Robust quadruped jumping via deep reinforcement learning

Guillaume Bellegarda [a],[1],[*], Chuong Nguyen [b],[1], Quan Nguyen [b]

[a] *Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, 1015, VD, Switzerland*
[b] *University of Southern California, Los Angeles, 90007, CA, USA*

ARTICLE INFO

ABSTRACT

In this paper, we consider a general task of jumping varying distances and heights for a quadrupedal robot in noisy environments, such as off of uneven terrain and with variable robot dynamics parameters. To accurately jump in such conditions, we propose a framework using deep reinforcement learning that leverages and augments the complex solution of nonlinear trajectory optimization for quadrupedal jumping. While the standalone optimization limits jumping to take-off from flat ground and requires accurate assumptions of robot dynamics, our proposed approach improves the robustness to allow jumping off of significantly uneven terrain with variable robot dynamical parameters and environmental conditions. Compared with walking and running, the realization of aggressive jumping on hardware necessitates accounting for the motors' torque-speed relationship as well as the robot's total power limits. By incorporating these constraints into our learning framework, we successfully deploy our policy sim-to-real without further tuning, fully exploiting the available onboard power supply and motors. We demonstrate robustness to environment noise of foot disturbances of up to 6 cm in height, or 33% of the robot's nominal standing height, while jumping $2x$ the body length in distance.

1. Introduction

Legged robots have potential to accomplish many tasks that may be unsafe for humans, in which overcoming uneven terrain or high obstacles may be necessary. Towards real world deployment, recent works have shown highly dynamic and agile motions such as biped [1] and quadruped [2] backflips, wheel-legged biped jumping [3,4], quadruped running and obstacle jumping [5], and continuous jumping on uneven terrain [6] or on stepping stones [7]. Such methods have used either a simple model for real-time planning, or there is no associated publication.

With respect to optimized jumping, our prior work optimizes over a full quadruped model to perform highly dynamic jumps [8,9]. A tethered quadruped model shows potential for energy efficient lunar jumping with flight phase pitch control through a reaction wheel [10]. Other works have shown single legged [11] and/or dynamic miniature [12, 13] jumping, for which more recent work shows SALTO performing prolonged jumping in non-laboratory settings [14].

To make jumping more robust to external disturbances and new unseen environments, deep learning offers an attractive and generalizable formulation. Deep reinforcement learning in particular has recently shown impressive results in learning control policies for quadrupeds [15–18]. Typically such methods train from scratch (i.e. use little or no prior information about the system) and rely on extensive simulation with randomized environment parameters. To facilitate the sim-to-real transfer, additional techniques are employed such as online parameter adaptation [18,19], learned state estimation modules [20], teacher–student training [16,19,21], and careful Markov Decision Process choices [22–24]. With respect to quadruped jumping, some recent works have designed complex multi-stage training processes with curriculums and carefully crafted reward functions to learn to jump over a gap, either in simulation [25] or on hardware [26].

In contrast, in this paper we seek to use deep reinforcement learning to improve upon jumping motions produced with a trajectory optimization framework. Under ideal conditions (i.e. starting on flat ground with a high enough coefficient of friction), the motions produced from the optimization can be accurately tracked on hardware, as shown in our prior work [8,9]. However, under disturbances in foot heights (i.e. <0.05 m), a feedforward controller using only the reference trajectory will lead to taking off at an incorrect pitch angle, causing significant deviation from the desired motion. In addition to the challenges associated with highly dynamic motions, such potential errors come with high risk and can have very costly consequences such as robot damage.

---

* Corresponding author.
E-mail addresses: guillaume.bellegarda@epfl.ch (G. Bellegarda), vanchuong.nguyen@usc.edu (C. Nguyen), quann@usc.edu (Q. Nguyen).
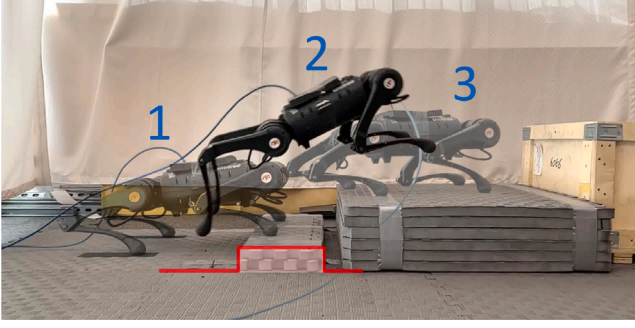[1] These authors contributed equally to this work.

Fig. 1. Unitree A1, a mini quadruped robot with leg length of 0.2 m, successfully jumping off of an unknown 0.06 m block (red line), with a goal distance of 0.6 m and height 0.2 m. Experiment video link: https://youtu.be/fVxXTofT1f4. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Many of the legged robots used to deploy learned locomotion policies in sim-to-real make use of direct current (DC) motors due to their ability to deliver high mobility with on-board batteries, because they offer a wide range of speeds and high torque. For learning locomotion, common assumptions include that (i) motor torque and velocity are independent in the operating region, and (ii) the on-board battery always has sufficient power to execute the learned policy (e.g. [17,18,23,24]). These assumptions are reasonable for locomotion tasks such as walking or running, which require high joint velocities rather than torques, and do not need to use the full available power of the battery. In contrast to walking or running, highly aggressive motions such as jumping require both high joint torque and high joint velocity. However, DC motors typically do not allow both torque and velocity reaching their maximum values at the same time due to their inherent relationship in motor dynamics constraints. Moreover, in order to accomplish jumping motions, robots typically require total power that rapidly reaches the limits of the on-board power supply. Therefore, it is critical to consider these constraints when trying to deploy learned control policies to successfully jump in sim-to-real. This motivates us to integrate motor dynamics and power constraints into our learning framework. Our integration considers both the torque–velocity relationship and the on-board power supply to represent the true system limits, which enables effective sim-to-real transfer for highly agile jumping motions.

Contribution: We present a method for improving the performance of the feedforward controller used on optimal jumping trajectories. We learn a general controller to track multiple desired jumping trajectories with deep reinforcement learning, to successfully jump in noisy environments with uneven ground, as shown in Fig. 1. This controller is trained on, and able to track, many different jumping trajectories, and works with different joint gains. By learning a single controller capable of achieving many different jumps, this also avoids re-running potentially computationally expensive optimization routines at run time for relatively small differences in initial state. Moreover, in contrast to our prior work on MIT Cheetah 3 [8] and Unitree A1 [9], our DRL controller is run as a real-time feedback controller, making our novel approach both more reliable and more robust. Importantly, we also incorporate motor dynamic constraints and power limits into the learning framework, allowing effective sim-to-real deployment without further tuning for robust and highly dynamic jumping motions. Our hardware results demonstrate that our method fully exploits the available onboard power supply and takes the motors to their limits to achieve robust and dynamic jumping motions under noisy conditions.

The rest of this paper is organized as follows. Section 2 provides background details on the robot model, reinforcement learning, and gives a brief overview of the jumping trajectory optimization. Section 3 describes our learning framework design choices, including the integration of motor dynamics and power constraints to achieve

Table 1
Motor and on-board battery parameters.

Parameter	Value	Units
Motor gear ratio	9	–
Max joint torque	33.5	N m
Max joint speed	21	Rad/s
Max battery voltage	21.5	V
Max battery current	60	A
Max battery power	1290	W

robust jumping. Section 4 shows extensive numerical simulations and experimental results from learning our general jumping controller, and a brief conclusion is given in Section 5.

2. Background

2.1. Robot model

In this paper, we validate our jumping controller on the Unitree A1 [27] quadruped robot. The A1 robot has low-inertial legs and high torque density DC motors with planetary gear reduction, and it is capable of ground force control without using any force or torque sensors. A1 uses these high-performance actuators for each hip, thigh, and knee joint to enable full 3D control of ground reaction forces. It is also equipped with contact sensors on each foot.

The A1 legs feature a large range of motion: the hip joints have a range of motion of $\pm 46°$, the thigh joints have a range of motion from $-60°$ to $240°$, and the knee joints have a range of motion from $-154.5°$ to $-52.5°$. Each of A1's actuators consist of a custom high torque density electric motor coupled to a single-stage 9:1 planetary gear reduction. The lower link is driven by a bar linkage which passes through the upper link. The legs are serially actuated, but to keep leg inertia low, the hip and knee actuators are co-axially located at the hip of each leg. The actuation capabilities of the A1 robot and battery power supply limits are summarized in Table 1.

