking performance in terms of the (b) overall desired versus actual jumping distance, with (c) the error (in meters) shown by the color gradient. The environments that have been terminated (due to any nonfoot–ground collisions) are shown in red, and the black 45° dashed line indicates the ideal tracking performance. Data are gathered from 8,000 simulated trials across the whole jumping range $x \in [0, 1.2]$, $y \in [-0.3, 0.3]$. Only eight environments have been terminated (of which just one was terminated due to body–ground contact), leading to a success rate of 99.9%.

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.

FORWARD JUMPING WITH OBSTACLES

To further demonstrate the versatility, we tested forward jumping with obstacles, using policy π_3. To be brief, only two scenarios are presented here, including jumping over a 5-cm-tall thin obstacle and landing on a 10-cm box. In the first task, the robot had to jump across 80 cm to avoid collision. As seen in Figure 11(a), the robot succeeded in jumping over the barrier and landed successfully. In the second case, the robot had to jump across more than 70 cm while maintaining a large height. As a result, better performance was observed, considering that a shorter forward distance enabled the robot to achieve a larger height throughout the flight.

COMPARISON WITH THE STATE OF THE ART

To better understand the strengths of our proposed method, we performed a detailed comparison of the kinodynamic profiles for a 60-cm forward jump with the runner-up approach in Table 3—Extreme Parkour [28]. As can be seen in Figure 12, our method has a much more pronounced squat-down phase compared to the baseline. We would like to emphasize that this is an emerging behavior and is learned

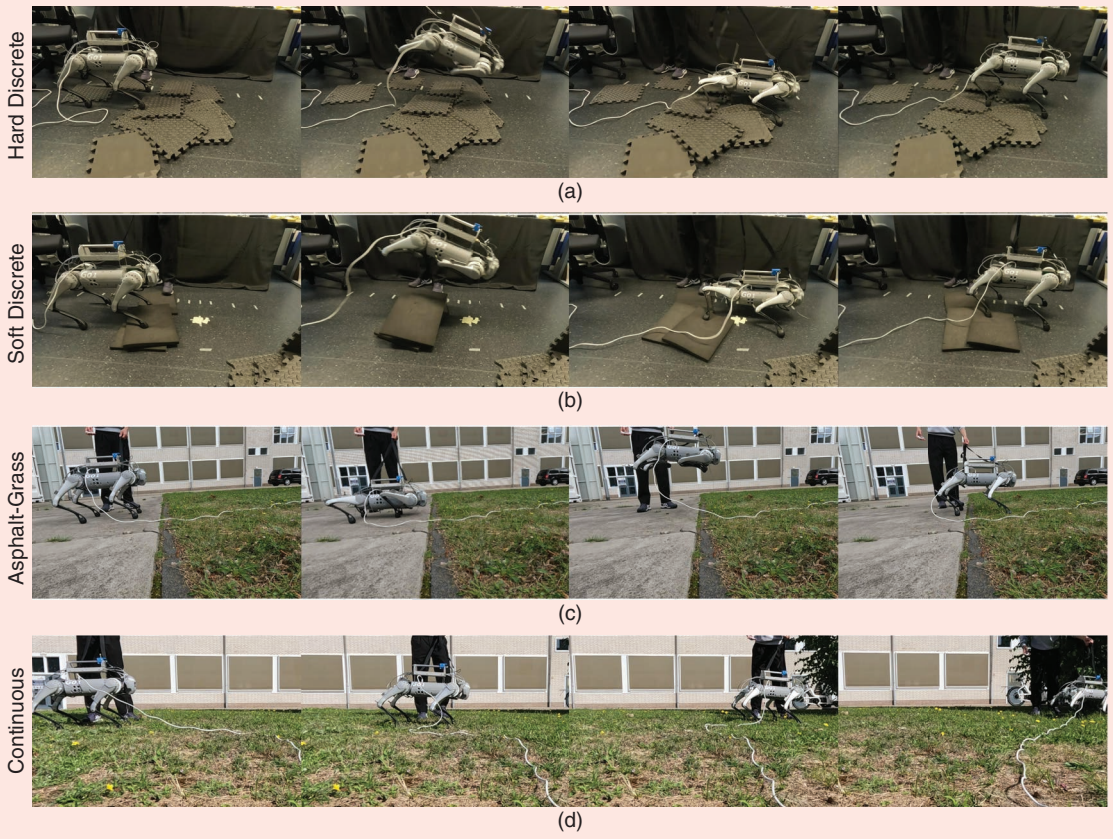


FIGURE 10. Several experiments showcasing the robustness of the policy π_2 to variations in the terrain: jumping across (a) discrete hard and (b) discrete soft objects, (c) an asphalt-to-grass jump, and (d) nine consecutive jumps on grass.

