Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization

Miroslav Bogdanovic[1]*, Majid Khadiv[1] and Ludovic Righetti[1,2]

[1]Movement Generation and Control Group, Max Planck Institute for Intelligent Systems, Tübingen, Germany, [2]Machines in Motion Laboratory, Tandon School of Engineering, New York University, New York, NY, United States

We present a general, two-stage reinforcement learning approach to create robust policies that can be deployed on real robots without any additional training using a single demonstration generated by trajectory optimization. The demonstration is used in the first stage as a starting point to facilitate initial exploration. In the second stage, the relevant task reward is optimized directly and a policy robust to environment uncertainties is computed. We demonstrate and examine in detail the performance and robustness of our approach on highly dynamic hopping and bounding tasks on a quadruped robot.

1 Introduction

Deep reinforcement learning (DRL) has recently shown great promises to control complex robotic tasks, e.g., object manipulation Kalashnikov et al. (2018), quadrupedal Hwangbo et al. (2019) and bipedal Xie et al. (2020) locomotion. However, exploration remains a serious challenge in RL, especially for legged locomotion control, mainly due to the sparse rewards in problems with contact as well as the inherent under-actuation and instability of legged robots. Furthermore, to successfully transfer learned control policies to real robots, there is still no consensus among researchers about the choice of the action space Peng and van de Panne, (2017) and what (and how) to randomize in the training procedure to generate robust policies Xie et al. (2021b).

Trajectory optimization (TO) is a powerful tool for generating stable motions for complex and highly constrained systems such as legged robot (Winkler et al., 2018; Carpentier and Mansard, 2018; Ponton et al., 2021). However, re-planning trajectories through a model predictive control (MPC) scheme is still a challenge, because the computation time for solving a high-dimensional non-linear program in real-time remains too high. Furthermore, apart from recent works explicitly taking into account contact uncertainty to design robust control policies (Drnach and Zhao, 2021; Hammoud et al., 2021), the inclusion of robustness objectives in trajectory optimization can quickly

end up in problems that cannot be solved in real time for high-dimensional systems in multi-contact scenarios.

In this work, we propose a general approach allowing for the generation of robust policies starting from a single demonstration. The main idea is to use TO to generate trajectories for different tasks that are then used for exploration for DRL.

Reinforcement learning for locomotion: The use of reinforcement learning for generating stable locomotion patterns is a relatively old problem (Hornby et al., 2000; Kohl and Stone, 2004). However, these works mostly use a hand-crafted policy with very few policy parameters that are tuned on the hardware. Later works use a notion of Poincare map to ensure the cyclic stability of the gaits (Morimoto et al., 2005; Tedrake et al., 2005). These approaches have enabled later a humanoid robot to walk Morimoto and Atkeson, (2009), but their underlying function approximator cannot handle large number of policy parameters which limits their application. Furthermore, the robot hardware at that time were not capable of dynamic movements which made the researchers focus mostly on the walking problem. Later, (Fankhauser et al., 2013), used more scalable approaches (PI2 algorithm by Theodorou et al. (2010)) for generating a hopping motion on a single planar leg. Recently, deep reinforcement learning has become the main approach for learning both locomotion and manipulation policies.

Combining Model-based Control with DRL. One approach to benefit from the efficiency of the model-based control methods and the robustness of DRL policies is to use a hybrid method. Works within this setting can be split into two categories; 1) Desired trajectories are generated by DRL using a reduced order model of the robot, e.g., centroidal momentum dynamics Xie et al. (2021a), 2) A residual policy adapts the trajectories from TO (Gangapurwala et al., 2020; Gangapurwala et al., 2021). Both approaches pass the generated trajectories to a whole-body controller to track the trajectories while satisfying constraints. The first category resolves the problem of exploration in DRL by using a reduced model which neglects the whole-body dynamics which is limiting for most highly dynamic locomotion tasks. The second category works well as long as the real robot behaviour remains close to the pre-generated trajectories. In cases that there exists a significant change in the environment or large external disturbances, those trajectories are not useful and the RL policy needs to learn to ignore them and find a whole new policy to learn the new behaviour. In such cases, it seems this approach is very limited.

Learning from demonstrations. An interesting approach to address this issue is to utilize demonstrations for the given task (Schaal, 1997; Ijspeert et al., 2002; Peters and Schaal, 2008). By providing to the reinforcement learning algorithm basic motions required for completing the task, we remove the need for the algorithm to find it on its own using random exploration. Here, one can combine trajectory optimization with deep reinforcement learning, by utilizing motions generated by trajectory optimization as demonstrations used to further generate robust policies using deep reinforcement learning. Demonstrations can be utilized by reinforcement learning approaches in several different ways. In order to train a reinforcement learning policy to reproduce the demonstration behavior, the majority of approaches have some notion of time in the input of the policy. Some approaches explicitly give time-indexed states from the demonstration trajectory directly as input (Peng et al., 2020; Li et al., 2021). Alternatively, some works train the policy to reproduce the demonstration behavior using only a phase variable in the policy input (Xie et al., 2020; Siekmann et al., 2021). Removing any notion of time from the input makes it difficult to train robust policies for real systems (Xie et al., 2020).

Robustness and time-dependence. There is however a crucial issue in learning control policies in such a way, in particular in the presence of environmental uncertainties. As an example, imagine we want to produce a hopping policy that can account for large uncertainties in the ground height. When contact is made at a different time than in the demonstration, the phase given as input to the policy will be different than the actual phase in the task. Instead of just going directly into a baseline hopping motion after making contact with the ground, the policy would need to force itself to get back into phase with the demonstration to have any luck to complete the task. Even removing any notion of time from the input does not on its own solve this issue. The policy would still need to get back in phase with the demonstration trajectory, but without the time-based inputs lack the information needed to be able to do so. Hence, fully removing time dependence from the demonstration trajectories in the final feedback policy is key in our approach to provide robustness with respect to contact timing uncertainties.

There are additional benefits in eliminating time dependence when training robust control policies. It allows us to more broadly randomize initial configurations of the system, something key in deploying learned policies on real robots. It also separates behavior given in the demonstration from the task goals, allowing us to improve beyond the demonstration for all the important aspects of the task at hand.

Our approach. In this work we propose a general approach that combines TO and DRL in order to produce robust policies that can be deployed on real robots. We benefit from trajectories produced by TO to bootstrap DRL algorithms and avoid exploration issues. We then use DRL to, based on these demonstrations, produce policies robust to environmental uncertainties. In this way we get the best of both worlds. Starting from TO trajectories affords solving complex tasks DRL would otherwise struggle with. The two-stage DRL approach we propose then allows us to avoid the above issues that arise when learning based on demonstrations and learn entirely time-independent robust policies.