2.2. Reinforcement learning

In the reinforcement learning framework [28], an agent interacts with an environment modeled as a Markov Decision Process (MDP). An MDP is given by a 4-tuple (S, A, P, R), where S is the set of states, A is the set of actions available to the agent, $P : S \times A \times S \rightarrow \mathbb{R}$ is the transition function, where $P(s_{t+1}|s_t, a_t)$ gives the probability of being in state s_t, taking action a_t, and ending up in state s_{t+1}, and $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, where $R(s_t, a_t, s_{t+1})$ gives the expected reward for being in state s_t, taking action a_t, and ending up in state s_{t+1}. The goal of an agent is to interact with the environment by selecting actions that will maximize future rewards. In this paper, we use Soft-Actor Critic (SAC) [29] to learn the optimal policy π to maximize jumping performance.

SAC learns a policy, $\pi(a|s)$, and a critic, $Q_\phi(s, a)$, and aims to maximize a weighted objective of the reward and the policy entropy, $\mathbb{E}_{s_t, a_t \sim \pi} \left[\sum_t r_t + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$. The critic parameters are learned by minimizing the squared Bellman error using transitions, $\tau_t = (s_t, a_t, s_{t+1}, r_t)$, replayed from an experience buffer, D:

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\tau \sim D} \left[\left(Q_\phi(s_t, a_t) - (r_t + \gamma V(s_{t+1})) \right)^2 \right] \quad (1)$$

The target value of the next state can be estimated by sampling an action using the current policy:

$$V(s_{t+1}) = \mathbb{E}_{a' \sim \pi} \left[Q_{\bar{\phi}}(s_{t+1}, a') - \alpha \log \pi(a'|s_{t+1}) \right] \quad (2)$$

where $Q_{\bar{\phi}}$ represents a more slowly updated copy of the critic. The policy is learned by minimizing the divergence from the exponential of the soft-Q function at the same states:

$$\mathcal{L}_\pi(\psi) = -\mathbb{E}_{a \sim \pi} \left[Q_\phi(s_t, a) - \alpha \log \pi(a|s_t) \right] \quad (3)$$

This is done via the reparameterization trick for the newly sampled action, and α is learned against a target entropy.
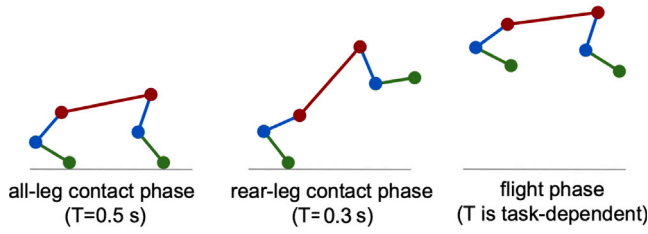
where $K_{p,joint}$ and $K_{d,joint}$ are diagonal matrices of proportional and derivative gains in the joint coordinates.

To improve tracking performance, a Cartesian PD controller is added. From the desired joint angle (q_d) and joint velocity ($\dot{q}_d$) trajectories, we extract desired foot positions (p_d) and foot velocities (v_d) in the leg frame. Thus the Cartesian PD and full controllers for tracking the desired jumping trajectory become:

$$\tau_{\text{Cartesian}} = J(q)^{\top} \left[K_p \left(p_d - p \right) + K_d \left(v_d - v \right) \right] \tag{5}$$

$$\tau_{\text{opt}} = \tau_{\text{Cartesian}} + \tau_{\text{ff}} \tag{6}$$

where $J(q)$ is the foot Jacobian at joint configuration q, K_p and K_d are diagonal matrices of proportional and derivative gains in Cartesian coordinates, and τ_{ff} is the feed-forward torque from Eq. (4).

3. Robust jumping with reinforcement learning

Given the already established Cartesian and joint space controller for tracking jumping motions, in this section we describe our process and reinforcement learning framework for learning to modify and track these optimal trajectories in the presence of environmental noise and disturbances. Our learning framework integrates motor dynamics and power constraints, taking into account the motor torque-speed relationship and maximum on-board power supply, to represent the actual system limits. This integration enables effective sim-to-real transfer for robust and agile jumping on the robot hardware.

3.1. Learning optimal trajectory offsets

In order to provide an intuitive mapping between actions and their effects on the system, we propose learning to appropriately offset the jumping trajectories to cope with disturbances in the environment. Specifically, we consider learning in Cartesian space, with the idea that the agent can more directly observe the effects of its actions, as well as more easily map offsets based on the environmental observation than it can in joint space. In particular, we consider learning Cartesian space offsets (Δp_{RL}) to modify the existing optimal trajectory, which will be combined with the existing jumping controller in Eq. (6). The corresponding torque contribution from these Cartesian space offsets can be written as:

$$\tau_{\text{RL Cartesian}} = J(q)^{\top} \left[K_p \left(\Delta p_{RL} - p \right) \right] \tag{7}$$

Such Cartesian space offsets could result in significant deviations in the optimal joint trajectories (q_d). To avoid joint space gain feedback from countering these deviations, we also add offsets in joint space corresponding to those desired in Cartesian space as:

$$\Delta q_{RL} = J(q_d)^{\top} \Delta p_{RL} \tag{8}$$

This makes the joint space reinforcement learning torque contribution as follows:

$$\tau_{\text{RL Joint}} = K_{p,joint} (\Delta q_{RL} - q) \tag{9}$$

The full controller for tracking the desired trajectories with learned reinforcement residual offsets is then the summation of the original jumping controller (Eq. (6)) with the offset contributions (Eqs. (7) and (9)):

$$\tau = \tau_{\text{opt}} + \tau_{\text{RL Cartesian}} + \tau_{\text{RL Joint}} \tag{10}$$

3.2. Reinforcement learning details

There are several challenging aspects of jumping that motivate our observation space, action space, and reward function choices. Firstly, while in the air, it is very difficult to meaningfully adjust body position or orientation. Secondly, even small noise or deviations in the trajectory before take-off can have a large effect on landing location
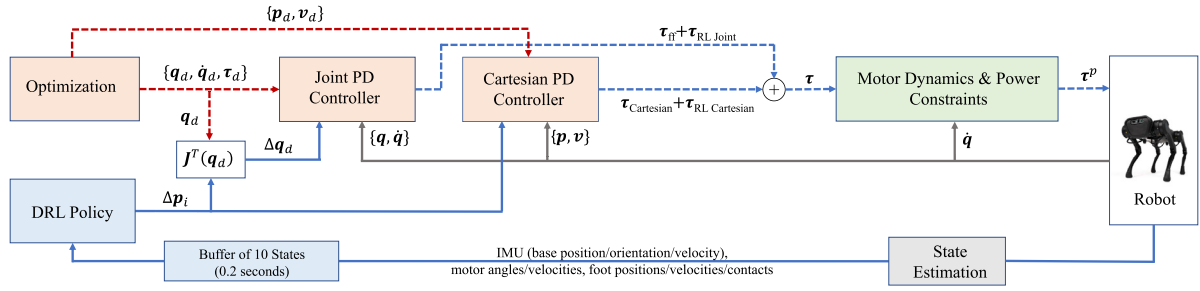

Fig. 2. Jumping motion phases from the trajectory optimization.

all-leg contact phase
(T=0.5 s)

rear-leg contact phase
(T=0.3 s)

flight phase
(T is task-dependent)

2.3. Jumping trajectory optimization

In this section we briefly describe the trajectory optimization framework to generate quadruped jumping motions, as well as the associated jumping controller. For full details, please see our prior work [8].

The robot model used in the trajectory optimization framework is a simplified sagittal plane quadruped model consisting of 5 links. The body link coordinates are represented by the position of the center of mass $[p_x, p_z]$ and the rotational angle (pitch) of the body θ, while the configuration of the other links (limbs) is denoted by q. The optimization problem is divided into 3 contact phases: double contact (pre-flight preparation), single contact (rear-leg), and flight, as shown in Fig. 2. The duration of each phase is manually determined based on desired jumping distance and height.