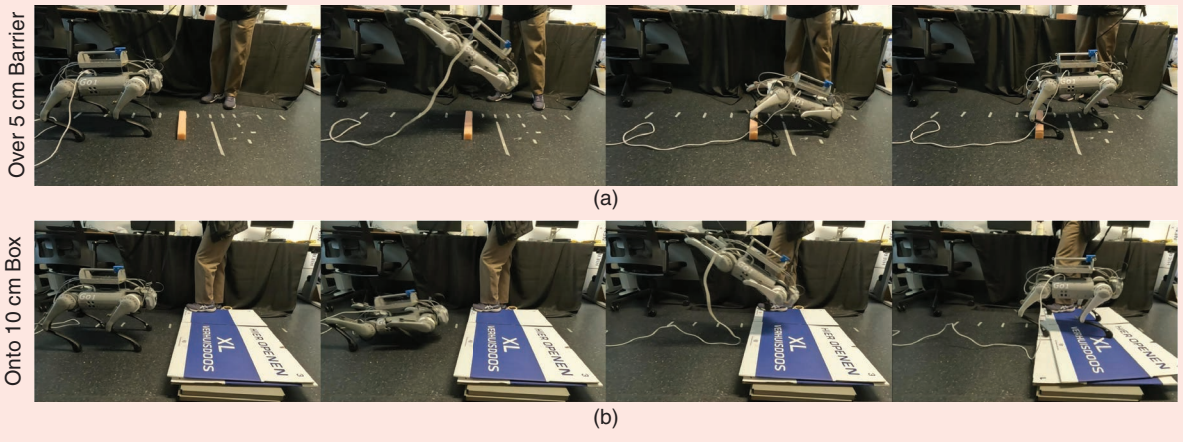


FIGURE 11. (a) Jumping over a 5-cm-tall, 5-cm-wide obstacle. (b) Jumping onto a 10-cm-tall box.

even without the squat-down reward. This allows the robot to accelerate upward and jump much more explosively and with larger momentum, as shown by the linear velocity and torque peaks, compared to the baseline. We attribute our larger peak heights and maximal jump distance to this type of learned motion. Despite the fact that [28] has a large initial x-velocity [see the first plot in Figure 12(b)], it still achieves a smaller maximum jumping distance compared to ours. While [28] converges to a locally optimal and simpler behavior, our curriculum and reward structure allow the robot to better explore

TABLE 3. Maximal jump length comparison with the state of the art.

METHOD	[19]	[21]	[30]	[22]	[23]	[28]	OURS
Jump length (m)	0.2	0.26	0.5	0.6	0.7	0.8	**0.9**
Robot	A1	Mini Cheetah	Custom	Go1	Go1	A1	Go1

Bold indicated the largest achieved distance (best result).

