

Legged Robots that Keep on Learning: Fine-Tuning Locomotion Policies in the Real World

Laura Smith¹, J. Chase Kew², Xue Bin Peng¹, Sehoon Ha^{2,3}, Jie Tan², Sergey Levine^{1,2}

¹Berkeley AI Research, UC Berkeley ²Google Research ³Georgia Institute of Technology

Email: smithlaura@berkeley.edu



Fig. 1: We demonstrate real-world improvement through fine-tuning multiple skills to various real-world environments. The robot learns to walk back and forth on grass (**top left**) and side-step on carpet (**bottom left**), while recovering seamlessly from failure (**right**).

Abstract—**Legged robots are physically capable of traversing a wide range of challenging environments, but designing controllers that are sufficiently robust to handle this diversity has been a long-standing challenge in robotics. Reinforcement learning presents an appealing approach for automating the controller design process and has been able to produce remarkably robust controllers when trained in a suitable range of environments. However, it is difficult to predict all likely conditions the robot will encounter during deployment and enumerate them at training-time. What if instead of training controllers that are robust enough to handle any eventuality, we enable the robot to continually learn in any setting it finds itself in? This kind of real-world reinforcement learning poses a number of challenges, including efficiency, safety, and autonomy. To address these challenges, we propose a practical robot reinforcement learning system for fine-tuning locomotion policies in the real world. We demonstrate that a modest amount of real-world training can substantially improve performance during deployment, and this enables a real A1 quadrupedal robot to autonomously fine-tune multiple locomotion skills in a range of environments, including an outdoor lawn and a variety of indoor terrains. (Videos and code¹)**

I. INTRODUCTION

Legged robots possess a unique physical capability to traverse a wide range of environments and terrains, from subterranean rubble to snowy hills [1], [2]. However, fully realizing this capability requires controllers that can effectively handle this broad range of environments. Engineering such robust controllers for each robot is a labor-intensive process, requiring human expertise and precise modeling of the system dynamics [3]–[5]. Reinforcement learning (RL) algorithms have been used to automatically learn robotic locomotion skills in a wide range of contexts, both in simulation and in the real world [6]–[17]. However, in order

for these methods to handle the full range of environments that the robot will encounter at test-time, they must be trained in an appropriately broad range of conditions – in a sense, the RL approach exchanges the burden of controller engineering for the burden of training-time environment engineering. While much of the work on learning locomotion skills has focused on training robust skills that can generalize to a variety of test-time conditions (e.g., different terrains) [18], [19], they all share the same fundamental limitation: they lack any recourse when the test-time conditions are so different that the trained controllers fail to generalize. In this paper we are specifically interested in the case where perfect zero-shot generalization is impossible. In this case, what can the robot do? In this paper, our approach to this problem is to enable *real-world fine-tuning*: when the robot inevitably fails, it would need the mechanisms necessary to recover and *fine-tune* its skills to this new environment.

Although in principle RL provides precisely the toolkit needed for this type of adaptation, in practice this kind of fine-tuning presents a number of major challenges: the fine-tuning must be performed rapidly, under real-world conditions, without reliance on external state estimation or human assistance. The goal in this paper is to design a complete system for fine-tuning robotic locomotion policies under such real-world conditions. In our proposed framework, the robot would first attempt the desired locomotion task in some new environment, such as the park shown in Figure 1. Initially, it may fall because the uneven ground is not compatible with its learned policy. At this point, it should immediately stand back up using an agile learned reset controller, make a few more attempts at the task, and then use the collected experience to update its policy. To enable open world learning, the reward signal for RL must be obtained from the robot’s own onboard sensors, and the robot must keep attempting

¹<https://sites.google.com/berkeley.edu/fine-tuning-locomotion>

the task until it succeeds, improving with each trial. This process must be successful both where it generalizes well (and is usually successful) and where it generalizes poorly (and fails on most initial trials). Concretely, we utilize motion imitation [19] to provide a general recipe for learning agile behaviors. To ensure that the robot operates autonomously, we use a learned recovery policy that enables the robot to quickly and robustly recover from falls. This autonomous process can either fine-tune one skill at a time, or fine-tune multiple complementary skills together, such as a forward and a backward walking motion. Due to our choice of RL formulation, learning these additional policies is a simple, straightforward extension. For efficient and stable real-world training, we opt for an off-policy RL algorithm that uses randomization over ensembles to stabilize and substantially improve the sample-efficiency of Q-learning methods [20].