At a high level, the resulting discrete time optimization can be formulated as follows:

$$\underset{x_k, u_k; \ k=1...N}{\text{minimize}} \ J(x_N) + h \sum_{k=1}^{N} w(x_k, u_k)$$

$$\text{subject to } d(x_k, u_k, x_{k+1}) = 0, \ k = 1...N-1$$

$$\phi(x_k, u_k) = 0, \ k = 1...N$$

$$\psi(x_k, u_k) \leq 0, \ k = 1...N$$

where $x_k = \left[p_{x,k}; \ p_{z,k}; \ \theta_k; \ q_k \right]$ is the full state of the system at sample k along the trajectory, u_k is the corresponding control input, J and w are final and additive costs to jump to a particular height and distance while minimizing energy, h is the time between sample points k and $k+1$, and N is the total number of samples along the trajectory. The constraints are specified as follows:

- The function $d(\cdot)$ captures the full-body dynamic constraints [8], which is discretized from

$$\begin{bmatrix} M & -J_c^T \\ -J_c^T & 0 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ f_c \end{bmatrix} = \begin{bmatrix} -C\dot{x} - g + S\tau + S_{fric}\tau_{fric} \\ \dot{J}_c(x)\dot{x} \end{bmatrix},$$

where M is the mass matrix, C represents the Coriolis and centrifugal terms, g denotes the gravity vector, J_c is the spatial Jacobian expressed at the foot contact, S and S_{fric} are distribution matrices of actuator torques τ and joint friction torques τ_{fric}, f_c is the spatial force at the foot contact. The dimensions of J_c and f_c depend on the contact phases.
- The function $\phi(\cdot)$ represents equality constraints on initial joint and body configurations, pre-landing configuration, and final body configuration.
- The function $\psi(\cdot)$ captures inequality constraints including joint angle/velocity/torque limits, friction cone limits, minimum ground reaction forces, and geometric constraints related to the ground and obstacle clearance.

The optimization produces desired joint angles (q_d), joint velocities ($\dot{q}_d$) and feed-forward joint torques (τ_d) at a sampling time of 10 ms, which are then linearly interpolated to 1 ms. These can be tracked by the following joint PD controller running at 1 kHz as:

$$\tau_{\text{ff}} = K_{p,joint}(q_d - q) + K_{d,joint}(\dot{q}_d - \dot{q}) + \tau_d \tag{4}$$

Fig. 3. Control diagram for learning to jump robustly with reinforcement learning by leveraging the optimized jumping controller, motor dynamics, and power constraints. The combination of trajectory reference from the optimization (dotted red lines) and trajectory offsets from the DRL policy are then tracked by joint PD and Cartesian PD controllers. The dotted lines and gray lines execute at 1 kHz, while the solid blue lines execute at 50 Hz during contact phases. The motor dynamics and power constraints module is designed to represent the hardware capability, which limits the final torque reference τ^p for the robot's motors.

and orientation. Thirdly, the entire motion happens very quickly, with the pre-jump phase taking only 0.8 s, and the flight phase roughly 0.4 s, depending on distance and height. To mitigate these issues, we learn actions to modulate movement while in contact only, and apply τ_{opt} in the air. Fig. 3 shows the full block diagram for integrating our reinforcement learning action with the optimal jumping controller, and we describe the MDP below.

Action Space: The action space consists of the desired trajectory offsets $\mathbf{a} \in \mathbb{R}^{12}$ for the contact phase of the jumping trajectory, which are updated at 50 Hz. The agent chooses offsets in $[-0.05, 0.05]$ m from each foot's local desired (x, y, z) Cartesian positions from the optimization.

Observation Space: The observation space consists of the full robot state at the initial state, trajectory end state (goal), as well as a history of states in the previous 0.2 s. The history of states is a stack of 10 observations updated at 50 Hz. Each of these states consists of: body state (position, orientation, linear and angular velocities), joint state (positions, velocities), foot state (positions, velocities), and foot contact booleans. All values are first normalized before being used for training purposes by reinforcement learning.

Reward: We give a sparse, single reward at the end of the jumping trajectory based on the error between the desired and actual landing position and orientation. We give a sparse reward, rather than dense rewards for tracking the optimized jumping trajectory at every time step, as significant deviations to the offline optimized trajectory can be expected (and will be needed) for large environmental noise. If there is large noise such as low coefficient of friction or very uneven terrain (large blocks under either the front or rear feet), executing the trajectory optimized for flat terrain will not result in a successful jump. For example, a large block under the front feet will result in over-pitch at take-off, therefore being unable to jump the desired distance (see Fig. 8 and video). Therefore, with DRL, we need to learn offsets to significantly deviate from the original trajectory to perform a successful jump, as can be seen in Fig. 12.

More precisely, the reward function attempts to minimize deviations in the body position (x_b, y_b, z_b) and orientation $(\phi_b, \theta_b, \psi_b)$ from the final desired states in the optimal trajectory: body position (x_N, y_N, z_N) and orientation $(\phi_N, \theta_N, \psi_N)$. The final orientation $(\phi_N, \theta_N, \psi_N)$ is always $(0, 0, 0)$ as we would like the agent to land upright at its standing orientation. The reward function is written as:

$$R(s_t, a_t, s_{t+1}) = w(1 - \|(x_b, y_b, z_b) - (x_N, y_N, z_N)\|$$
$$- \|(\phi_b, \theta_b, \psi_b)\|) \quad (11)$$

where w is a terminal weight. This reward scheme ensures a reward of w for perfect tracking, and will decrease from there, and even be negative, for very poor tracking.

Table 2
SAC hyperparameters.

Parameter	Value
Optimizer	Adam
Learning rate	$3 \cdot 10^{-4}$
Discount (γ)	0.99
Replay buffer size	10^6
Initial steps	1000
Number of hidden layers (all networks)	2
Number of hidden units per layer	512
Nonlinearity	tanh
Batch size	64
Target smoothing coefficient (τ)	0.005
Target update interval	1
Gradient steps	1

3.3. Training details

We first generate 13 jumping trajectories, with final desired positions ranging in distance in $[0.5, 0.8]$ m and in height in $[0, 0.4]$ m. At the beginning of each episode, one of the trajectories is randomly selected to track, and random noise is added to the environment. The noise consists of blocks of up to 0.1 m in height under each foot, and the body mass and inertia are each varied randomly by up to 5% of their nominal values.

We use PyBullet [30] as the physics engine for training and simulation purposes, and the A1 quadruped model introduced in Section 2.1. For SAC [29], our neural networks are multi-layer perceptrons with two hidden layers of 512 neurons each, with tanh activation. Other training hyperparameters are listed in Table 2.

3.4. Motor dynamics and power constraints

Since legged robots must rapidly reach their motor and on-board power supply limits to accomplish dynamic jumping maneuvers, it is critical to model and integrate the motor dynamics and power constraints during training to represent the true system limits. This integration in turn limits the reference torque individually applied to each motor, which enables successful sim-to-real transfer for aggressive motions such as jumping.

3.4.1. Motor and power modeling

First, we revisit a simplified DC motor model which captures the inherent torque–velocity relationship. Since the inductance of stator windings is typically small (approximately 1 mH for an A1 robot motor [27]), the voltage applied to each motor $i \in \{1, \ldots, n\}$ can be simplified as follows

$$V_i(\tau_i^m, \dot{q}_i^m) = I_i^m(\tau_i^m) R_i + \varphi_i(\dot{q}_i^m), \quad (12)$$

where R_i is the resistance of the coil windings, and $\dot{q}_i^m$ is the motor velocity. The back electromotive force (EMF) of the windings generated

by the rotation of the motor is estimated by $\varphi_i(\dot{q}_i^m) = K_v \dot{q}_i^m$, and the current $I_i^m(\tau_i^m)$ flowing in the windings relates to the motor torque via $I_i^m = \tau_i^m / K_\tau$. Here, K_v and K_τ are the electric motor velocity constant and torque constant, respectively. Considering the gear ratio g_r which relates

$$\tau_i = \tau_i^m g_r, \quad \dot{q}_i = \dot{q}_i^m / g_r$$

we can rewrite the voltage Eq. (12) as a linear combination of joint torque and joint velocity as

$$V_i(\tau_i, \dot{q}_i) = \alpha \tau_i + \beta \dot{q}_i, \qquad (13)$$

where $\alpha = R_i / (K_\tau g_r)$ and $\beta = K_v g_r$, respectively.