1.1 Related work

Several recent works have used reinforcement learning to compute policies for locomotion tasks in simulation before deploying them successfully on a real quadruped (Hwangbo et al., 2019; Lee et al., 2020; Peng et al., 2020) or biped robot (Xie et al., 2020; Li et al., 2021; Siekmann et al., 2021).

In (Xie et al., 2020), similarly to our work, the authors start by learning a policy to track a reference motion and then further improve it to enable its transfer to the real system. Crucially, unlike our approach, some notion of time remains present in the policy input in all stages of training, either as a reference motion or a phase variable. The reward for tracking the original trajectory also remains present in further stages of training, preventing the policy to freely adapt away from it in order to optimize task performance. Finally, while the resulting policies show some robustness to external perturbations, there are no environment uncertainties present and no need to adapt the timing of the behavior to account for it.

In (Li et al., 2021), the authors aim to improve upon some aspects of (Xie et al., 2020). Instead of learning the policy output in the residual space, i.e. learning only the correction with respect to the demonstration trajectory, they learn the full control signal. While this makes it easier for the policy to adapt its actions away from the demonstration, the time-dependence is still present in the policy input and the issue of adaptation of timing of the demonstration remains.

In (Peng et al., 2020) the control policy is computed in a single stage of training, by learning to track the given demonstration behavior. For successful transfer to real robot they rely on domain adaptation, finding the latent encoding over the set of dynamics parameters that performs the best. There is however no randomization of the environment during training and all test are performed on flat ground. Similarly to the previous works, the notion of time remains present in the policies here as well, explicitly as a goal given to the policy containing robot states from the reference motion in several of the following time steps.

Unlike these approaches, the most successful recent learning based approach for locomotion in a challenging uncertain environment does not utilize demonstrations at all. The authors of (Lee et al., 2020) learn a robust locomotion policy for a quadruped robot that performs well on uneven and uncertain surfaces. The lack of any notion of pre-determined timing affords more room to the policy to adapt to environmental uncertainties (in this case even explicitly by outputting the frequency of the motion). While learning from scratch is possible in this case, it is true for one specific task (trotting without any flight phase) and with the structure imposed by the proposed controller. With the structure we mean that the policy only outputs stepping frequency for each

foot and this is mapped to the joint space using inverse kinematics. For highly dynamic tasks with flight phases and impacts, with no specific structure imposed for joint behavior utilizing demonstrations is still necessary. Therefore, the issue we address in this work, i.e. finding ways to build fully adaptive policies starting from demonstration behaviors, remains a challenge.

1.2 Contributions

The main contributions of this paper are as follows:

- We propose a framework to exploit the benefits of both TO and DRL to generate control policies that are robust to environmental uncertainties. A key aspect of our framework is to lose time-dependence from the initial trajectories and to build a policy that can adapt to large uncertainties in the environment.
- We evaluate the method on two highly dynamic tasks on a quadruped and show that our framework can deal with random uneven terrains as well as external disturbances. To the best of our knowledge, these results are the first demonstrating the successful use of DRL to robustly realize such behaviors.

2 Proposed algorithm

Algorithm overview. Our proposed algorithm has three main parts as shown in Figure 1. First we use TO to generate efficiently, based on a nominal model of the robot, initial trajectories for new tasks. The first stage of DRL training then proceeds to build a control policy around this trajectory, caching the solution to a light neural network. In the second stage of DRL training we replace the trajectory tracking optimization with one that directly optimizes task performance and introduce uncertainties in the training environment. This allows us to further adapt the policy from the first stage, making it robust and independent from the initial demonstration. As a final result, we get a policy that can be directly deployed on the real system without any additional training.

2.1 Trajectory optimization

Generating initial demonstration. We use trajectory optimization to generate a nominal motion for the desired task based on a nominal model of the robot and the environment. In this work, we use the trajectory optimization algorithm proposed in (Ponton et al., 2021) to compute such demonstrations. It is important to emphasize that any other trajectory optimization algorithm could also be used, as long as it
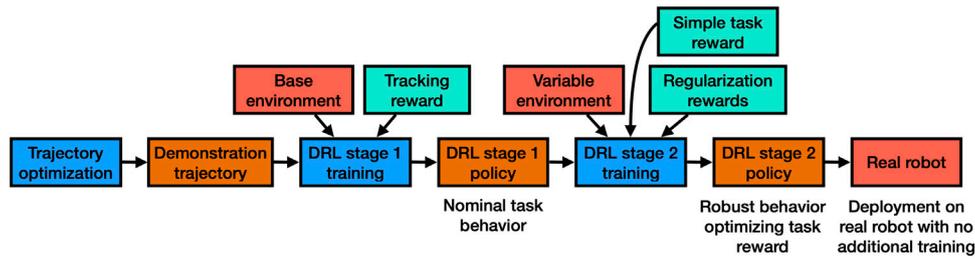
FIGURE 1
Schematic of our proposed framework. We start from a single demonstration trajectory generated by TO. In the first DRL stage we learn a policy that tracks that demonstration trajectory and successfully produces nominal behavior in simulation. To enable transfer to the real robot, the second DRL stage starts with the resulting policy from the first stage and tries to robustify the policy by randomizing contact and to optimize for performance by replacing demonstration tracking reward with task reward. We directly apply the output policy from the second stage to the robot without any domain randomization of robot parameters.

provides a set of full-body trajectories. However, the more realistic the generated motion is, the easier it is for the first stage of DRL to find a policy that tracks it. In addition, we do not utilize control actions provided by the demonstration trajectory. Only the state trajectories are required. This allows us to use any control parametrization for the policy being learned, potentially different than the one used to generate the demonstration. For instance, it is well known that having the policy output the next desired state for a fixed PD controller (rather than torque) is beneficial in terms of transfer to the real world (Hwangbo et al., 2019; Bogdanovic et al., 2020; Siekmann et al., 2021). However, finding these actions in a trajectory optimization setting is not necessarily trivial.

2.2 DRL stage 1: Learning a policy to track a given trajectory

Control policy. We use a neural network to parametrize the control policy. A relatively small network proved sufficient for the tasks we considered, with two layers of 64 units each. We keep the observation space of the policy simple, with the positions and velocities for each joint ($\mathbf{q}^{joint}$, $\dot{\mathbf{q}}^{joint}$) and only robot base variables relevant for the current task. For the hopping task, this consists of the base position and velocity along the Z-axis (z^{base}, $\dot{z}^{base}$). For the bounding task we also add the angular position and velocity around Y-axis (θ_y^{base}, $\dot{\theta}_y^{base}$). The policy outputs parameters for a PD controller in joint space. It gives desired joint positions at each step, while utilizing fixed P and D gains for control.