The main contribution of our work is a system for real-world autonomous fine-tuning of agile quadrupedal locomotion skills. To our knowledge, our system is the first to show real-world fine-tuning using RL, with automated resets and onboard state estimation, for multiple agile behaviors with an underactuated robot. In our experiments, we take advantage of simulation data to pre-train a policy, reaping the safety and efficiency benefits of training in simulation, while retaining the ability to continue learning in new environments with real-world training. Although the particular components we integrate into our real-world fine-tuning system are based on prior works, the combination of these components is unique to our system and together, they enable efficient real-world fine-tuning of agile and varied locomotion behaviors, together with highly efficient resets between trials and recoveries from falls. We demonstrate in our experiments that our system enables an A1 quadruped robot to learn dynamic skills, such as pacing forwards and backwards in an outdoor grass field, and side-stepping on 3 indoor terrains: carpet, doormat with crevices, and memory foam.

II. RELATED WORK

Robotic locomotion controllers are typically constructed via a combination of footstep planning, trajectory optimization, and model-predictive control (MPC) [21]. This has enabled a range of desirable gaits, including robust walking [5] and high-speed bounding [22]. However, such methods require characterization of the robot’s dynamics and typically a considerable amount of manual design for each robot and each behavior. RL provides an appealing approach to instead learn such skills, both in simulation [13]–[15] and in the real world [6], [7], [9], [11], [16], [17]. Due to safety considerations and the data intensive nature of RL algorithms, RL-based locomotion controllers are often trained in simulation. Various methods are used to improve transfer to the real world, such as building high fidelity simulators [11], [23], using real world data to improve the accuracy of simulators [24]–[26], and simulating diverse conditions to capture the variations a robot may encounter during real world deployment [2], [27], [28]. However, legged robots are capable to traverse such a wide variety

of terrains. It is thus difficult to anticipate all the conditions they may encounter at test-time, and even the most robust learned policies may not generalize to every such situation.

Another line of work trains adaptive policies by incorporating various domain adaptation techniques to perform few-shot adaptation [17], [19], [23], [29]–[31]. Particularly related to our work, two prior works have proposed to train locomotion policies in simulation [18], [19] that include a learned adaptation structure, which infers a latent or explicit descriptor of the environment. However, although such policies are adaptive, their ability to adapt is also limited by the variability of conditions seen at training-time — if the test-time conditions differ in ways that the designer of the simulation did not anticipate, they may likewise fail, as we illustrate in our experimental comparison in subsection V-A. Thus, in this work, we rather aim to perform consistent adaptation through fine-tuning with RL, and present a method that enables continuous improvement under any test-time condition in the real world. Julian et al. [32] uses an off-policy model-free RL approach to fine-tune a visual grasping policy to a variety of conditions that are not covered during pre-training. Our approach similarly uses off-policy model-free learning to continuously learn subject to changes in the environment, but we instead consider a variety of skills and challenges introduced by learning legged locomotion skills, such as underactuation and falling.

Several works have approached the challenge of training locomotion policies in the real world. This approach, however, has yet to scale to more complex motions due to the supervision requirements: these systems often rely on heavy instrumentation of the environment, such as motion capture systems, to provide reward supervision and resets, through engineering [33] or manual human intervention [16], [34], [35]. To make real-world training more broadly applicable (e.g., in the outdoors) we perform all state estimation on board, without any motion capture or external perception. While these prior works have demonstrated learning very conservative walking gaits on simple robots in the real world from scratch, we demonstrate learning of pacing and side-stepping, behaviors that are naturally unstable and require careful balancing on a more agile A1 robot. Thus, we found that it was crucial to use motion imitation and adopt a real-world fine-tuning approach rather than learning completely from scratch. Lastly, rather than manually resetting the robot or hand-designing a recovery controller specific to the robot, we used RL to automatically produce a reset controller, an approach that can be applied to automatically produce reset controllers for other quadrupeds as well.

III. FINE-TUNING LOCOMOTION IN THE REAL WORLD

In this section, we present our system for fine-tuning locomotion policies. It combines a stable and efficient RL algorithm with multi-task training, thereby allowing our robot to learn quickly with minimal human intervention. Sample-efficient learning is achieved by using a recently proposed off-policy RL algorithm, randomized ensembled double Q-learning (REDQ), which has demonstrated efficient learning

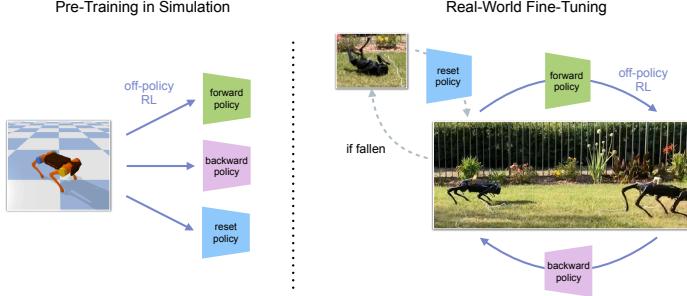


Fig. 3: Example of our system. First, we pre-train skills (in this example, forward/backward pacing and reset) in simulation using RL. We then deploy the policies in the real world. The robot executes forward or backward pacing depending on which will bring it closer to the origin. After each episode, it automatically runs its reset policy in preparation for the next. We continue to update the policies with the data collected in the real world using the same RL method to facilitate perpetual improvement.