Moreover, it is noteworthy that jumping maneuvers are highly demanding and normally quickly drain the battery's power capacity. Hence, it is also essential to consider the total power required to run all of the robot's motors in the learning framework. The total power can be estimated by:

$$\sum_{i=1}^{n} P_i(\tau_i, \dot{q}_i) = \sum_{i=1}^{n} V_i I_i = \sum_{i=1}^{n} \left(\frac{\tau_i}{K_\tau g_r} \right)^2 R_i + \frac{K_v \tau_i \dot{q}_i}{K_\tau} \qquad (14)$$

The power consists of two parts: the first part is power dissipation on the windings, which is proportional to $\tau_i^2 R_i$; the second part is the power of rotation $\tau_i \dot{q}_i$.

Having revisited the torque-speed relationship and total power estimation, we will propose and integrate motor dynamic constraints and power limits into the simulation environment.

3.4.2. Implementation of motor dynamics and power constraints

We incorporate the motor dynamics and power constraints into the DRL framework in order to enforce restrictions on the final reference torque that is applied to the motors, as depicted in Fig. 3. In particular, the final reference needs to satisfy the following conditions:

i. *Motor dynamic constraints (MDC)* establish a key relationship between joint torque and velocity in conjunction with the available voltage supply capability V_{bat}, i.e.,

$$|V_i(\tau_i, \dot{q}_i)| = |\alpha \tau_i + \beta \dot{q}_i| \le V_{bat} \qquad (15)$$

ii. *Power limits*: The total power supplied to all n motors is constrained by the power supply capability. This requires that the total power for operating all motors is limited by the battery power P_{bat}, i.e.,

$$\sum_{i=1}^{n} P_i(\tau_i, \dot{q}_i) \le P_{bat} \qquad (16)$$

It is noted that the motor dynamic constraints (15) imply that the joint torques and joint velocities cannot simultaneously reach their respective limits. In particular, the DC motor reaches maximum velocity when running at no load, and the back EMF approaches the supply voltage. Approximately, $\dot{q}_i^{max} = V_{bat}/\beta$, giving rise to the following constraints:

$$V_{bat} \ge \beta \dot{q}_i, \quad -V_{bat} \le \beta \dot{q}_i \qquad (17)$$

With these conditions established, we integrate the motor dynamics and power constraints into the simulation environment, as described in Algorithm 1.

Algorithm 1 should be executed sequentially, starting with the MDC block, followed by the power limits. In case the torque τ_i^v obtained from the MDC block violates the power constraints (16), we will proportionally decrease this torque by setting $\tau_i^p = \eta \tau_i^v$. This modified reference will then be utilized for the motor in simulation. Consequently, the task is to find a value of $\eta \in (0, 1)$ that satisfies the following quadratic equation:

$$f(\eta) \triangleq \eta^2 \sum_{i=1}^{n} \frac{R_i (\tau_i^v)^2}{K_\tau^2 g_r^2} + \eta \sum_{i=1}^{n} \frac{\tau_i^v \dot{q}_i}{K_\tau K_v} = P_{bat} \qquad (18)$$

Algorithm 1: Integration of Motor Dynamics and Power Constraints into the Simulation Environment

1 **Input**: The total torque τ (Eq. (10)), feedback joint velocity $\dot{q}$, battery and motor parameters.

2 **Output**: Final reference torque for each motor

3   *(i) Motor Dynamic Constraints (MDC):*

4 Compute: $V_i \leftarrow \alpha \tau_i + \beta \dot{q}_i$

5 **if** $V_i > V_{bat}$ **then** $\tau_i^v \leftarrow (V_{bat} - \beta \dot{q}_i)/\alpha$

6 **else if** $V_i < -V_{bat}$ **then** $\tau_i^v \leftarrow (-V_{bat} - \beta \dot{q}_i)/\alpha$

7 **else** $\tau_i^v = \tau_i$

8 **return** τ^v;

9   *(ii) Power Limits:*

10 Compute $P_{total}^d \leftarrow \sum_{i=1}^{n} P_i(\tau_i^v, \dot{q}_i)$

11 **if** $P_{total}^d > P_{bat}$ **then**

12     $\tau_i^p \leftarrow \eta \tau_i^v$: reduce torque proportionally $(0 < \eta < 1)$

13   where $\eta = \frac{\sqrt{B^2 + 4AP_{bat}} - B}{2A}$,

14 **else if** $P_{total}^d \le P_{bat}$ **then**

15     $\tau_i^p \leftarrow \tau_i^v$;

16 **end**

17 **return** τ^p;

Let $A = \sum_{i=1}^{n} \frac{R_i (\tau_i^v)^2}{K_\tau^2 g_r^2}$ and $B = \sum_{i=1}^{n} \frac{\tau_i^v \dot{q}_i}{K_\tau K_v}$, then (18) yields:

$$\eta = \frac{\sqrt{B^2 + 4AP_{bat}} - B}{2A} \qquad (19)$$

In the following, we will prove that the final output τ^p at the end of Algorithm 1 also satisfies the voltage constraints in Eq. (15), i.e., $|V_i(\tau_i^p, \dot{q}_i)| \le V_{bat}, \forall i \in \{1, \dots, n\}$. Indeed, if $P_{total}^d \le P_{bat}$, then Algorithm 1 assigns $\tau_i^p = \tau_i^v$. This output torque τ_i^p trivially satisfies the voltage constraints

$$|V_i(\tau_i^p, \dot{q}_i)| = |\alpha \tau_i^p + \beta \dot{q}_i| = |\alpha \tau_i^v + \beta \dot{q}_i| \le V_{bat} \qquad (20)$$

Therefore, it remains to be shown that $|V_i(\tau_i^p, \dot{q}_i)| \le V_{bat}$ when $\tau_i^p = \eta \tau_i^v$, $\eta \in (0, 1)$ for the case $P_{total}^d > P_{bat}$.

We start with the voltage values, which are obtained from the MDC block: $|V_i(\tau_i^v, \dot{q}_i)| = |\alpha \tau_i^v + \beta \dot{q}_i| \le V_{bat}$. This inequality is equivalent to $(-V_{bat} - \beta \dot{q}_i)/\alpha \le \tau_i^v \le (V_{bat} - \beta \dot{q}_i)/\alpha$. Then, by multiplying all sides of the inequalities by $\alpha \eta$, we obtain $\eta (-V_{bat} - \beta \dot{q}_i) \le \alpha \tau_i^p \le \eta (V_{bat} - \beta \dot{q}_i)$. Therefore, adding the term $\beta \dot{q}_i$ to the inequalities yields

$$(1 - \eta)\beta \dot{q}_i - \eta V_{bat} \le V_i^p \le (1 - \eta)\beta \dot{q}_i + \eta V_{bat} \qquad (21)$$

Combining with Eq. (17) and $0 < \eta < 1$, one can verify that

$$V_i^p - V_{bat} \le (1 - \eta)(\beta \dot{q}_i - V_{bat}) \le 0 \qquad (22a)$$

$$V_i^p + V_{bat} \ge (1 - \eta)(\beta \dot{q}_i + V_{bat}) \ge 0 \qquad (22b)$$

This yields $|V_i^p| \le V_{bat}, \forall i \in \{1, \dots, n\}$, and the proof is complete.

In conclusion, our proposed algorithm computes final torque τ^p that theoretically guarantees both motor dynamic constraints (Eq. (15)) and power limits (Eq. (16)). In hardware experiments, we will verify the key role of these constraints in attaining effective sim-to-real transfers.

4. Results

In this section, we discuss results from using our method to achieve robust jumping. Example snapshots of the jumping task are shown in Figs. 1, 4, 5, 8, and the reader is encouraged to watch the supplementary videos[2,3] for clearer visualizations. In particular, we show the

---

[2] Simulation video: https://youtu.be/Y44GK__QuY.