Throughout all the stages of training, we use an additional cost term incentivizing the policy to output values for desired joint positions that are actually tracked as well as possible. Specifically, we penalize the difference between the value given for the desired position by the policy at step t and the actual position achieved at the next step $t + 1$:

$$r_{tt} = -k_{tt} \left\| \mathbf{q}_{des}^{joint}(t) - \mathbf{q}^{joint}(t+1) \right\|^2 \qquad (1)$$

We note that while this reward term incentivizes a policy to produce a trajectory that is well tracked, it does not prevent it to give values off the trajectory to create forces during contact when necessary. This has been a crucial aspect of our previous work (Bogdanovic et al., 2020) to enable direct policy transfer to real robots without domain randomization of robot parameters.

Training procedure. We use Proximal Policy Optimization (PPO) (Schulman et al., 2017) to optimize the policies in both stages of our framework, but we do not have many requirements in the choice of the algorithm. PPO is an on-policy reinforcement learning method, working similarly to a trust-region method, but relying on a clipped objective function in order to simplify the algorithm and ensure better sample complexity. As we use the provided demonstration to resolve exploration issues, we do not need a, potentially off-policy, reinforcement learning algorithm with strong characteristics in this regard. We instead choose an on-policy algorithm with good convergence properties.

In this stage, the optimized reward consists of two parts: a part for tracking the time-based demonstration and the above-defined regularization term that is a part of the controller parametrization:

$$r_{s1} = r_{ti} + r_{tt}$$
$$r_{ti} = k_{ti1} \exp\left(-k_{ti2}\|\mathbf{x}_{demo}^{base} - \mathbf{x}^{base}\|\right) + k_{ti3} \exp\left(-k_{ti4}\|\dot{\mathbf{x}}_{demo}^{base} - \dot{\mathbf{x}}^{base}\|\right)$$
$$+ k_{ti5} \exp\left(-k_{ti6}\|\mathbf{q}_{demo}^{base} \ominus \mathbf{q}^{base}\|\right)$$
$$+ k_{ti7} \exp\left(-k_{ti8}\|\boldsymbol{\omega}_{demo}^{base} - \boldsymbol{\omega}^{base}\|\right)$$
$$+ k_{ti9} \exp\left(-k_{ti10}\|\mathbf{q}_{demo}^{joint} - \mathbf{q}^{joint}\|\right)$$
$$+ k_{ti11} \exp\left(-k_{ti12}\|\dot{\mathbf{q}}_{demo}^{joint} - \dot{\mathbf{q}}^{joint}\|\right)$$
$$r_{tt} \text{ as defined in (1),} \qquad (2)$$

where $\mathbf{x}^{base}$ is the position of the robot base, $\mathbf{q}^{base}$ the base quaternion, $\boldsymbol{\omega}^{base}$ the base angular velocity and $\mathbf{q}^{joint}$ the joint positions. $k_{ti1}, \ldots, k_{ti12}$ represent individual weight and scale

constants for each term. We mark difference between two quaternions with ⊖.

Following Peng et al. (2018), we make two further design choices that prove to be vital in making the training robustly work:

- We initialize each episode at a randomly chosen point on the demonstration trajectory.
- We terminate episodes early if the robot enters states that are not likely to be recoverable (based for example on tilt angle of the robot base) or just not conducive for learning (for example knees of the robot making contact with the ground).

Output. In the first DRL stage, we aim to produce a policy that provides some nominal behavior on the task in simulation. However, in our experiments, these policies failed to transfer to the real robot. They remain static, cause shaky behavior on the robot, or result in motions with severe impacts. We give some examples in the accompanying video. As can be seen there, the behavior is not even close to the gaits in simulation which makes any quantitative analysis unfeasible. To solve these problems, we need an additional stage of training that generates policies that can transfer to the real robot.

2.3 DRL stage 2: Generating robust time-independent policy

To create policies that are successfully transferable to the real robot, we continue training starting from the policies outputted from the first stage. As a note, we preserve the entire policy, including the parameters controlling the variance of the action distribution that the policy outputs. While the lower variance from the resulting policies from stage 1 might lower exploration capabilities in stage 2, the initial policy already performs nominal behavior on the task, so there is no need for significant exploration away from it. Additionally, we found that any attempts to artificially increase the variance prior to stage 2 result in quick loss of the behavior from stage 1 that we are attempting to carry over. On the other hand, we found no issues with potentially low resulting variance from stage 1 preventing further adaptation of the policy in stage 2.

We further introduce the following changes in the training procedure:

Initialization. We replace initialization on the demonstrated trajectories with initialization in a wider range of states. This allows us to better cover the range of states the policy might observe when deployed on the real system, allowing it to learn how to recover and continue the motion in those cases.

Environment uncertainties. We introduce uncertainties in the environment in order to produce more robust motions when deployed on the real system. In this work, we are mainly concerned with randomization of the contact surface heights and friction.

Time-independent task reward. We replace the time-based demonstration tracking reward with a time-independent, direct task reward. We have already noted the issues that can arise while trying to account for environment uncertainties while tracking a time-based demonstration trajectory. The policy is locked into trying to follow the specific time schedule regardless of the environment, whereas adapting it would produce much better recovery behavior. Switching to a time-independent reward, directly defining the task helps us deal with this.

Switching to this task reward has additional benefits. It allows us to directly optimize desirable aspects of the task, whereas the demonstration only needs to give us some nominal behavior on the task. The policy is free to change the behavior in a way that is needed to perform the task in the best possible way, without being penalized for not doing it in the same way as in the demonstration. We can also, as we will see later, produce varied behavior starting from a single demonstration by adapting this task reward.

Regularization rewards. Finally, we add additional reward terms in this stage to further regularize the behavior of the learned policies. We aim to incentivize desirable aspects of policies in the tasks, like torque smoothness and smooth contact transitions (refer to Section 2.3 for details).

3 Evaluation

3.1 Tasks setup

We evaluate our approach on two different dynamic tasks on a quadruped robot: hopping and bounding. In the hopping task, the robot has to perform continuous hops on four legs, reaching a specific height and softly landing on the ground. Bounding consists of the robot performing oscillatory behavior around the pitch angle, with two full flight phases during a single period and each time making contact on exactly two legs (front or back). In both cases we start from a basic demonstration trajectory that provides state trajectories for the current task. Starting from that we apply our two-step training procedure in simulation to produce robust policies that we then test on a real robot.