in simulated environments [20]. To enable autonomous training in the real world without requiring human intervention, we stitch together episodes with a learned reset policy.

a) Overview: Our framework, shown in Figure 3, involves learning a set of policies, one for each desired skill. Because RL algorithms are data intensive and untrained policies can be dangerous on a physical robot, we pre-train our policies in simulation, as is typical for legged locomotion controllers [2], [11], [36]. In this phase, we independently train a policy π_i for each of the skills, including a recovery policy. Once the policies are pre-trained in simulation, we perform fine-tuning in the real world by simply continuing the training process using the same RL algorithm. Because the dynamics may be significantly different in the real world, we reset the replay buffers \mathcal{D} for each policy. After each episode, the learned recovery policy resets the robot in preparation for the next rollout, preventing time- and labor-intensive manual resets. For some skills, we use a multi-task framework, which leverages multiple skills to further facilitate autonomous learning. Algorithm 2 provides an overview of the complete training process.

b) Motion imitation: Our policies are trained to perform different skills by imitating reference motion clips using the RL framework proposed by [19]. Given a reference motion \mathcal{M} comprising a sequence of poses, a policy is trained to imitate the motion using a reward function that encourages tracking the target poses at each timestep (see subsection IV-B). This general framework allows us to learn different skills by simply swapping out the reference motion. We also learn a recovery policy within this framework by training the robot to imitate a standing pose along with a few important modifications; see subsection IV-C for details.

c) Off-policy RL: We leverage the off-policy REDQ algorithm [20], a simple extension to SAC [37] that allows for a larger ratio of gradient steps to time steps, for sample-efficient RL. REDQ utilizes an ensemble of Q -functions $Q_\theta = \{Q_{\theta^k}\}_{k=1}^{N_{\text{ensemble}}}$ that are all trained with respect to the same target value, which is computed by minimizing over a random subset of the ensemble. This avoids overestimation issues that can occur when using too many gradient steps. Our update procedure is summarized in Algorithm 1.

Algorithm 1 TRAIN: RL Subroutine

Require: Critic to actor update ratio K
Require: $\mathcal{M}_i, Q_{\theta_i}, \pi_i, \mathcal{D}_i$

```

1: // COLLECT DATA
2: Calculate the goal  $\mathbf{g}_t$  from  $\mathcal{M}_i$ .
3: Collect trajectory  $\tau$  with  $\pi_i(\mathbf{a} | \mathbf{s}, \mathbf{g})$ .
4: Store  $\tau$  in  $\mathcal{D}_i$ .

5: // PERFORM UPDATES
6: for iteration  $i = 1, 2, \dots, n_{\text{updates}}$  do
7:   Update  $Q_{\theta_i}$  by minimizing  $\mathcal{L}_{\text{critic}}^{\text{REDQ}}$ .
8:   if  $i \% K == 0$  then
9:     Update  $\pi_i$  by minimizing  $\mathcal{L}_{\text{actor}}^{\text{REDQ}}$ .
return  $\mathcal{M}_i, Q_{\theta_i}, \pi_i, \mathcal{D}_i$ 

```

Algorithm 2 Real-World Fine-Tuning with Pre-training

Require: N_{refs} reference motions $\{\mathcal{M}_i\}_{i=1}^{N_{\text{refs}}}$.

- 1: Initialize: Q-function ensembles $\{Q_{\theta_i}\}_{i=1}^{N_{\text{refs}}+1}$, policies $\{\pi_i\}_{i=1}^{N_{\text{refs}}+1}$, replay buffers $\{\mathcal{D}_i\}_{i=1}^{N_{\text{refs}}+1}$, skills \mathcal{S} .
- 2: // PRE-TRAINING IN SIMULATION
- 3: **for** skill $i = 0, 2, \dots, N_{\text{refs}}$ **do**
- 4: **repeat**
- 5: $S_i \leftarrow \text{TRAIN}(\mathcal{M}_i, Q_{\theta_i}, \pi_i, \mathcal{D}_i)$
- 6: **until** convergence
- 7: $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_i\}$
- 8: // REAL-WORLD FINE-TUNING
- 9: **repeat**
- 10: Choose a skill S_i to execute.
- 11: **if** in new environment **then**
- 12: Optionally clear \mathcal{D}_i
- 13: $S_i \leftarrow \text{TRAIN}(\mathcal{M}_i, Q_{\theta_i}, \pi_i, \mathcal{D}_i)$
- 14: **until** forever

IV. SYSTEM DESIGN

We use the A1 robot from Unitree as our robot platform and build our simulation using PyBullet [38]. For our motion imitation skills, we retarget a mocap recording of dog pacing from a public dataset [39] and an artist generated side-step motion for the A1 using inverse-kinematics (see [19]). The policies $\{\pi_i\}_{i=1}^{N_{\text{refs}}}$ and Q-functions $\{Q_{\theta_i}\}_{i=1}^{N_{\text{refs}}}$ are modeled using separate fully-connected neural networks. Updates are computed using the Adam optimizer [40] with a learning rate of 10^{-4} and a batch size of 256 transitions. All networks are constructed and trained using TensorFlow [41].