[3] Experiment video: https://youtu.be/fVxXTofT1f4.
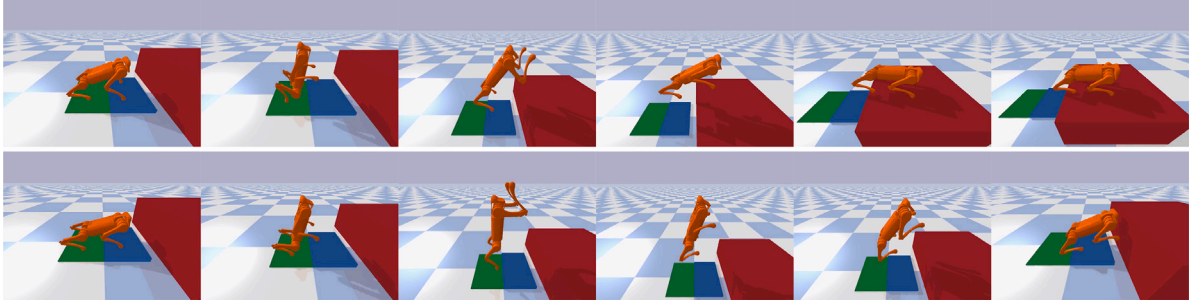
Fig. 4. Motion snapshots of a jump of distance 0.7 m and height 0.4 m. The front feet have a 0.05 m block beneath them, and the rear feet have a 0.01 m block beneath them. **Top:** The learned policy successfully outputs trajectory offsets to jump onto the platform. **Bottom:** The feedforward controller results in overpitch and overjumps vertically, falling short of the platform.
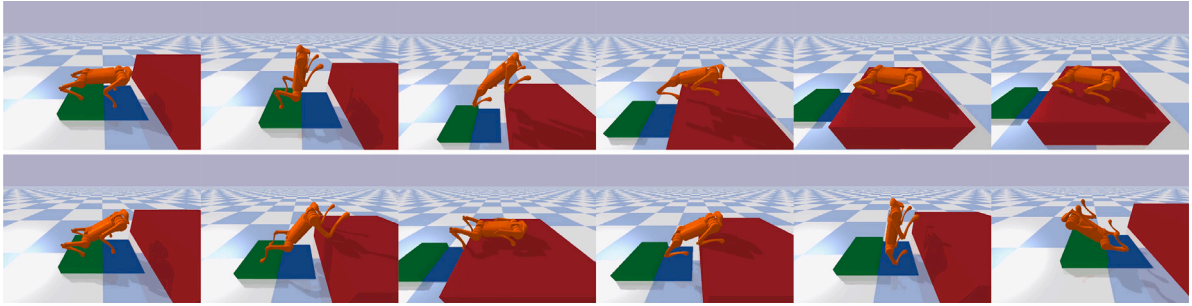
Fig. 5. Motion snapshots of a jump of distance 0.7 m and height 0.4 m. The front feet have a 0.01 m block beneath them, and the rear feet have a 0.1 m block beneath them. **Top:** The learned policy successfully outputs trajectory offsets to jump onto the platform. **Bottom:** The feedforward controller results in underpitch and overjumps horizontally, making the rear legs catch on the edge of the platform, resulting in falling off.
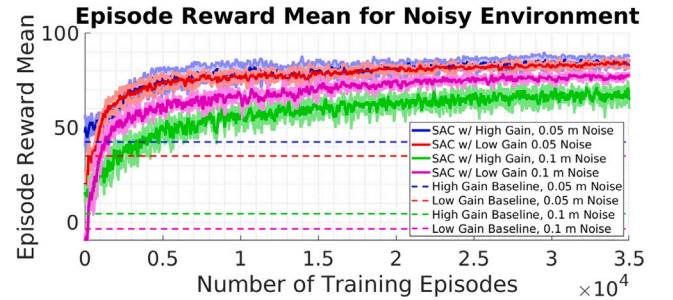

Fig. 6. Episode reward mean while training under ideal conditions. The baseline feedforward controllers' performance are shown as dotted lines. Our framework is able to track the trajectories accurately for either set of joint gains studied.

Fig. 7. Episode reward mean while training with noisy environment conditions: either up to 0.05 m or 0.1 m blocks under each foot, and base mass/inertia vary by up to 5% of their nominal values. While the feedforward controllers' performance is extremely poor, our method is able to learn to jump accurately through significantly noisy environment conditions.

results of zero-shot sim-to-real transfers of the learned trajectory-offset policies from PyBullet to the Unitree A1 hardware.

For our experiments, we are specifically interested in the following questions:

1. How does choice of joint gain affect tracking performance?
2. Can we improve upon tracking performance in ideal conditions?
3. How does (magnitude of) noise affect the agent's ability to learn?
4. What is the importance of integrating both motor dynamics constraints and power constraints into the learning environment?

We consider two different sets of joint gains, which we name "high" ($K_{p,joint} = 300I_3$, $K_{d,joint} = 3I_3$) and "low" ($K_{p,joint} = 100I_3$, $K_{d,joint} = 2I_3$) gains. Oftentimes these gains must be tuned by hand, and may also need to be adapted for different trajectories. Thus, our goal with learning with different gains is that it may give some insight on if we can indirectly tune these all at once for multiple trajectories by

selecting trajectory offsets, rather than manual human trial and error. We set the Cartesian gains as $K_p = 500I_3$, $K_d = 10I_3$.

4.1. Simulation results

Fig. 6 shows training results for learning to offset the trajectories under ideal conditions. With the default baseline "high" joint gain jumping controller, the tracking is already very accurate, getting close to $w = 100$ rewards. On the other hand, the baseline "low" gain controller does not perform as well, primarily due to errors in pitch when at the end of the trajectory, as well as falling short in distance. However, through our framework, we are able to accurately track the desired jumping motions using either set of gains, though the "high" gains still result in slightly better performance. This shows that our framework is general enough to learn to improve several jumping behaviors without the need to explicitly tune gains on a per-motion basis, as may often be needed in general.
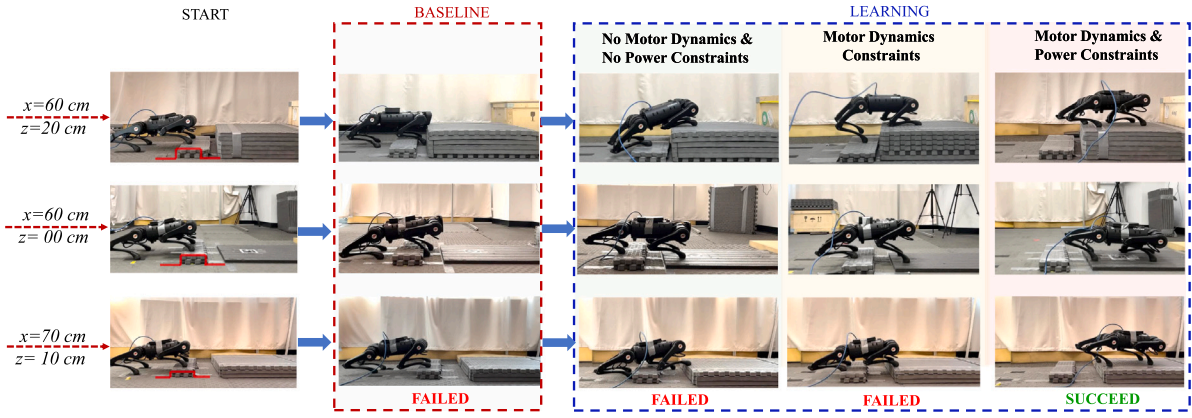
Fig. 8. Jumping different target heights and distances under unknown disturbances. The figures show baseline experiments (feedforward controller) and sim-to-real transfers for different subsets of motor dynamics and power constraints integration during the learning process. For each jumping target, the robot starts with the same initial configuration, and an unknown disturbance of a 6 cm block (red line) is placed under the front feet. We use the same controller gains for all experiments: $K_{p,joint} = 300, K_{d,joint} = 3$. Experiment video link: https://youtu.be/fVxXTofT1f4. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 7 shows training results for learning to offset the trajectories under the noisy conditions described in 3.3. We train under two sets of noisy environment conditions: with either up to 0.05 m height noise, or up to 0.1 m height noise under each foot, in addition to the variability in body mass and inertia. The baseline controllers for either set of gains are not able to accurately track the desired motions, predominantly due to over/under pitching during the single contact rear back phase. This becomes especially apparent as we increase the environment noise to 0.1 m, where under our reward scheme, the feedforward controller averages approximately 0 reward across 100 random trials, for either set of joint gains, corresponding to extremely poor performance where the robot is not even close to the goal location.