We perform experiments on the open-source torque-controlled quadruped robot, Solo8 (Grimminger et al., 2020) (see Figure 2), which is capable of very dynamic behaviors. For simulating the system we use PyBullet (Coumans and Bai, 2016).

We use exactly the same training procedure in both cases, with the only differences arising from the need to allow for base rotation around one axis in the bounding task. This is a particular benefit of the approach we present here–for a new task we only need a single new demonstration trajectory and a single simple reward term defining the task.
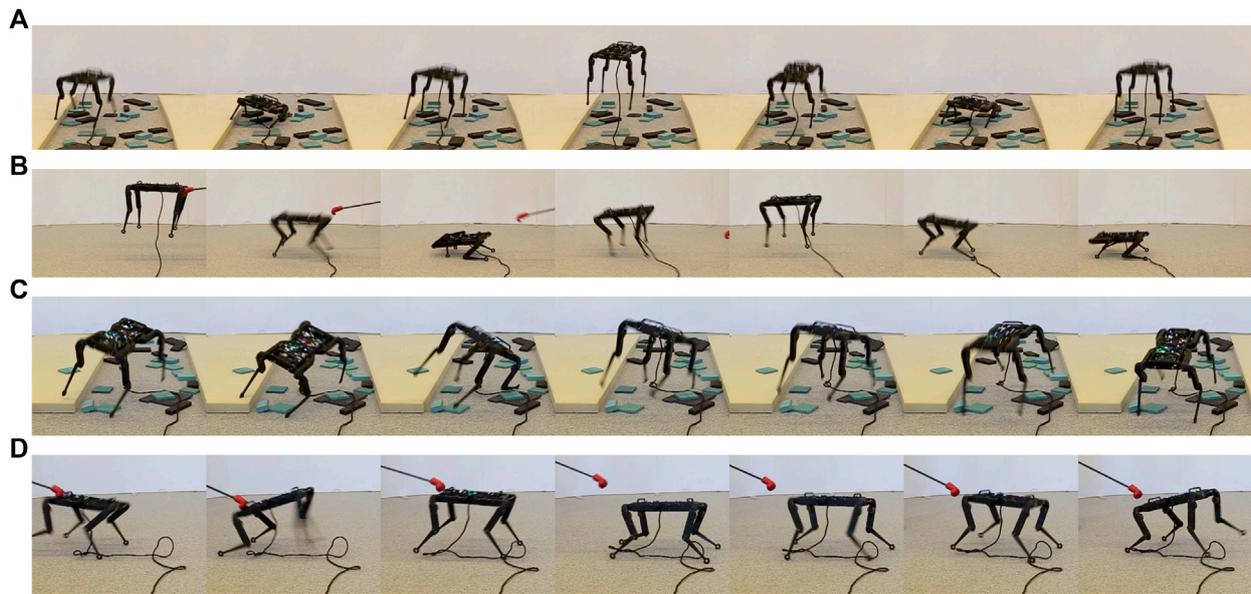
FIGURE 2
Examples of the robustness tests carried out on the quadruped robot Solo8 **(A)** Hopping on a surface comprised of a soft mattress and small blocks **(B)** Hopping with push recovery **(C)** Bounding on a surface comprised of a soft mattress and small blocks **(D)** Bounding with push recovery.

DRL stage 1: Early termination. The only aspect in the first stage of training specific to the chosen tasks is how we perform early termination. We perform early stopping here based on the current tilt of the robot base (we increase the range appropriately for the bounding task), as well as when any part of the robot that is not the foot touches the ground.

DRL stage 2: Initialization states. As noted in the method description, in the second stage of DRL training, we introduce a wider range of initialization states. In the two tasks investigated here, this consists of randomizing the initial height of the base of the robot, tilt of the base around x- and y-axis and randomness in the initial joint configuration. We preserve the early termination criteria from stage 1, only extending the range of allowed base tilt angles with the way it is increased in the initialization.

DRL stage 2: Environment uncertainties. We also introduce uncertainties in the training environment. We randomize the ground position up and down in the range of $[-5 \text{ cm}, 5 \text{ cm}]$ (approximately 20% of the robot leg length). We also randomize the ground surface friction coefficient in the range $[0.5, 1.0]$. While we restrict ourselves only to this limited set of initial state and environment randomizations, as we will see in the later evaluations, this produces policies that are quite robust as they can also handle uneven ground or external perturbations.

DRL stage 2: Reward structure (hopping). By using a demonstration trajectory to deal with exploration issues, we can define the individual task rewards to be very simple, without the need for any reward shaping.

For the hopping task we use the following reward

$$r_{s2hp} = r_{hp} + r_{ps} + r_{ct} + r_{ts} + r_{tt} \qquad (3)$$

We use the r_{hp} reward term to define the task

$$r_{hp} = \begin{cases} k_{hp} z^{base}, & \text{if } z^{base}_{min} < z^{base} < z^{base}_{max}, \\ 0, & \text{otherwise.} \end{cases} \qquad (4)$$

The reward at each timestep is proportional to the current height of the robot base (z^{base}), with constant weight k_{hp}. It is clipped to zero below a certain threshold (z^{base}_{min}), one that the robot can reach without leaving the ground. We additionally clip the value of this reward to be zero above a certain height threshold (z^{base}_{max}) to incentivize lower hops. We will additionally vary this threshold to produce policies with different hopping heights starting from the same demonstration trajectory.

We also introduce several reward terms to incentivize different desirable aspects in the resulting behavior. They reward the base to be close to its horizontal default posture (r_{ps}) and smooth contact transitions (r_{ct}) and torque smoothness (r_{ts}).

We reward the policy for being static in all the base dimensions (positions (x^{base}, y^{base}) and Euler angles (θ^{base}_x, θ^{base}_y, θ^{base}_z)) except the one the motion is performed on (z-axis in this case). With $k_{ps1}, \ldots, k_{ps10}$ being weight and scale constants.