A. State and Action Spaces

The state \mathbf{s}_t contains a history of 3 timesteps for each of the following features: root orientation (read from the IMU), joint angles, and previous actions. Similar to prior motion imitation approaches [14], [19], the policy receives not only proprioceptive input but a goal \mathbf{g}_t , which comprises the target poses (root position, root rotation, and joint angles) calculated from the reference motion for future timesteps. In our experiments, we use 4 future target poses, the latest of which is a target for approximately 1 second ahead of the current timestep. Actions \mathbf{a}_t are PD position targets for each of the 12 joints and applied at a frequency of 33Hz. To ensure

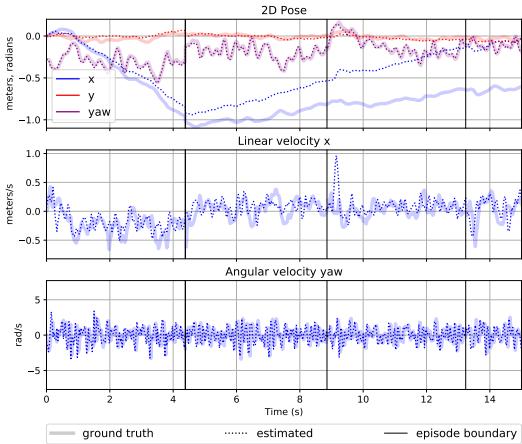


Fig. 4: Real-world state estimation compared to motion capture for a robot walking indoors. Yaw and yaw velocity are very accurate, linear velocity is acceptable, and x- and y-position drift over the course of multiple episodes. smoothness of the motions, we process the PD targets with a low-pass filter before supplying them to the robot.

B. Reward Function

We adopt the reward function from [19], where the reward r_t at each timestep is calculated according to:

$$r_t = w^p r_t^p + w^v r_t^v + w^e r_t^e + w^{rp} r_t^{rp} + w^{rv} r_t^{rv} \quad (1)$$

$$w^p = 0.5, w^v = 0.05, w^e = 0.2, w^{rp} = 0.15, w^{rv} = 0.1$$

The pose reward r_t^p encourages the robot to match its joint rotations with those of the reference motion. Below, \hat{q}_t^j represents the local rotation of joint j from the reference motion at time t , and q_t^j represents the robot's joint,

$$r_t^p = \exp \left[-5 \sum_j \|\hat{q}_t^j - q_t^j\|^2 \right]. \quad (2)$$

r_t^v and r_t^e assume a similar form but encourage matching the joint velocities and end-effector positions, respectively. Finally, the root pose reward r_t^{rp} and root velocity reward r_t^{rv} encourage the robot to track the reference root motion. See [19] for a detailed description of the reward function.

To estimate the linear root velocity during real-world training, we use a Kalman filter that takes acceleration and orientation readings from the IMU, then corrects them with the foot contact sensors. When a sensor is triggered, we take that foot as a point of 0 velocity, calculate the root velocity using the leg joint velocities, and correct the estimate from the IMU. We integrate the linear velocity for a rough approximation of the robot's position. Some example data is shown in Figure 4. We find that the angular velocity and orientation readings are very accurate, linear velocity is reasonable, and position drifts but is good enough within each episode for our reward calculations.

C. Reset Controller

Similar to [42], the reset policy is trained in simulation by generating a diverse set of initial states. At the start of each episode, we drop the robot from a random height and

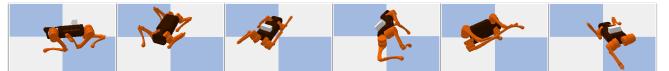


Fig. 5: Examples from the initial state distribution used to train the reset policy in simulation. To get these states, we drop the robot from about a half meter above the ground with a random initial orientation of the robot's torso. Specifically, its roll, pitch and yaw are drawn from uniform distributions over the intervals $[-\frac{3\pi}{4}, \frac{3\pi}{4}]$, $[-\frac{\pi}{4}, \frac{\pi}{4}]$, and $[-\pi, \pi]$, respectively.

orientation (see Figure 5). The robot's objective then is to recover back to a default standing pose. [42] stitches together two policies for reset, first, a self-righting behavior which puts the robot in a stable sitting position, followed by a stand-up behavior which then puts the robot in a standing pose.