The bottom row of Fig. 4 shows the over-pitching behavior of the baseline controller when the front legs start higher than the rear ones, during one of the more difficult jumps in terms of height and distance. This results in jumping vertically and not coming close to landing on the platform.

The bottom of Fig. 5 shows the opposite result (under-pitching) when the rear feet start at a higher z height than that of the front feet. In this case, the baseline feedforward controller does not pitch enough before take off, leading to a more horizontal jump that crashes horizontally into the platform. For both of these scenarios, our learned controller is able to successfully jump onto the platform, as can be seen in the top rows of Figs. 4 and 5.

A noteworthy observation is that while the "low" gain baseline performance (as well as when training with our method) is not as good as the "high" gain controller for ideal conditions, as the noise increases significantly, we see that the agent is able to exploit the lower gain joint controller to outperform the policy using the high gain controller, as can be seen in Fig. 7.

These results show that through our method, using either set of gains, we are able to learn offsets to significantly improve jumping performance under noisy environmental conditions, close to as well as under ideal conditions.

4.2. Experimental verification

We validate the effectiveness of our proposed learning framework, which incorporates motor dynamics and power constraints, in enabling robust jumping on the Unitree A1 robot hardware. We conduct various experiments with different jumping targets of $(x, z) \in \{(60, 20), (60, 0), (70, 10)\}$ (cm), and different block disturbances $\{3, 6\}$ (cm) introduced under the robot's feet. We focus our discussion on the 6 cm disturbance, as illustrated in Fig. 8. This disturbance amounts to 33% of the robot's initial height and is not explicitly known by the agent.

In the baseline experiments, we only use the joint PD and Cartesian PD controller to track the joint and foot trajectory references from the full-body trajectory optimization, as described in Section 2.3. Since there is no feedback controller to compensate for the uneven terrain disturbance, robot trajectory errors propagate during the jumping process, resulting in overpitching before take-off and failed jumps (Fig. 8-Baseline).

In contrast to the baseline, our learned jumping policy is run as a real-time feedback controller to compensate jumping trajectory errors. In order to verify the effectiveness of integrating motor dynamics and power constraints into the learning environment, we compare the sim-to-real transfer performance of training controllers with the following subset of constraints:

1. No motor dynamics nor power constraints.
2. Only motor dynamics constraints.
3. Both motor dynamics and power constraints.

Case 1 - Learning with No Constraints: In the first learning experiment for each target, we only consider a naïve implementation of torque limits that is widely utilized for learning locomotion (e.g., [17,18,23, 24]), in which only a saturation function is applied for the final torque command. As can be seen in Fig. 8, the robot has learned to compensate for an unknown noise of a 6 cm block, thereby jumping farther than the baseline cases. However, jumping onto the $(x, z) = (60, 20)$ cm box, for example, requires high voltage of up to 30 V for the motors of the rear right leg at the time of taking off, as illustrated in Fig. 9b&d (roughly at 840 ms). It also demands a significant total power supply of approximately 3750 W (Fig. 10). These requirements exceed the battery capability of $V_{bat} = 21.5$ V, $P_{bat} = 1290$ W (i.e. violates both motor dynamics and battery power limits). These violations cause poor tracking performance and result in the robot falling short of the target, as can be seen in Fig. 8.

Case 2 - Learning with Only MDC: For the second learning experiment, our policy learns to output actions which are then constrained by the torque-speed relationship throughout the whole jump, as described in Eq. (15). Therefore, the voltages for both thigh and calf motors are always within the limits (Fig. 9b&d). However, the robot still fails to reach the target because this aggressive motion did not consider the power limits. In particular, this jumping motion requires up to 2500 W when taking off, which is nearly double the maximum battery power (Fig. 10).

Case 3 - Learning with MDC and Power Constraints: The third learning experiment demonstrates the importance of considering both Motor Dynamic Constraints and Power Constraints in order to realize successful sim-to-real transfers for highly aggressive jumping maneuvers. The
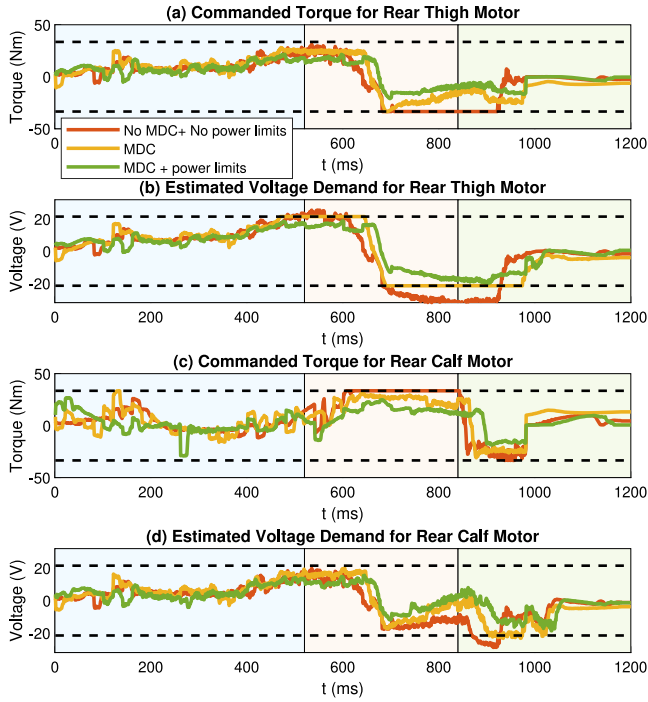
Fig. 9. Experiments for jumping on box $(x, z) = (60, 20)$ cm. Estimated voltage demands and total torque commands for the Rear Right leg thigh and calf motors. For this motion, the rear legs typically require more torques than front legs, so it is sufficient to consider the torque and voltage profiles for the rear legs. The maximum motor torques and battery voltage are specified by dotted black lines. The single-leg contact and flight phases start at approximately 520 ms and 840 ms, which are specified as vertical black lines.

Fig. 10. Estimated total power requirement to operate all motors for jumping onto a $(x, z) = (60, 20)$ cm box, associated with Fig. 9. The dotted black line represents the maximum power of the onboard battery supply.

Fig. 11. Experiments for different jumping targets. The estimated total power demand for jumping to different targets with policies trained with both motor dynamics and power constraints, corresponding to Fig. 8. The maximum power of the battery ($P_{max} \approx 1290$ W) is specified as the dotted black line.
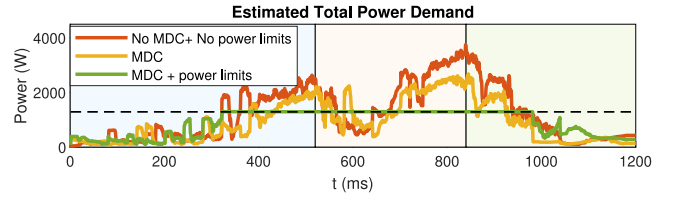
proposed integration of both motor dynamics and power limits ensure (i) the voltage demanded for operating the motors can be supplied by the battery and (ii) the required total power for all motors satisfies the on-board power supply. Both of these limits can be verified to not be violated in Figs. 9b&d and 10 for the example target jump of $(x, z) = (60, 20)$ cm. Additionally, our method enables the robot to successfully reach various jumping targets while ensuring robustness against unknown and large disturbances (e.g. 6 cm block), as illustrated in Fig. 8.

A noteworthy observation from Fig. 11 is that all jumping motions rapidly reach the battery power limits, even for jumping forward without a desired height goal $(x, z) = (60, 0)$ cm. Our learning framework, which integrates motor dynamics and power limits, provides a practical solution to achieve various jumping targets despite the limited power capacity of the onboard battery.
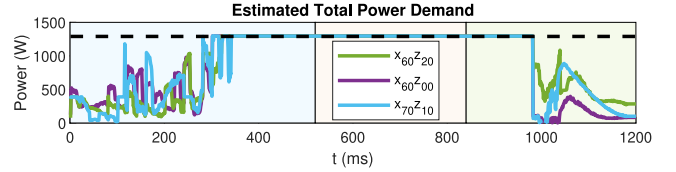
Fig. 12 shows that the learned policy significantly deviates from the original optimized trajectory in order to perform a successful jump. With the 0.06 m block underneath the front feet, the baseline optimized trajectory causes an over-pitch of the robot, which is then only able to jump half of the desired distance (green line). We also note that the DRL method has learned to significantly modulate the foot position in order to successfully complete the jump, as can be seen by the front and rear foot trajectories compared with the baseline in Fig. 12b&c.