$$\begin{aligned} r_{ps} = & k_{ps1} \exp\left(-k_{ps2}|x^{base}|^2\right) + k_{ps3} \exp\left(-k_{ps4}|y^{base}|^2\right) \\ & + k_{ps5} \exp\left(-k_{ps6}|\theta^{base}_x|^2\right) \quad + k_{ps7} \exp\left(-k_{ps8}|\theta^{base}_y|^2\right) \\ & + k_{ps9} \exp\left(-k_{ps10}|\theta^{base}_z|^2\right) \end{aligned} \qquad (5)$$
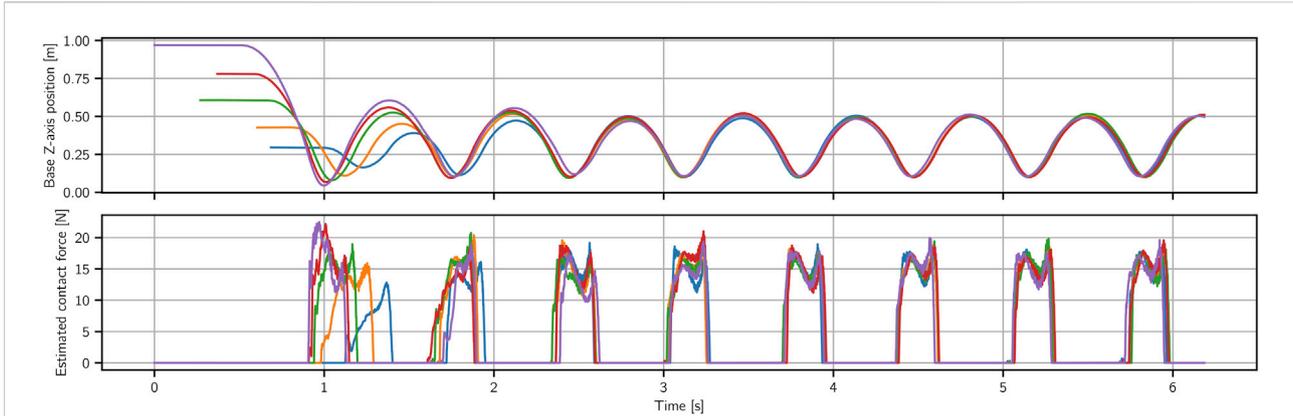
FIGURE 3
Hopping experiments started from different initial heights. The top figure shows that after dropping the robot from a set of different heights ranging between 0.3 and 1 m, the robot goes back to the nominal behavior of hopping at height of 0.5 m in one or at most two cycles. The bottom figure shows that, dropping from different initial heights, the robot is able to adapt its landing such that the impact forces remain very low and almost invisible in the estimated force.

This term is crucial as it drives the policy to stay at the default posture as much as possible. Without it the policy could perform the task well in the simulation while always being close to falling over-which would likely happen when it was transferred to the real system.

The second key reward term asks for smooth contact transition (r_{ct})

$$r_{ct} = \begin{cases} -k_{ct} \sum_{i=1}^{4} F_i^{foot}, & \text{if } \sum_{i=1}^{4} F_i^{foot} > F_{limit}^{foot}, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

We do so by simply penalizing any contact force values (F_i^{foot}) above a certain threshold (F_{limit}^{foot}), to penalize impact, with k_{ct} being a constant weight. Without this term we would have the feet hitting the surface hard on each landing-this is precisely what we observe in policies from DRL stage 1 where this reward term is not present. This is not the kind of behavior desired on the real system and these impacts can cause actual damage to the robot. These types of policies also transfer less well between simulation and real world. They can learn to perform the task well in simulation by generating hard impacts, but doing so exploits the weaknesses of the contact model in simulation, resulting in a poor performance when transferred to the real system. Smooth contact transitions enable a better transition between simulation and the real system. Thus, policies which incentivize those better transfer to the real system.

The third reward term (r_{ts}) prevents the policy to ask for a very quick change in the desired torque which is not realizable on the real robot with a limited control bandwidth

$$r_{ts} = -k_{ts1} \exp(k_{ts2} \|\boldsymbol{\tau}(t) - \boldsymbol{\tau}(t-1)\|) \quad (7)$$

with τ being the joint torque and k_{ts1} and k_{ts2} weight and scale constants respectively.

The final reward term r_{tt} is the same one we use in the first stage of training (as defined in (Eq. 1).

DRL stage 2: Reward structure (bounding). For the bounding task, we only make changes to the parts of the reward defining the task:

$$r_{s2bn} = r_{bn} + r_{cc} + r_{ps} + r_{ct} + r_{ts} + r_{tt} \quad (8)$$

We define the task reward here in two parts, r_{bn} and r_{cc}. r_{bn} rewards the policy for being close to the path the demonstration takes in the $[z^{base}, \theta_y^{base}]$ space:

$$r_{bn} = -k_{bn} \min_i \|[z^i, \theta_y^i] - [z^{base}, \theta_y^{base}]\| \quad (9)$$

where $[z^i, \theta_y^i]$ are points from the demonstration trajectory, $[z^{base}, \theta_y^{base}]$ is the current state of the base, with k_{bn} being a constant weight. We do this with no concept of time in this case, by just taking the distance to the closest point. This gives the policy freedom to perform the motion slower or faster, with different amplitude. We will later see that this results in a variety in bounding behaviors from repeated trainings, independent of the timing in the original demonstration.

The second part of the task reward, r_{cc}, is related to the contact state

$$r_{cc} = \begin{cases} k_{cc}, & \text{only front two legs in contact,} \\ k_{cc}, & \text{only back two legs in contact,} \\ k_{cc}, & \text{no legs in contact,} \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