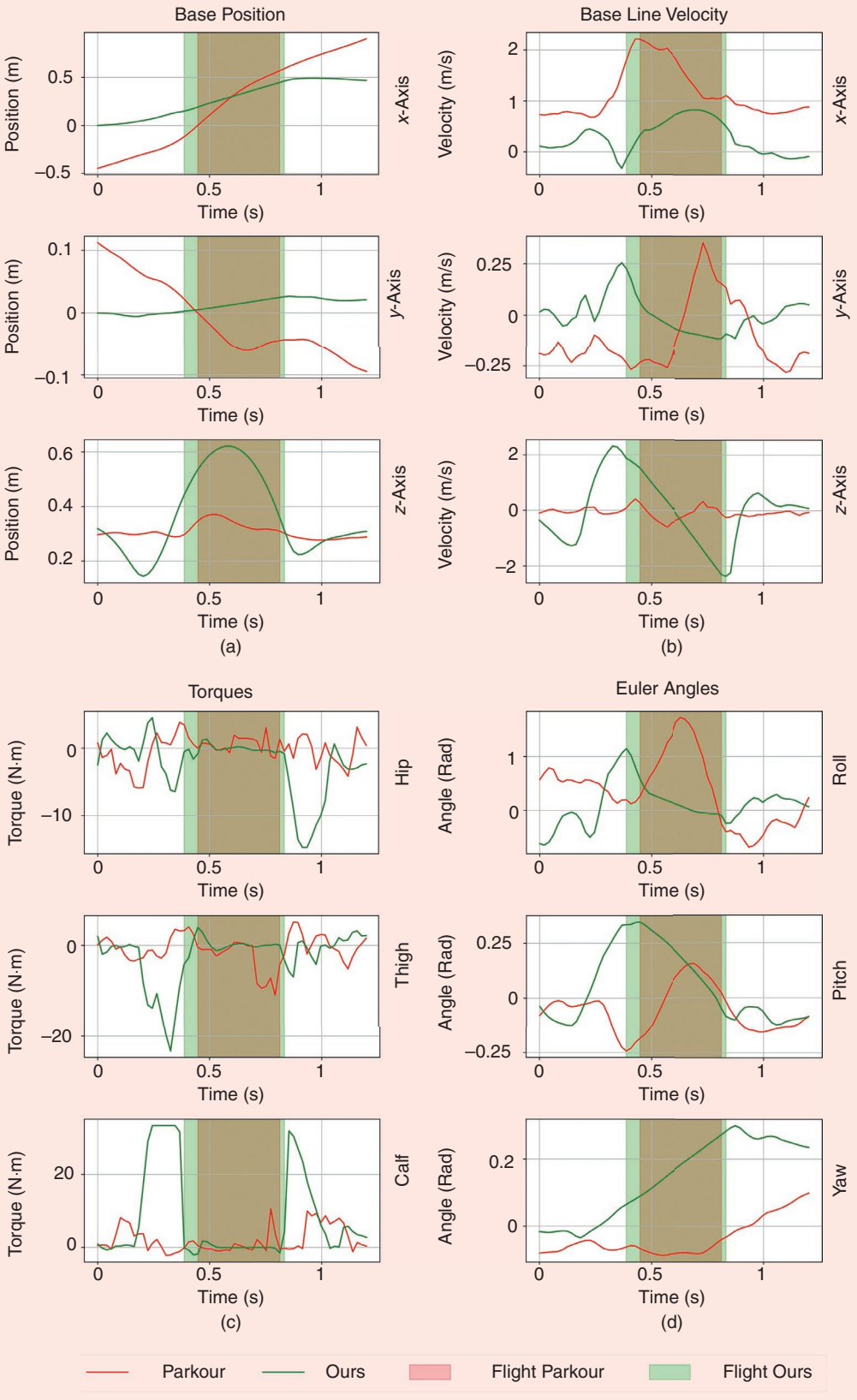


FIGURE 12. A comparison between Extreme Parkour [28] (red) and our work (green). We analyze the (a) base position, (b) base linear velocity, (c) joint torques, and (d) base orientation (Euler angles) for a 60-cm jump for both methods.

and learn how to jump, causing it to discover that squatting down allows it to better accelerate its body and take off with a larger momentum.

ABLATION STUDY

To better understand the effect of our curriculum, we compared our approach to several baselines in Figure 13:

- *no RSI*: training stage 1 without RSI, i.e., no height and upward velocity initialization
- *no curriculum*: directly training stage 2 without pretraining stage 1, but with RSI height and velocity initialization. For fairness, we train this baseline for an additional 3,000 steps
- *no curriculum and no domain randomization*: same as "no curriculum" but without any applied domain randomization
- *no curriculum and no RSI*: same as "no curriculum" but without any RSI.

As can be seen from Figure 13(a), the RSI is required for learning the jumping-in-place task. Without it, the agent converges to a local optimum and fails to complete the task. Despite the overall high reward, it can be seen in Figure 13(b) that directly training the long-distance jump also results in an early convergence to a standing behavior, which highlights the need for our curriculum strategy. Similarly, we also observe this behavior even without applying domain randomization, which can have a regularizing effect.