We find that we are able to train a single, streamlined reset policy by modifying the motion imitation objective. Rather than using a reference motion to prescribe exactly how the robot should stand up, we modify our standard imitation reward as follows. First, the policy is only rewarded for rolling right side up. If the robot is upright, we add the motion imitation reward, where the reference is a standing pose, to encourage the robot to stand. Note that although the objective for this reset policy is simple, the behaviors that it acquires are quite complex and agile. This policy is significantly more versatile than hand-designed recoveries used in prior work [33] or company-provided reset motions that take more than 10 seconds — it can recover quickly from a fall by rolling and jumping upright, and if the robot is already upright, it quickly stabilizes it for the next trial. We encourage the reader to view a video of the agile reset controller on the project website. In our work, we found that the reset policy transfers successfully to all our test terrains, so we perform no fine-tuning.

V. EXPERIMENTS

We aim to answer the following through our experiments:

- (1) How does our finetuning-based method compare to prior approaches that utilize simulated training, including those that perform real-world adaptation?
- (2) What effects do our design decisions have on the feasibility of real-world training?
- (3) How much can autonomous, online fine-tuning improve robotic skills in a range of real-world settings?

A. Simulation Experiments

To compare our approach to prior methods, we evaluate on a simulated transfer scenario, where the policy is first trained in one simulated environment, and then “deployed” to another simulated setting, which is meant to be representative of the kind of domain shifts we would expect in real-world deployment. Performing this comparison in simulation allows us to abstract away other parts of the adaptation process, such as resets, since prior methods generally do not handle this, and allows us to provide highly controlled low-variance comparisons for each method.

a) *Comparison to prior work:* To address (1), we compare our fine-tuning method to prior work, adapting a learned forward pacing gait to several test environments. To

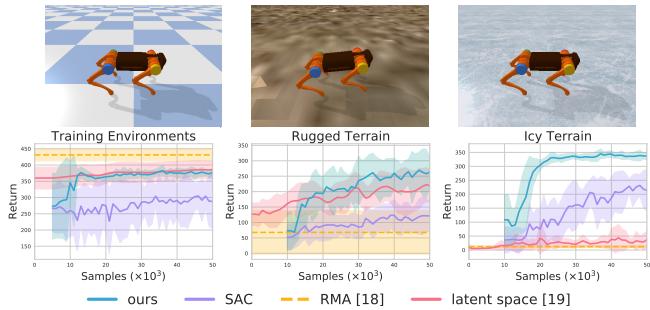


Fig. 6: We report each adapted policy’s performance (mean and standard deviation across 10 trials) in the target domain (pictured in the top row) with respect to the samples used for adaptation. Prior methods that learn adaptation strategies during training (pink and yellow) excel in environments that are similar to those seen during training (left) but fail in environments that are sufficiently different (middle, right). In contrast, our fine-tuning method (blue) continues to improve under all circumstances. We also note that our use of REDQ (blue) improves over SAC (purple).

this end, we pre-train all methods with standard dynamics randomization (varying mass, inertia, motor strength, friction, latency) on flat ground until convergence, and then deploy them on various terrains for adaptation. The test terrains include a flat ground terrain that is similar to the environments seen during pre-training, as well as two terrains that differ substantially from the training settings: randomized heightfield simulating rugged terrain, and a low-friction surface simulating a slippery, icy terrain (with a friction coefficient far below that seen during training). Note that we intentionally select these test settings to be different from the training environment: while prior works generally carefully design the training environments to cover the range of settings seen at test-time, we intentionally want to evaluate the methods under *unexpected* conditions. For each type of environment, we use 10 instantiations of the environment with different dynamics parameters, and test each method’s ability to adapt to them. The dynamics of the environment is fixed for each adaptation trial, and the same 10 environments are used for all methods.

Peng et al. [19] uses dynamics randomization to learn a latent representation of behaviors that are effective for different settings of the dynamics parameters. During adaptation, this method searches in the latent space for a behavior that maximizes the agent’s test-time performance using AWR [43]. Rapid Motor Adaptation (RMA) [18] also uses dynamics randomization to learn a latent representation of different strategies. But instead of searching in latent space during adaptation, RMA trains an additional ‘adaptation module,’ similarly to Yu et al. [30], [31], to predict the appropriate latent encoding given a recent history of observations and actions. Since RMA corresponds to a policy with memory and does not actually use data collected from a new domain to update the parameters of the model, it does not improve with additional data. We implement both methods using SAC as the policy optimizer at training-time, to match our method.

b) REDQ vs. SAC: One of our design decisions is to use the recently-proposed REDQ algorithm for policy optimization. To evaluate the importance of this choice, we consider an ablation of our method that uses vanilla SAC

instead of REDQ for both pre-training and fine-tuning. For REDQ, we use 10 Q-functions, and randomly sample 2 when computing the target value. For both fine-tuning methods, we collect an initial buffer of samples before starting to fine-tune (hence these curves start at 5000 or 10000 samples).