with k_{cc} being a constant reward. It incentivizes the policy to, when making contact with the ground, only do so with front or back legs at the same time. Without anything to incentivize the policy to do this, we have observed DRL stage 1 policies reproducing the bounding motion while keeping all feet in contact with the ground. This reward part ensures appropriate contact states with flight phases in between.
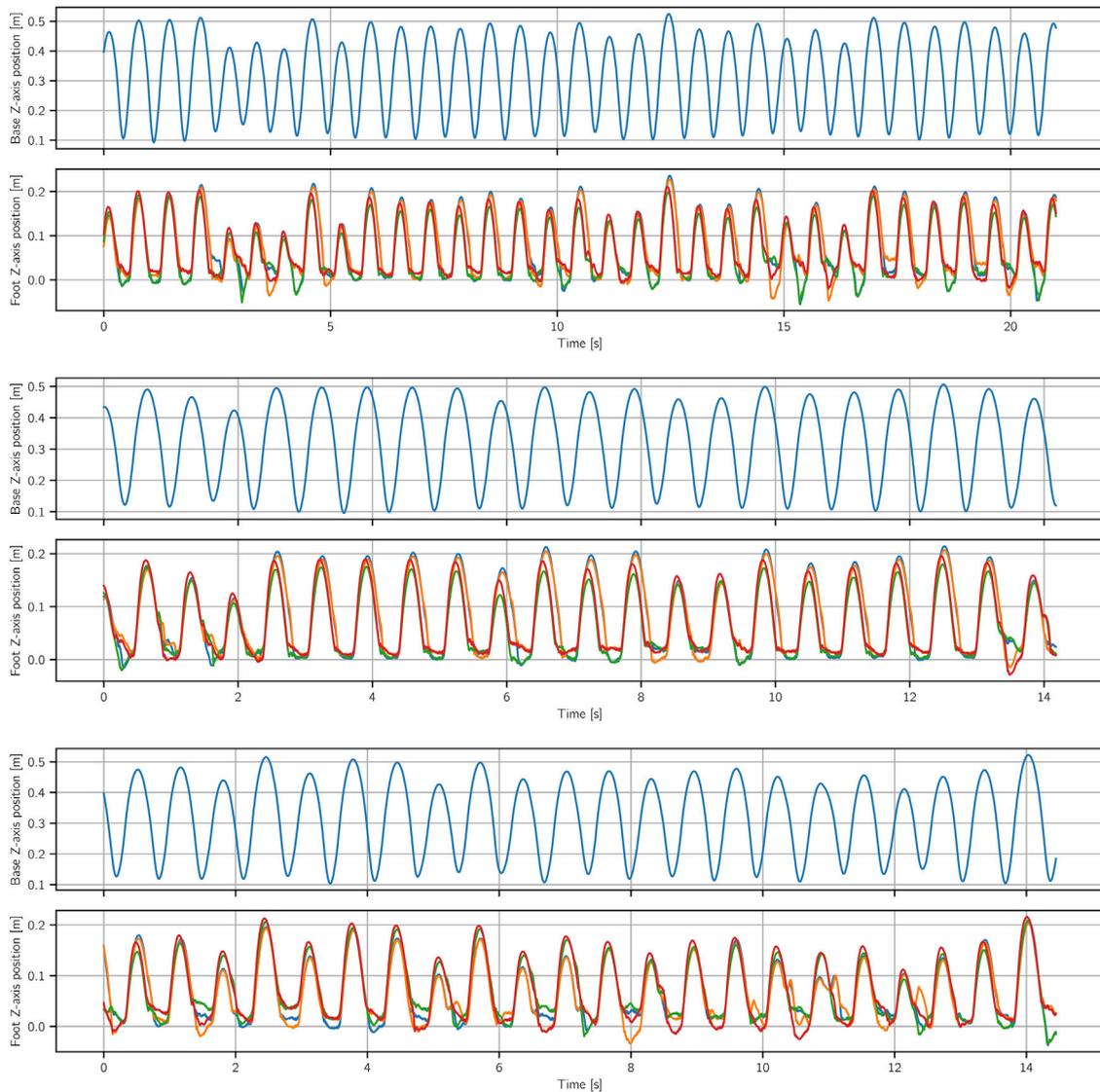
FIGURE 4
Three different execution of a jumping policy in the real world on a random uneven terrain with 5 cm height uncertainty. As we can see, different feet land on different height in each jump, but the policy manages to keep the jumping height within a certain bound.

We keep the other reward terms, ones used to incentivize desired aspects of the behavior, the same as in the hopping task (r_{tt} as defined in (Eq. 1), r_{ct}, r_{ts} as defined in (Eq. 7). The one change we make is to the reward incentivizing the robot base staying close to the default posture, r_{ps}

$$r_{ps} = k_{ps1} \exp\left(-k_{ps2}|x^{base}|^2\right) + k_{ps3} \exp\left(-k_{ps4}|y^{base}|^2\right)$$
$$+ k_{ps5} \exp\left(-k_{ps6}|\theta_x^{base}|^2\right) + k_{ps9} \exp\left(-k_{ps10}|\theta_z^{base}|^2\right)$$

$$(11)$$

We do not reward staying at default posture in the θ_y^{base} direction in this case, as that is the angle the robot is moving around while bounding.

Details of the policy structure and values of all the reward function parameters can be found in Appendix Table A1.

3.2 Hopping task results

Figure 3 shows results when the real robot is dropped from different heights to start the motion. We show the base height and estimated contact force for one of the legs as a function of time. As we do not have force sensors in the feet, we estimate the contact forces based on the torques the robot applies, using $F_i^{foot} = (S_i J_i^T)^{-1} S_i \tau$, where S_i and J_i are the joint selection
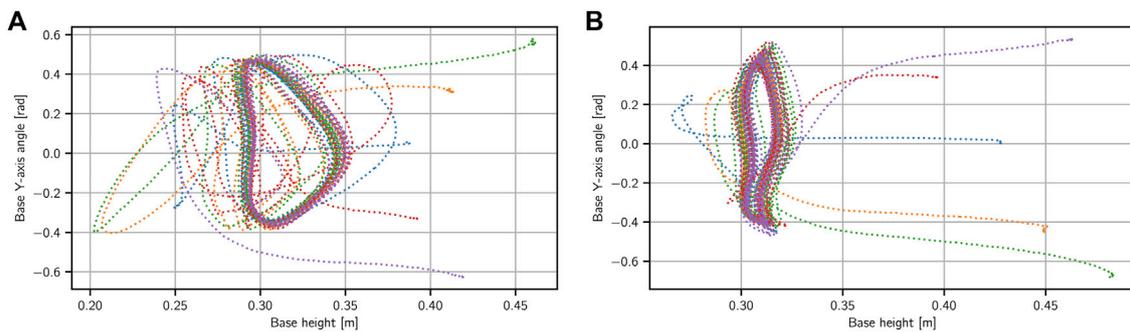
FIGURE 5
Two different bounding behaviors on the robot with different initial conditions. Starting with a wide range of initial angles for the base in *y*-direction (roughly between −35 and 35 deg), the robot quickly converges back to the desired behavior.

matrix of the leg *i* and Jacobian of the foot *i*, respectively. Note that this estimation ignores the energy dissipated through damping of the robot structure and drive system. However, it provides an approximate measure of contact forces sufficient for the analyses of this paper. We further align the plots based on the later part of the motion–the stable cycle the robot gets into.

First, we can note that regardless of the drop height the robot goes into the same stable hopping cycle. What is more, it does so very quickly, as we can see all the individual rollouts matching after only two hops. We can also note here the benefits of time independence of the policy. It is what allows us to be able to start the motion from this large range of initial heights. It is also what enables this fast stabilization, as we can see that the two initial hops are on a different cycle–one needed to stabilize the motion properly.

This test also highlights the general quality of contact interaction achieved with this approach. We can see that the impact forces, even on the highest drop (1 m height), barely go over the force values for the stable hopping cycle (50 cm height). This is purely learned behavior, as a result of impact penalties introduced in stage 2 of DRL training. It is not present in the demonstration and when we test DRL stage 1 policies on the real system high impact forces are generated and the policies are very fragile. Smooth contact transitions can also be observed in the accompanying video.

Further, the behavior is robust to uneven terrain and external pushes although this was never explicitly trained for. The robot is able to recover from significant tilt of the base arising from either external pushes or landing on an uneven surface (Figures 2A,B). More extensive examples of recovery behavior can be seen in the accompanying video.