DISCUSSION AND CONCLUSION

In this work, we present a curriculum-based end-to-end DRL approach capable of 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, unlike other reference-free works, our curriculum guides the robot to more effectively explore the task of jumping and develop a much more agile and explosive style of jumping, significantly outperforming existing methods. Our framework is flexible, and it can be adopted to learn better jumping skills in existing methods, which learn a separate set of skills for different each task [29], [30], [31].

Furthermore, through domain randomization, we successfully deployed the policy onto the real system and closely matched the expected behavior 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 90-cm 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 despite only having been trained on flat terrain in simulation. 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 10 cm.

> "OUR POLICY ACHIEVED A 90-CM FORWARD JUMP ON THE UNITREE GO1 ROBOT, A DISTANCE GREATER THAN THOSE REPORTED BY OTHER MODEL- AND LEARNING-BASED CONTROLLERS."

When executing a long-distance jump, real animals exhibit a four-legged contact phase, followed by an upward pitch and pure rear-leg contact at takeoff. During landing, a mirrored behavior is observed—the body is pitching downward, and contact is first gained with the front legs. Previous model-based control works [4], [5] have manually incorporated this contact schedule into their optimizers. It would be interesting to investigate how such behavior can be learned through DRL without supplying a reference trajectory and validate its benefits compared to the style of jumping exhibited here.

The number of rewards can be seen as a limitation of our approach, although they are intuitive and relatively easy to adjust. During our evaluation, we found that most of these

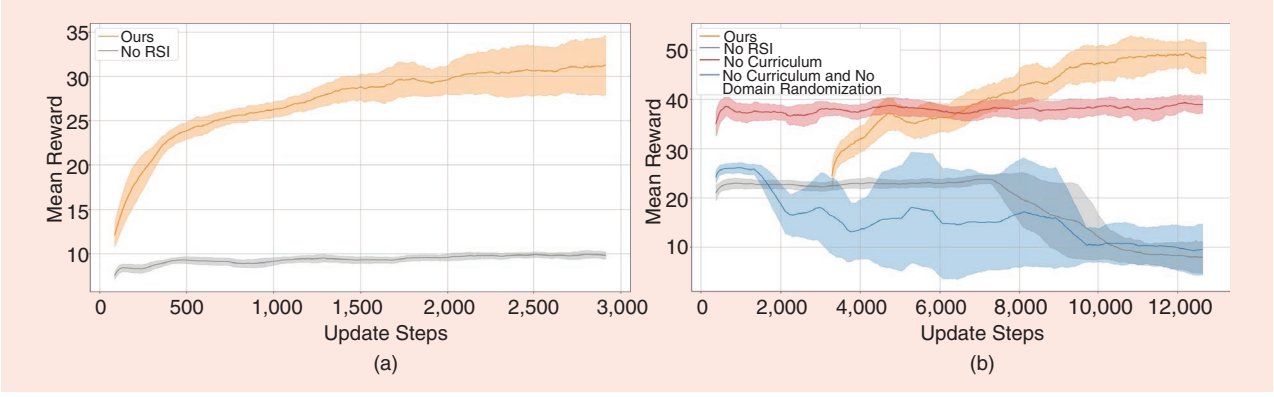


FIGURE 13. The mean reward throughout training for the (a) stage 1: jumping in place and (b) stage 2: long-distance jump tasks.

rewards primarily serve to regularize the motion and are not essential for learning the jumping behavior. It would be interesting to investigate how improved exploration methods could reduce the need for many of the dense rewards. Additionally, in the future, we aim to enhance the robot's mobility by generalizing to a broader range of agile behaviors, including climbing skills, various jumping styles, and acrobatic motions.

ACKNOWLEDGMENT

This work is supported in part by the EU project 101016970 NI and in part by the European Union's Horizon Europe Program from Project EMERGE under Grant 101070918. A supplementary video can be found at https://www.youtube.com/watch?v=nRaMCrwU5X8. The code associated with this work can be found at https://github.com/vassil-atn/Curriculum-Quadruped-Jumping-DRL. The corresponding author is Jiatao Ding.