We report the adaptation performance of each method in Figure 6. When tested on the training environments, we see that both RMA and the latent space method perform well. This indicates that these methods indeed excel in regimes where the environments seen at training-time resemble the ones that the method must adapt to at test-time. However, when these policies, which are trained on flat ground, are placed on uneven or extremely slippery terrain, they exhibit a significant drop in performance. Both methods suffer because they assume that a pre-trained encoder or latent space can generalize, which is too strong an assumption when the test environment differs sufficiently from the training environments. RMA especially suffers in this case because it relies entirely on the pre-trained encoder to adapt, and when this encoder fails to generalize, it does not have any other recourse. The latent space method does adapt, but it relies on the latent space already containing suitable strategies for the new environment and ends up with a suboptimal policy. In contrast, our finetuning approach is able to continuously improve and eventually succeed. These results show that pretrained models, even prior adaptive models, can fail if tested on environments that deviate too much from those seen in training. Our method initially fails also, but is able to recover good performance through fine-tuning.

B. Real-World Experiments

Our real-world experiments aim to evaluate how much autonomous online fine-tuning can improve a variety of robotic skills in a range of realistic settings. We evaluate our fine-tuning system in four real-world domains: an outdoor grassy lawn, a carpeted room, a doormat with crevices, and memory foam (see Figure 8), each of which presents unique challenges. The outdoor lawn presents a slippery surface, where the feet can either slip on the grass or get stuck in the dirt. In this domain, we finetune a pacing gait in which both legs on one side of the body swing forward in unison. For the indoors experiments, the robot is tasked with performing a side-stepping motion on the various surfaces mentioned above. The carpeted room, in contrast to the grass, is high-friction, causing the robot’s soft rubber feet to deform in a manner inconsistent with simulation. The doormat presents a textured surface for the feet to get stuck. The 4cm-thick memory foam mattress is especially difficult, because the feet can sink into the mattress and the gaits need to change substantially to achieve ground clearance. For all experiments, we pretrain the policy on flat ground in simulation, run it for 5000 samples to initialize the buffer, and then finetune the policy in the real world.

We report the average return of the policies during training in Figure 7 with respect to the number of real-world samples collected. In all environments, our framework leads to substantial performance improvement with a modest amount

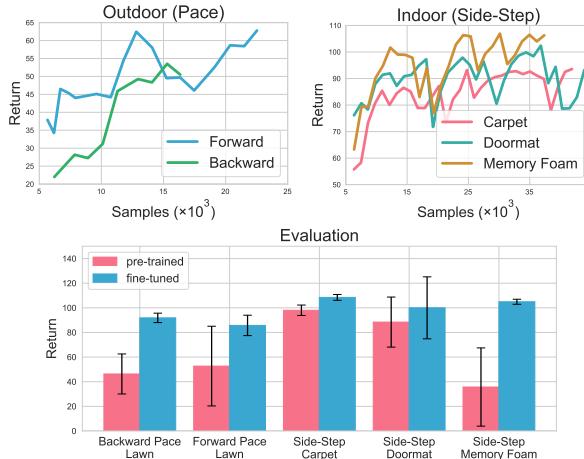


Fig. 7: **Top:** Learning curves for all the real-world fine-tuning experiments showing the average return of data collected by a stochastic policy during each iteration of training. **Bottom:** We evaluate each policy with a deterministic policy before and after fine-tuning and report mean and standard deviation over ten trials. In all domains, fine-tuning leads to improvement, and the improvement is particularly pronounced on the most difficult surfaces, such as the lawn and memory foam mattress. See the project website for videos of each of these policies throughout training.

of data. On the lawn, the pre-trained forward pacing policy makes very little forward progress, whereas the pre-trained backward pacing policy tends to trip and fall. After less than 2 hours in total of operation, the robot learns to consistently and stably pace forward and backward with very few failures. Indoors, the pre-trained sidestepping policy tends to twitch violently and fall to the floor before completing its motion. This is true across all the terrains: carpet, memory foam, and doormat with crevices. But on each terrain, in less than 2.5 hours of training, the robot learns to consistently execute the skill without stumbling. This figure includes overhead such as swapping batteries and handling robot malfunctions. Figure 8 shows a comparison of the behaviors of the policy before and after fine-tuning. For all experiments, we include videos of the training process as well as evaluation of the performance before and after training on the project website.

a) Semi-autonomous training: In our experiments, we find that our learned recovery controller is very effective in providing efficient, automatic resets in between trials. Over all experiments, the recovery policy was 100% successful. We compare our reset controller to the built-in rollover controller from Unitree. On hard surfaces, both controllers are effective while the built-in one is substantially slower than the learned policy. On the memory foam, the built-in controller performs less reliably. Please see the supplementary video or project website for videos of the reset policy in all environments and a comparison to the built-in controller. When the robot occasionally drifts outside the workspace area, it is re-positioned by a human supervisor. The need for this intervention, though, was greatly reduced by the use of the simultaneous learning of complementary skills.