To present quantitatively the performance of the policy on a random uneven terrain, we scattered different objects with heights ranging from 1 to 5 cm and executed a jumping policy on this surface. As it can be seen in Figure 4, while the robot feet land on the ground in different heights, the policy manages to keep the robot base stable and perform jumps close to the desired height which is 50 cm.

Finally, we demonstrate the variety of robust behaviors that can be optimized from the same demonstration by doing repeated trainings

with different values of the z_{max}^{base} threshold in the hopping task reward. With this simple change in the task reward, starting from one demonstration, we can produce hopping behaviors at different heights. Examples of this can be seen in the accompanying video.

3.3 Bounding task results

In Figure 5 we show results for a test where we drop the robot from different angles to start the motion. We perform the same test for two different final DRL stage 2 policies for this task. We can see that the policies can handle a wide range of initial base angles–around 35° in both directions. What is more, as was the case with the hopping task, we can see that here as well all the initializations end up in the same stable motion cycle.

The bounding motion exhibits similar robustness to uneven terrain and external perturbations as the hopping motion (Figures 2C,D). Same as with the hopping task, the policies rely on their knowledge of how to handle a varied set of base states to recover from anything that arises from these conditions even through it was not explicitly trained for.

In this task, we would also like to demonstrate the variety of behaviors we can generate from one single demonstration. Unlike in the hopping task, where we made simple changes in the task reward to achieve different jumping heights, we instead give more freedom to the task reward and examine the variety of produced behaviors. As noted in the task reward definition, the policy has the freedom to produce slower or faster bounds with smaller or larger amplitudes. As seen in the accompanying video we arrive at a variety of bounding behaviors in this way, starting from the same initial demonstration trajectory.

4 Discussion

Simplicity and generality of the approach. One of the main benefits of the approach is its generality and simplicity. The only

elements needed for each new task, as can be seen from the two tasks studied here, are a single demonstration trajectory and a direct straightforward task reward.

The demonstration does not need to provide ideal performance on the task, as we optimize task performance in the later stage of training. It simply needs to provide a sequence of states, and not necessarily actions, that enable a decent performance. Providing only states is also simpler in cases where it is not trivial to calculate the exact forces to realize a particular motion.

As for the task reward, reward shaping is not needed, as the demonstration resolves any exploration issues that could occur as a result of sparse reward signal. We can directly reward aspects of the task of interest. We keep reward terms other than the task reward as general as possible, encoding characteristics of general good robot behavior. We expect those to remain constant across a varied range of tasks.

Learning from scratch. In this paper, we proposed a framework to learn a policy for dynamic legged locomotion that is transferable to the real world. One might argue that for some locomotion tasks, it is possible to generate the policy without the need for the demonstration, e.g. Lee et al. (2020). However, for the two tasks examined in this paper, with given task rewards, we failed to find successful policies when training from scratch, with policies being unable to learn any notion of the task, even in simplest conditions. It is particularly difficult to learn from scratch in highly dynamic tasks with flight phase which has been the main focus of this paper. In such case, the robot crashes into the ground repeatedly, ending the episode, without providing any information for the learning algorithm on how to fix it. Additionally, the majority of recent successful approaches for doing RL in locomotion tasks use demonstrations in some way, e.g., Xie et al. (2020); Peng et al. (2020). This suggests that learning from scratch is often not viable and using demonstrations is one of the predominant solutions being utilized.

Reinforcement learning perspective. From the reinforcement learning perspective our approach presents a simple and effective way to deal with exploration issues in robotic tasks. We also remove the trajectory tracking reward in the second stage of our training, so, as seen in our experiments, we are able to change the policy away from the exact behavior defined in the demonstration.

Trajectory optimization perspective. From the trajectory optimization perspective, our approach proposes a systematic way to consider different types of uncertainty and find a robust control policy for robotic tasks, especially those with contact. Furthermore, our approach caches the solution of a model-based approach for future use and eliminates the need for re-generating repetitive motions. We believe this is a practical way to combine the strength of trajectory optimization and reinforcement learning for continuous control problems; 1) Trajectory optimization is used to generate a desired behavior efficiently to achieve the task at hand 2) different types of realistic uncertainties are easily added to the simulation, e.g. contact timing uncertainty, and DRL is used to produce a robust feedback policy.

5 Conclusion

In this work, we presented a general approach for going from trajectories optimized using TO to robust learned policies on a real robot. We showed how we can start from a single trajectory and arrive at a robust policy that can be directly deployed on a real robot, without any need for additional training. Through extensive tests on a real quadruped robot, we demonstrated significant robustness in the behaviors produced by our approach. Importantly, we do so in setups, uneven ground and external pushes, for which the robot was not explicitly trained for. All this gives hope that such approaches could be used across varied robotic tasks to simply generate robust policies to be used on real hardware, bridging the gap between trajectory optimization and reinforcement learning in such tasks.

In future work, we would like to take more advantage of model-based approaches to make our framework more efficient. We intend to study how replacing the first stage of our algorithm with a form of Behavior Cloning (Pomerleau, (1988)) to imitate a whole-body MPC policy can improve efficiency.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary Material, further inquiries can be directed to the corresponding author.

Author contributions

MB, MK, and LR designed research; MB performed numerical simulations; MK prepared the demonstrations; MB and MK performed hardware experiments; MB, MK, and LR analyzed the results and wrote the paper.

Funding

Acknowledgments

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Supplementary material

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frobt.2022. 854212/full#supplementary-material

References

Bogdanovic, M., Khadiv, M., and Righetti, L. (2020). Learning variable impedance control for contact sensitive tasks. *IEEE Robot. Autom. Lett.* 5, 6129–6136. doi:10. 1109/lra.2020.3011379

Carpentier, J., and Mansard, N. (2018). Multicontact locomotion of legged robots. *IEEE Trans. Robot.* 34, 1441–1460. doi:10.1109/tro.2018.2862902

[Dataset] Coumans, E., and Bai, Y. (2016–2020). Pybullet, a python module for physics simulation for games, robotics and machine learning. Available at: http://pybullet.org.

Drnach, L., and Zhao, Y. (2021). Robust trajectory optimization over uncertain terrain with stochastic complementarity. *IEEE Robot. Autom. Lett.* 6, 1168–1175. doi:10.1109/lra.2021.3056064

Fankhauser, P., Hutter, M., Gehring, C., Bloesch, M., Hoepflinger, M. A., and Siegwart, R. (2013). "Reinforcement learning of single legged locomotion," in 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IEEE), 188–193.