AUTHORS

Vassil Atanassov, Oxford Robotics Institute, Department of Engineering Science, University of Oxford, OX1 3PJ Oxford, U.K. E-mail: vassilatanassov@robots.ox.ac.uk.

Jiatao Ding, Department of Cognitive Robotics, Delft University of Technology, 2628CD Delft, The Netherlands. E-mail: j.ding-2@tudelft.nl.

Jens Kober, Department of Cognitive Robotics, Delft University of Technology, 2628CD Delft, The Netherlands. E-mail: j.kober@tudelft.nl.

Ioannis Havoutis, Oxford Robotics Institute, Department of Engineering Science, University of Oxford, OX1 3PJ Oxford, U.K. E-mail: ioannis@robots.ox.ac.uk.

Cosimo Della Santina, Department of Cognitive Robotics, Delft University of Technology, 2628CD Delft, The Netherlands, and the Institute of Robotics and Mechatronics, German Aerospace Center, 82234 Wessling, Germany. E-mail: cosimodellasantina@gmail.com.

REFERENCES

[1] J. Hwangbo et al., "Learning agile and dynamic motor skills for legged robots," *Sci. Robot.*, vol. 4, no. 26, Jan. 2019, Art. no. eaau5872, doi: 10.1126/scirobotics.aau5872.

[2] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Sci. Robot.*, vol. 7, no. 62, Jan. 2022, Art. no. eabk2822, doi: 10.1126/scirobotics.abk2822.

[3] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, "Legged locomotion in challenging terrains using egocentric vision," 2022, *arXiv:2211.07638*.

[4] Q. Nguyen, M. J. Powell, B. Katz, J. D. Carlo, and S. Kim, "Optimized jumping on the MIT cheetah 3 robot," in *Proc. Int. Conf. Robot. Automat.*, May 2019, pp. 7448–7454, doi: 10.1109/ICRA.2019.8794449.

[5] C. Nguyen and Q. Nguyen, "Contact-timing and trajectory optimization for 3D jumping on quadruped robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 11,994–11,999, doi: 10.1109/IROS47612.2022.9981284.

[6] J. Ding, V. Atanassov, E. Panichi, J. Kober, and C. D. Santina, "Robust quadrupedal jumping with impact-aware landing: Exploiting parallel elasticity," *IEEE Trans. Robot.*, vol. 40, pp. 3212–3231, 2024, doi: 10.1109/TRO.2024.3411988.

[7] J. Ding, M. A. v. L. Sels, F. Angelini, J. Kober, and C. D. Santina, "Robust jumping with an articulated soft quadruped via trajectory optimization and iterative learning," *IEEE Robot. Autom. Lett.*, vol. 9, no. 1, pp. 255–262, Jan. 2023, doi: 10.1109/LRA.2023.3331288.

[8] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid motor adaptation for legged robots," in *Robotics: Science and Systems XVII*, D. A. Shell, M. Toussaint, and M. A. Hsieh, Eds., Robotics: Science and Systems Foundation, Jul. 2021, doi: 10.15607/RSS.2021.XVII.011.

[9] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Proc. Conf. Robot Learn.*, Jan. 2022, pp. 91–100.

[10] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Sci. Robot.*, vol. 5, no. 47, Oct. 2020, Art. no. eabc5986, doi: 10.1126/scirobotics.abc5986.

[11] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning agile skills via adversarial imitation of rough partial demonstrations," in *Proc. Conf. Robot Learn.*, 2022, vol. 205, pp. 342–352.

[12] Z. Li, X. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Robust and versatile bipedal jumping control through reinforcement learning," in *Robotics: Science and System XIX*, K. Bekris, K. Hauser, S. Herbert, and J. Yu, Eds., Robotics: Science and Systems Foundation, Jul. 2023, doi: 10.15607/RSS.2023.XIX.052.

[13] G. Bellegarda, C. Nguyen, and Q. Nguyen, "Robust quadruped jumping via deep reinforcement learning," 2023, *arXiv:2011.07089*.

[14] Y. Fuchioka, Z. Xie, and M. van de Panne, "OPT-Mimic: Imitation of optimized trajectories for dynamic quadruped behaviors," in *Proc. Int. Conf. Robot. Automat.*, 2023, pp. 5092–5098, doi: 10.1109/ICRA48891.2023.10160562.