VI. CONCLUSION

We present a system that enables legged robots to fine-tune locomotion policies in real-world settings, such as

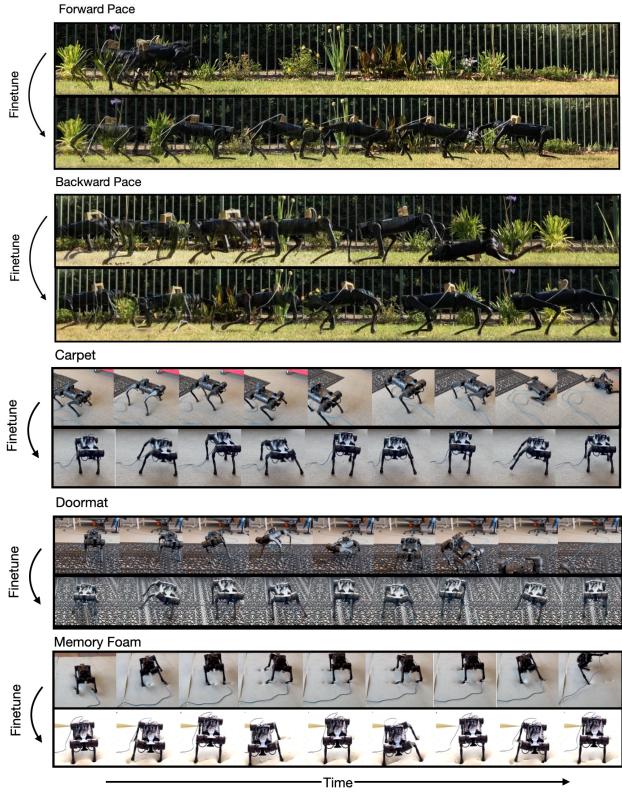


Fig. 8: **1st and 2nd Rows:** Example rollouts of the pacing policies before and after fine-tuning. The figure depicts a timelapse of a rollout of each policy, where the opacity of the robot indicates time progression (i.e., more opaque corresponds to more recent). Before fine-tuning, the forward policy struggles to make progress, while the pre-trained backwards policy makes fast progress but is quite unstable. After fine-tuning, both policies make forward progress without falling. **3rd to 5th Rows:** Example rollouts of the side-step policies before and after fine-tuning. On all terrains, the pre-trained policies fail to complete the task without falling. After fine-tuning, the policies are fine-tuned to successfully and reliably execute the skill on all three domains.

grass, carpets, doormats and mattresses, via a combination of autonomous data collection and data-efficient model-free RL. Our system provides for automated recoveries from falls, reward calculation through state estimation using onboard sensors, and data-efficient fine-tuning of a variety of locomotion skills. The fine-tuning improves the performance substantially, reaching a high level of proficiency even when starting with a gait that frequently stumbles and falls. In this work, we focus on fine-tuning in each environment separately. It would be interesting to adapt our system into a lifelong learning process, where a robot never stops learning. When the robot encounters a new environment, the fine-tuning will quickly adapt its policy. When the robot stays in the same environment for an extended period of time, the fine-tuning will gradually perfect its skills. We plan to test such a lifelong learning system for legged robots in complex, diverse and ever-changing real-world environments.

Acknowledgements

This work was supported by ARL DCIST CRA W911NF-17-2-0181, the Office of Naval Research, and Google. Laura Smith is supported by NSF Graduate Research Fellowship.