4.3. Ablation studies

In this section we perform several hardware ablation studies on different components of our framework: high vs. low joint gains, dense vs. sparse rewards, and varying policy update rates.

4.3.1. Hardware joint gains

We validate the importance of selecting the "high" joint gains (vs. the "low" joint gains) for the hardware studies by performing several jumps with both sets of gains. During simulations, employing soft gains $(K_p, K_d) = (100, 2)$ yields satisfactory joint tracking performance. However, transitioning this soft gain configuration to the hardware setup fails to achieve comparable tracking performance. Fig. 13 illustrates the disparity observed when employing different joint gains for a jumping task of 60 cm under disturbance in the hardware environment. Specifically, we focus on the joint tracking of the rear legs for discussion. As depicted in Fig. 13, adopting higher joint gains (Fig. 13b&d) demonstrates superior joint tracking performance compared to lower joint gains (Fig. 13a&c). The issue with tracking performance associated with lower joint gains is attributed to hardware motor properties. Inaccurate joint tracking has the potential to induce undesirable foot bouncing motions, thereby adversely affecting the overall jumping performance.

4.3.2. Dense vs. sparse rewards

In our method, we train policies to perform jumps with only a single sparse reward at the end of the jump to evaluate the error with the desired landing location and orientation. We now investigate dense rewards, where the agent receives feedback after every timestep based on the error between the base position (x_b, y_b, z_b) and orientation $(\phi_b, \theta_b, \psi_b)$ with respect to the corresponding desired position/orientation at that time index in the optimized trajectory (subscript d):

$$r_{\text{dense}}(s_t, a_t, s_{t+1}) = (1 - \|(x_b, y_b, z_b) - (x_d, y_d, z_d)\|$$
$$- \|(\phi_b, \theta_b, \psi_b) - (\phi_d, \theta_d, \psi_d)\|) \tag{23}$$

We train policies with dense rewards and compare the final tracking performance across several different jumps to evaluate the importance of the reward function decision. We consider 10 different jumping tests to the target $(x, z) = (60, 0)$ cm with the following block disturbance heights (in cm) underneath the front and rear legs:

$$(h_{\text{front}}, h_{\text{rear}}) = \{(10, 0), (8, 1), (8, 3), (7, 5), (5, 7),$$
$$(6, 4), (4, 6), (3, 8), (1, 8), (0, 10)\} \tag{24}$$

We compute the error norm of the final location and orientation of the actual jump with the target location and orientation for (1) the sparse reward policy and (2) the dense reward policy. As shown
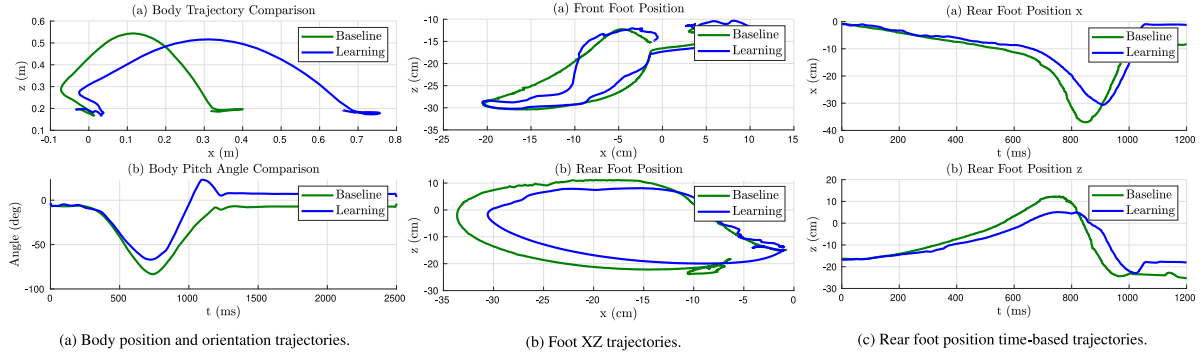
Fig. 12. Baseline and learned policy comparison: body position and orientation trajectories, foot XZ trajectories, and rear foot position time-based trajectories for the 0.06 m front foot jumping disturbance. The learned policy significantly deviates from the original baseline trajectory in order to successfully complete the jump.
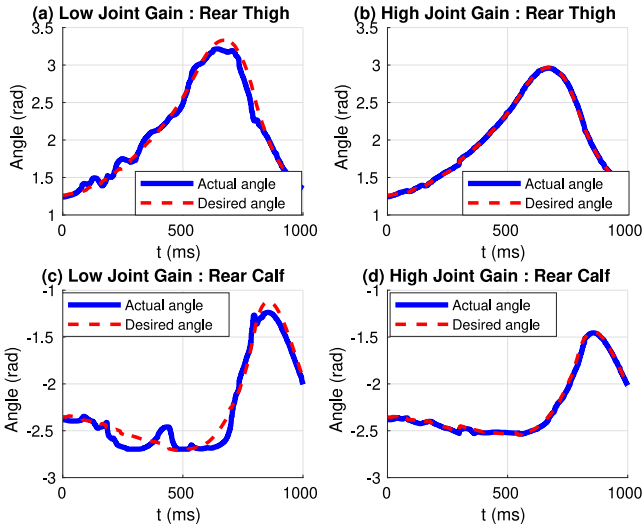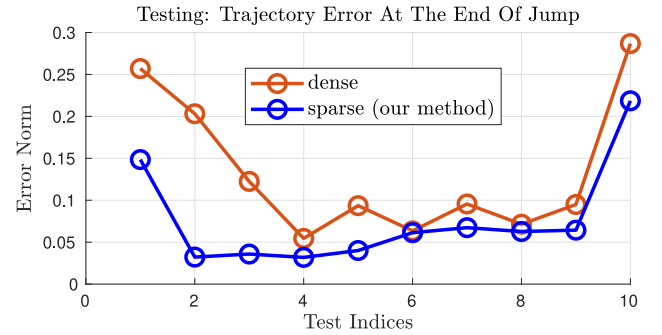

Fig. 13. Joint gain comparison hardware experiments: comparison of joint tracking performance when using low vs. high joint gains.

Fig. 14. Tracking performance with dense and sparse reward policies: comparison of the actual error of the final position and orientation with both policies for the 10 jumps from initial disturbance heights in order shown in Eq. (24).

in Fig. 14, the policies trained with sparse rewards yield smaller error norms than those trained with dense rewards. Therefore, unlike most learning-based locomotion controllers, our observations during the jumping task suggest that dense rewards do not offer significant advantages over sparse rewards. In fact, for the jumping task, prioritizing target jumping proves more crucial than tracking the entire reference trajectory. With dense rewards, the agent may compromise some degree of final target tracking for overall reference tracking. Conversely, employing sparse rewards that only consider the final jumping goal location enhances target jumping performance.

4.3.3. Policy update rate

While our nominal policy runs at 50 Hz, we investigate training additional policies at different rates, including both at a lower update rate (20 Hz), and at a higher update rate (100 Hz). We find it is possible to have both a higher and lower frequency update for the learning policy, without any observable (quantitative or qualitative) differences in performance. We use the same history observation window size of $L = 10$, but also test different window sizes in order to correspond to the same history time interval. All frequency rates result in successful training and jumping. This is inline with recent studies on locomotion (without jumping), such as [31,32], which find that a range of policy update rates and action delays can still produce successful locomotion gaits. As shown in the video, we validate the successful jumping experiments with different frequency rates in hardware experiments.

5. Conclusion

In this work, we have proposed a method to improve jumping performance of optimal trajectories with deep reinforcement learning. Instead of learning from scratch, we learn to modify and augment the existing trajectories in Cartesian space, which proved to be robust to significantly varying environmental conditions. In addition to robustness to environmental perturbations, we showed robustness to different joint gains, and further benefits include avoiding re-running potentially expensive optimization routines at run time for changes to the initial robot state, uncertainty in the system dynamics, and uncertainty in the environment such as varying uneven terrain and coefficients of friction. In order to realize highly aggressive jumps on hardware, we proposed and integrated motor dynamics and power limits as key components of the learning environment, enabling effective sim-to-real transfers without any further tuning. Our results demonstrate full exploitation of the available hardware power and motor limits to jump twice the body length in distance while subject to uneven terrain noise of 33% of the nominal standing height.