[15] F. Yu, R. Batke, J. Dao, J. Hurst, K. Green, and A. Fern, "Dynamic bipedal turning through sim-to-real reinforcement learning," in *Proc. IEEE-RAS 21st Int. Conf. Humanoid Robots (Humanoids)*, Piscataway, NJ, USA: IEEE Press, Nov. 2022, pp. 903–910, doi: 10.1109/Humanoids53995.2022.10000225.

[16] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa, "AMP: Adversarial motion priors for stylized physics-based character control," *ACM Trans. Graphics (ToG)*, vol. 40, no. 4, pp. 1–20, 2021, doi: 10.1145/3476576.3476723.

[17] A. Escontrela et al., "Adversarial motion priors make good substitutes for complex reward functions," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Piscataway, NJ, USA: IEEE Press, Oct. 2022, pp. 25–32, doi: 10.1109/IROS47612.2022.9981973.

[18] E. Vollenweider, M. Bjelonic, V. Klemm, N. Rudin, J. Lee, and M. Hutter, "Advanced skills through multiple adversarial motion priors in reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, Piscataway, NJ, USA: IEEE Press, May 2023, pp. 5120–5126, doi: 10.1109/ICRA48891.2023.10160751.

[19] L. M. Smith et al., "Learning and adapting agile locomotion skills by transferring experience," 2023, *arXiv:2304.09834*.

[20] Z. Yin, Z. Yang, M. van de Panne, and K. Yin, "Discovering diverse athletic jumping strategies," *ACM Trans. Graphics*, vol. 40, no. 4, pp. 91:1–91:17, 2021, doi: 10.1145/3450626.3459817.

[21] G. B. Margolis et al., "Learning to jump from pixels," in *Proc. Conf. Robot Learn.*, PMLR, 2021, vol. 164, pp. 1025–1034.

[22] Y. Yang, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Continuous versatile jumping using learned action residuals," in *Proc. Annu. Learn. Dyn. Control Conf.*, Jun. 2023, pp. 770–782.

[23] Y. Yang et al., "CAJun: Continuous adaptive jumping using a learned centroidal controller," in *Proc. 7th Conf. on Robot Learn.*, PMLR, Dec. 2023, pp. 2791–2806.

[24] 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 Trans. Robot.*, vol. 38, no. 1, pp. 317–328, Feb. 2022, doi: 10.1109/TRO.2021.3084374.

[25] F. Vezzi, J. Ding, A. Raffin, J. Kober, and C. D. Santina, "Two-stage learning of highly dynamic motions with rigid and articulated soft quadrupeds," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, Piscataway, NJ, USA: IEEE Press, May 2024, pp. 9720–9726, doi: 10.1109/ICRA57147.2024.10610561.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[27] Z. Xie, H. Y. Ling, N. H. Kim, and M. van de Panne, "ALLSTEPS: Curriculum-driven learning of stepping stone skills," *Comput. Graphics Forum*, vol. 39, no. 8, pp. 213–224, 2020, doi: 10.1111/cgf.14115.

[28] X. Cheng, K. Shi, A. Agarwal, and D. Pathak, "Extreme parkour with legged robots," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, Piscataway, NJ, USA: IEEE Press, May 2024, pp. 11,443–11,450, doi: 10.1109/ICRA57147.2024.10610200.

[29] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, "Anymal parkour: Learning agile navigation for quadrupedal robots," *Sci. Robot.*, vol. 9, no. 88, 2024, Art. no. eadi7566, doi: 10.1126/scirobotics.adi7566.

[30] K. Caluwaerts et al., "Barkour: Benchmarking animal-level agility with quadruped robots," May 2023, *arXiv:2305.14654*.

[31] Z. Zhuang et al., "Robot parkour learning," in *Proc. 7th Conf. Robot Learn.*, J. Tan, M. Toussaint, and K. Darvish, Eds., Nov. 2023, PMLR, vol. 229, pp. 73–92.

[32] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graphics*, vol. 37, no. 4, pp. 143:1–143:14, Jul. 2018, doi: 10.1145/3197517.3201311.