REFERENCES

- [1] I. D. Miller, A. Cohen, A. Kulkarni, J. D. Laney, C. J. Taylor, V. Kumar, F. C. Ojeda, A. Cowley, S. S. Shivakumar, E. S. Lee, L. Jarin-Lipschitz, A. Bhat, N. Rodrigues, and A. Zhou, "Mine tunnel exploration using multiple quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 5, pp. 2840–2847, 2020.
- [2] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, 2020.
- [3] G. Bledt, M. J. Powell, B. Katz, J. Carlo, P. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 2245–2252, 2018.
- [4] C. Gehring, S. Coros, M. Hutter, D. Bellicoso, H. Heijnen, R. Diethelm, M. Bloesch, P. Fankhauser, J. Hwangbo, M. Höpflinger, and R. Siegwart, "Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot," *IEEE Robotics and Automation Magazine*, vol. 23, pp. 34–43, 2016.
- [5] M. Hutter, C. Gehring, D. Jud, A. Lauber, D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Höpflinger, "Anymal - a highly mobile and dynamic quadrupedal robot," *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 38–44, 2016.
- [6] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, vol. 3, pp. 2619–2624 Vol.3, 2004.
- [7] R. Tedrake, T. Zhang, and H. Seung, "Stochastic policy gradient reinforcement learning on a simple 3d biped," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2849–2854 vol.3, 2004.
- [8] R. Tedrake and H. Seung, "Learning to walk in 20 minutes," 2005.
- [9] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning cpg sensory feedback with policy gradient for biped locomotion for a full-body humanoid," in *AAAI*, 2005.
- [10] N. Heess, T. Dhruva, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *ArXiv*, vol. abs/1707.02286, 2017.
- [11] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *ArXiv*, vol. abs/1804.10332, 2018.
- [12] Z. Xie, G. Berseeth, P. Clary, J. Hurst, and M. V. D. Panne, "Feedback control for cassie with deep reinforcement learning," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1241–1246, 2018.
- [13] L. Liu and J. Hodgins, "Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 37, pp. 1 – 14, 2018.
- [14] X. Peng, P. Abbeel, S. Levine, and M. V. D. Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Trans. Graph.*, vol. 37, pp. 143:1–143:14, 2018.
- [15] S. Lee, M. Park, K. Lee, and J. Lee, "Scalable muscle-actuated human simulation and control," *ACM Transactions on Graphics (TOG)*, vol. 38, pp. 1 – 13, 2019.
- [16] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *Robotics: Science and Systems (RSS)*, 2020.
- [17] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, 2019.
- [18] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," *Robotics: Science and Systems (RSS)*, 2021.
- [19] X. Peng, E. Coumans, T. Zhang, T. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *Robotics: Science and Systems (RSS)*, vol. abs/2004.00784, 2020.
- [20] X. Chen, C. Wang, Z. Zhou, and K. Ross, "Randomized ensemble double q-learning: Learning fast without a model," *ArXiv*, vol. abs/2101.05982, 2021.
- [21] B. Katz, J. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadrupedal control," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6295–6301, 2019.
- [22] H.-W. Park, P. M. Wensing, and S. Kim, "High-speed bounding with the mit cheetah 2: Control design and experiments," *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 167–192, 2017. [Online]. Available: <https://doi.org/10.1177/0278364917694244>
- [23] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. V. D. Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conference on Robot Learning (CoRL)*, 2019.
- [24] J. Tan, Z. Xie, B. Boots, and C. Liu, "Simulation-based design of dynamic controllers for humanoid balancing," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2729–2736, 2016.
- [25] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. D. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, 2019.
- [26] Y. Du, O. Watkins, T. Darrell, P. Abbeel, and D. Pathak, "Auto-tuned sim-to-real transfer," *ArXiv*, vol. abs/2104.07662, 2021.
- [27] X. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, 2018.
- [28] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.
- [29] Z. He, R. C. Julian, E. Heiden, H. Zhang, S. Schaal, J. J. Lim, G. Sukhatme, and K. Hausman, "Zero-shot skill composition and simulation-to-real transfer by learning task representations," *ArXiv*, vol. abs/1810.02422, 2018.
- [30] W. Yu, V. Kumar, G. Turk, and C. Liu, "Sim-to-real transfer for biped locomotion," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3503–3510, 2019.
- [31] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *IEEE Robotics and Automation Letters*, vol. 5, pp. 2950–2957, 2020.
- [32] R. C. Julian, B. Swanson, G. Sukhatme, S. Levine, C. Finn, and K. Hausman, "Efficient adaptation for end-to-end vision-based robotic manipulation," *ArXiv*, vol. abs/2004.10190, 2020.
- [33] S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan, "Learning to walk in the real world with minimal human effort," *ArXiv*, vol. abs/2002.08550, 2020.
- [34] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," *Conference on Robot Learning (CoRL)*, vol. abs/1907.03613, 2019.
- [35] S. Choi and J. Kim, "Trajectory-based probabilistic policy gradient for learning locomotion behaviors," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 1–7, 2019.
- [36] J. Siekmann, K. R. Green, J. Warila, A. Fern, and J. Hurst, "Blind bipedal stair traversal via sim-to-real reinforcement learning," *ArXiv*, vol. abs/2105.08328, 2021.
- [37] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.
- [38] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," <http://pybullet.org>, 2016–2021.
- [39] H. Zhang, S. Starke, T. Komura, and J. Saito, "Mode-adaptive neural networks for quadruped motion control," *ACM Transactions on Graphics (TOG)*, vol. 37, pp. 1 – 11, 2018.
- [40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.
- [41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [42] J. Lee, J. Hwangbo, and M. Hutter, "Robust recovery controller for a quadrupedal robot using deep reinforcement learning," *ArXiv*, vol. abs/1901.07517, 2019.
- [43] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," *arXiv preprint arXiv:1910.00177*, 2019.