CRediT authorship contribution statement

Guillaume Bellegarda: Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft. **Chuong Nguyen:** Conceptualization, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – review & editing. **Quan Nguyen:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Writing – review & editing.

Declaration of competing interest

Data availability

Data will be made available on request.

Acknowledgments

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2024.104799.

References

[1] Boston Dynamics, Atlas gets a grip, 2023, https://www.youtube.com/watch?v=-e1_QhJ1EhQ.

[2] B. Katz, J. Di Carlo, S. Kim, Mini cheetah: A platform for pushing the limits of dynamic quadruped control, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 6295–6301.

[3] Boston Dynamics, Introducing handle, 2017, https://www.youtube.com/watch?v=-7xvqQeoA8c.

[4] V. Klemm, A. Morra, C. Salzmann, F. Tschopp, K. Bodie, L. Gulich, N. Küng, D. Mannhart, C. Pfister, M. Vierneisel, et al., Ascento: A two-wheeled jumping robot, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 7515–7521.

[5] H.-W. Park, P.M. Wensing, S. Kim, High-speed bounding with the MIT Cheetah 2: Control design and experiments, Int. J. Robot. Res. 36 (2) (2017) 167–192.

[6] G. Bellegarda, M. Shafiee, M.E. Özberk, A. Ijspeert, Quadruped-Frog: Rapid online optimization of continuous quadruped jumping, in: 2024 IEEE International Conference on Robotics and Automation, ICRA, 2024, pp. 1443–1450.

[7] C. Nguyen, L. Bao, Q. Nguyen, Continuous jumping for legged robots on stepping stones via trajectory optimization and model predictive control, in: 2022 IEEE 61th Conference on Decision and Control, CDC, 2022, pp. 93–99.

[8] Q. Nguyen, M.J. Powell, B. Katz, J.D. Carlo, S. Kim, Optimized jumping on the MIT Cheetah 3 robot, in: 2019 International Conference on Robotics and Automation, ICRA, 2019, pp. 7448–7454, http://dx.doi.org/10.1109/ICRA.2019.8794449.

[9] C. Nguyen, Q. Nguyen, Contact-timing and trajectory optimization for 3D jumping on quadruped robots, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2022, pp. 11994–11999.

[10] H. Kolvenbach, E. Hampp, P. Barton, R. Zenkl, M. Hutter, Towards jumping locomotion for quadruped robots on the Moon, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2019, pp. 5459–5466.

[11] Y. Ding, H.-W. Park, Design and experimental implementation of a quasi-direct-drive leg for optimized jumping, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2017, pp. 300–305.

[12] D.W. Haldane, M.M. Plecnik, J.K. Yim, R.S. Fearing, Robotic vertical jumping agility via series-elastic power modulation, Science Robotics 1 (1) (2016).

[13] M. Noh, S.-W. Kim, S. An, J.-S. Koh, K.-J. Cho, Flea-inspired catapult mechanism for miniature jumping robots, IEEE Trans. Robot. 28 (5) (2012) 1007–1018.

[14] J.K. Yim, E.K. Wang, R.S. Fearing, Drift-free roll and pitch estimation for high-acceleration hopping, in: 2019 International Conference on Robotics and Automation, ICRA, IEEE, 2019, pp. 8986–8992.

[15] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, M. Hutter, Learning agile and dynamic motor skills for legged robots, Science Robotics 4 (26) (2019) http://dx.doi.org/10.1126/scirobotics.aau5872.

[16] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning quadrupedal locomotion over challenging terrain, Science Robotics 5 (47) (2020) http://dx.doi.org/10.1126/scirobotics.abc5986.

[17] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, Sim-to-Real: Learning agile locomotion for quadruped robots, in: Proceedings of Robotics: Science and Systems, Pittsburgh, Pennsylvania, 2018, http://dx.doi.org/10.15607/RSS.2018.XIV.010.

[18] X.B. Peng, E. Coumans, T. Zhang, T.-W.E. Lee, J. Tan, S. Levine, Learning Agile robotic locomotion skills by imitating animals, in: Robotics: Science and Systems, 2020, http://dx.doi.org/10.15607/RSS.2020.XVI.064.

[19] A. Kumar, Z. Fu, D. Pathak, J. Malik, Rma: Rapid motor adaptation for legged robots, in: Robotics: Science and Systems, 2021.

[20] G. Ji, J. Mun, H. Kim, J. Hwangbo, Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion, IEEE Robot. Autom. Lett. 7 (2) (2022) 4630–4637.

[21] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning robust perceptive locomotion for quadrupedal robots in the wild, Science Robotics (2022).

[22] G. Bellegarda, K. Byl, Training in task space to speed up and guide reinforcement learning, in: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2019, pp. 2693–2699.

[23] G. Bellegarda, Y. Chen, Z. Liu, Q. Nguyen, Robust high-speed running for quadruped robots via deep reinforcement learning, in: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2022, pp. 10364–10370, http://dx.doi.org/10.1109/IROS47612.2022.9982132.

[24] G. Bellegarda, A. Ijspeert, CPG-RL: Learning central pattern generators for quadruped locomotion, IEEE Robot. Autom. Lett. 7 (4) (2022) 12547–12554.

[25] A. Iscen, G. Yu, A. Escontrela, D. Jain, J. Tan, K. Caluwaerts, Learning agile locomotion skills with a mentor, in: 2021 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2021, pp. 2019–2025.

[26] V. Atanassov, J. Ding, J. Kober, I. Havoutis, C. Della Santina, Curriculum-based reinforcement learning for quadrupedal jumping: A reference-free design, 2024, arXiv preprint arXiv:2401.16337.

[27] Unitree Robotics, A1, 2021, https://www.unitree.com/products/a1/.

[28] R.S. Sutton, A.G. Barto, Reinforcement Learning - An Introduction, in: Adaptive Computation and Machine Learning, MIT Press, 1998.

[29] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: International Conference on Machine Learning, PMLR, 2018, pp. 1861–1870.

[30] E. Coumans, Y. Bai, PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2016–2019, http://pybullet.org.

[31] S. Gangapurwala, L. Campanaro, I. Havoutis, Learning low-frequency motion control for robust and dynamic robot locomotion, in: 2023 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2023, pp. 5085–5091.

[32] G. Bellegarda, M. Shafiee, A. Ijspeert, Visual CPG-RL: Learning central pattern generators for visually-guided quadruped locomotion, in: 2024 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2024, pp. 1420–1427.

Guillaume Bellegarda is a postdoctoral researcher in the Institute of Mechanical Engineering at École Polytechnique Fédérale de Lausanne (EPFL). He was previously a postdoctoral researcher at University of Southern California, and received his Ph.D. and M.S. degrees in Electrical and Computer Engineering from University of California, Santa Barbara, and his B.S. degree in Electrical Engineering and Computer Science from University of California, Berkeley. His research draws inspiration from machine learning, model-based control, and neuroscience to maximize explainable performance for dynamic robotic systems, as well as to deepen understanding of their biological system counterparts to adapt to real world situations.

Chuong Nguyen is a Ph.D. student in the Department of Aerospace and Mechanical Engineering at the University of Southern California. He earned M.Sc. degrees in Mechanical Engineering from University of Southern California in 2023, and from Gwangju Institute of Science and Technology in 2018. Before that, he achieved a B.Sc degree in Control and Automation Engineering in 2014 from Hanoi University of Science and Technology, Hanoi, Vietnam. His research interests span the intersection of control, optimization and learning approaches for dynamic legged systems and multi-agent dynamic systems.

Quan Nguyen is an Assistant Professor of Aerospace and Mechanical Engineering at the University of Southern California. Prior to joining USC, he was a Postdoctoral Associate in the Biomimetic Robotics Lab at the Massachusetts Institute of Technology (MIT). He received his Ph.D. from Carnegie Mellon University (CMU) in 2017 with the Best Dissertation Award. His research interests span different control and optimization approaches for highly dynamic robotics including nonlinear control, trajectory optimization, real-time optimization-based control, robust and adaptive control.