Gangapurwala, S., Geisert, M., Orsolino, R., Fallon, M., and Havoutis, I. (2021). "Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning," in 2021 IEEE International Conference on Robotics and Automation (ICRA) (IEEE), 5973.

Gangapurwala, S., Geisert, M., Orsolino, R., Fallon, M., and Havoutis, I. (2020). Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control. *arXiv preprint arXiv:2012.03094*.

Grimminger, F., Meduri, A., Khadiv, M., Viereck, J., Wüthrich, M., Naveau, M., et al. (2020). An open torque-controlled modular robot architecture for legged locomotion research. *IEEE Robot. Autom. Lett.* 5, 3650–3657. doi:10.1109/lra.2020.2976639

Hammoud, B., Khadiv, M., and Righetti, L. (2021). Impedance optimization for uncertain contact interactions through risk sensitive optimal control. *IEEE Robot. Autom. Lett.* 6, 4766–4773. doi:10.1109/lra.2021.3068951

Hornby, G. S., Takamura, S., Yokono, J., Hanagata, O., Yamamoto, T., and Fujita, M. (2000). "Evolving robust gaits with aibo," in Proceedings 2000 iCRA. millennium conference. iEEE international conference on robotics and automation. symposia proceedings (cat. no. 00CH37065) (IEEE), 3040–3045.

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., et al. (2019). Learning agile and dynamic motor skills for legged robots. *Sci. Robot.* 4, eaau5872. doi:10.1126/scirobotics.aau5872

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). *Learning attractor landscapes for learning motor primitives.* (Vancouver, Canada: Tech. rep.)

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*.

Kohl, N., and Stone, P. (2004). "Policy gradient reinforcement learning for fast quadrupedal locomotion," in IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 (IEEE), 2619–2624.

Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2020). Learning quadrupedal locomotion over challenging terrain. *Sci. Robot.* 5, eabc5986. doi:10. 1126/scirobotics.abc5986

Li, Z., Cheng, X., Peng, X. B., Abbeel, P., Levine, S., Berseth, G., et al. (2021). Reinforcement learning for robust parameterized locomotion control of bipedal robots. *arXiv preprint arXiv:2103.14295*.

Morimoto, J., and Atkeson, C. G. (2009). Nonparametric representation of an approximated poincaré map for learning biped locomotion. *Auton. Robots* 27, 131–144. doi:10.1007/s10514-009-9133-z

Morimoto, J., Nakanishi, J., Endo, G., Cheng, G., Atkeson, C. G., and Zeglin, G. (2005). "Poincare-map-based reinforcement learning for biped walking," in Proceedings of the 2005 IEEE International Conference on Robotics and Automation (IEEE), 2381.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.* 37, 1–14. doi:10.1145/3197517.3201311

Peng, X. B., Coumans, E., Zhang, T., Lee, T.-W., Tan, J., and Levine, S. (2020). Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*.

Peng, X. B., and van de Panne, M. (2017). "Learning locomotion skills using deeprl: Does the choice of action space matter?" in Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (ACM), 12.

Peters, J., and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Netw.* 21, 682–697. doi:10.1016/j.neunet.2008.02.003

Pomerleau, D. A. (1988). Alvinn: An autonomous land vehicle in a neural network. *Adv. neural Inf. Process. Syst.* 1, 305–313. doi:10.5555/89851.89891

Ponton, B., Khadiv, M., Meduri, A., and Righetti, L. (2021). Efficient multicontact pattern generation with sequential convex approximations of the centroidal dynamics. *IEEE Trans. Robot.* 1, 1661–1679. doi:10.1109/TRO.2020.3048125

Schaal, S. (1997). Learning from demonstration. *Adv. neural Inf. Process. Syst.*, 1040–1046.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017).Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Siekmann, J., Green, K., Warila, J., Fern, A., and Hurst, J. (2021). "Blind bipedal stair traversal via sim-to-real reinforcement learning," in Robotics: Science and Systems.

Tedrake, R., Zhang, T. W., and Seung, H. S. (2005). "Learning to walk in 20 minutes," in Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems (Beijing), 1939–1412.

Theodorou, E., Buchli, J., and Schaal, S. (2010). A generalized path integral control approach to reinforcement learning. *J. Mach. Learn. Res.* 11, 3137–3181. doi:10.5555/1756006.1953033

Winkler, A. W., Bellicoso, C. D., Hutter, M., and Buchli, J. (2018). Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robot. Autom. Lett.* 3, 1560–1567. doi:10.1109/lra.2018. 2798285

Xie, Z., Clary, P., Dao, J., Morais, P., Hurst, J., and Panne, M. (2020). "Learning locomotion skills for cassie: Iterative design and sim-to-real, in Conference on Robot Learning (PMLR), 317.

Xie, Z., Da, X., Babich, B., Garg, A., and van de Panne, M. (2021a). Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model. *arXiv preprint arXiv:2104.09771*.

Xie, Z., Da, X., van de Panne, M., Babich, B., and Garg, A. (2021b). "Dynamics randomization revisited: A case study for quadrupedal locomotion," in 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, (IEEE), 4955–4961.

Appendix

TABLE A1 Parameter values used in the training of the policies.

Policy network (fully-connected) parameter		Reward parameters (cont.)	
Number of layers	2	k_{hp}	0.5
Units per layer	64	k_{ps1}	1.0
Nonlinearity	tanh	k_{ps2}	0.05
Output dimension	8	k_{ps3}	1.0
Output nonlinearity	tanh	k_{ps4}	0.05
Reward parameters		k_{ps5}	1.0
k_{tt}	2.25	k_{ps6}	0.05
k_{ti1} (hopping)	0.3	k_{ps7}	1.0
k_{ti1} (bounding)	0.4	k_{ps8}	0.05
k_{ti2}	5.0	k_{ps9}	1.0
k_{ti3} (hopping)	0.1	k_{ps10}	0.05
k_{ti3} (bounding)	0.0	k_{ct}	0.2
k_{ti4}	0.2	F_{limit}^{foot}	50.0
k_{ti5} (hopping)	0.1	k_{ts}	0.02
k_{ti5} (bounding)	0.4	k_{bn}	8.0
k_{ti6}	5.0	k_{cc}	0.5
k_{ti7} (hopping)	0.1	Termination angles	
k_{ti7} (bounding)	0.0	θ_x^{limit}	0.2
k_{ti8}	0.2	θ_y^{limit} (hopping)	0.2
k_{ti9} (hopping)	0.3	θ_y^{limit} (bounding)	0.6
k_{ti9} (bounding)	0.2		
k_{ti10}	0.5		
k_{ti11} (hopping)	0.1		
k_{ti11} (bounding)	0.2		
k_{ti12}	0.05		