Multi-Agent Manipulation via Locomotion using Hierarchical Sim2Real

Ofir Nachum
Google AI
ofirnachum@google.com

Michael Ahn
Google AI
michaelahn@google.com

Hugo Ponte*
Google AI
hugo.p.cmu@gmail.com

Shixiang (Shane) Gu
Google AI
shanegu@google.com

Vikash Kumar
Google AI
vikashplus@gmail.com

Abstract: Manipulation and locomotion are closely related problems that are often studied in isolation. In this work, we study the problem of coordinating multiple mobile agents to exhibit manipulation behaviors using a reinforcement learning (RL) approach. Our method hinges on the use of *hierarchical sim2real* – a simulated environment is used to learn low-level goal-reaching skills, which are then used as the action space for a high-level RL controller, also trained in simulation. The full hierarchical policy is then transferred to the real world in a zero-shot fashion. The application of domain randomization during training enables the learned behaviors to generalize to real-world settings, while the use of hierarchy provides a modular paradigm for learning and transferring increasingly complex behaviors. We evaluate our method on a number of real-world tasks, including coordinated object manipulation in a multi-agent setting.[1]

Keywords: real-world, multi-agent, hierarchy

1 Introduction

The coordinated effort of a group allows for far greater results than any individual could ever hope to achieve; e.g., a colony of ants can work together to build intricate underground structures. This is one of the primary reasons behind species evolving to live in groups [1]. The behaviors exhibited by coordinating agents are often far too complex to be classified as either primarily manipulation or primarily locomotion. For example, consider a group of lions attacking a prey zebra. Are they navigating to the vulnerable spots of the zebra? Or are they manipulating the zebra for a kill? When we walk – are we navigating forward in the world, or are we manipulating the Earth via our feet?

In this work, we aim to enable legged robots to exhibit these *manipulation via locomotion* behaviors by use of model-free reinforcement learning (RL). In contrast to traditional approaches relying on manual hand-design or detailed understanding of system dynamics, the use of model-free RL has the potential to provide more robust policies, as it requires little domain knowledge other than access to a reward signal. This makes it especially desirable in domains where dynamics are difficult to model, such as diverse environments (e.g., an indoor office terrain vs. an outdoor natural terrain) [2] and in the presence of complex or faulty hardware. While recent works have made great progress in applying RL to learn robust locomotion policies on real-world legged robots [3, 4, 5], examples of using RL on legged robots for coordination or manipulation remain elusive.
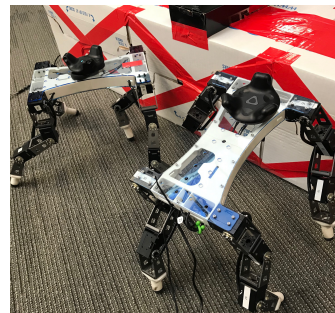

Figure 1: Multiple agents coordinate to move a large object.

---

*Work done as a member of the Google AI Residency program (g.co/airesidency).
[1]See videos at https://sites.google.com/view/manipulation-via-locomotion.
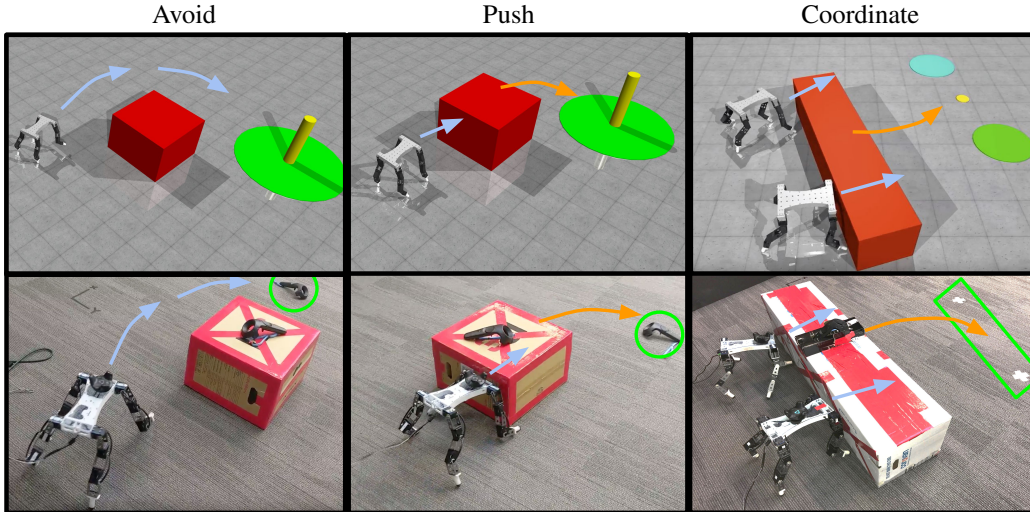
Figure 2: We consider three quadrupedal locomotion tasks of increasing complexity, utilizing the **D'Kitty** robot (see Section 4.1 for details on this robot). From left to right, we present the simulated (top row, using MuJoCo [13]) and real-world (bottom row) versions of the three tasks: *Avoid*, in which the quadruped must walk to a target location while avoiding a block object; *Push*, in which a quadruped must push a block object to a desired location; and *Coordinate*, in which two quadrupeds coordinate to push a long block to a target location and orientation. We utilize HTC Vive controllers and trackers to track the real-world position and orientation of agents, objects, and (for *Avoid* and *Push*) the desired target locations.

In part, this is due to the increased complexity of agent-object and agent-agent interactions, which exacerbates the reliance of modern RL algorithms on large amounts of real-world trial-and-error experience. While recent works have proposed to bypass the experience bottleneck by training policies in simulation and accounting for unknown differences between real-world and simulated dynamics (*sim2real*) via domain randomization [6, 7, 8, 9], domain randomization has its own scalability challenges. As a robotic task becomes more complex, its simulated counterpart must utilize more randomizations; e.g., a walking task may only need to randomize properties associated with the robot joint dynamics, but if objects are introduced to the scene, the object properties must be randomized as well. In addition to the scalibility issues associated with appropriately tuning these randomizations [10], if too many randomizations are used, the optimal policy found in simulation may be too conservative [11], to the point of being unable to adequately solve either the simulated or the original real-world task.

We propose to tackle these scalability challenges and enable legged robots (specifically in this paper, quadrupeds) to learn coordination and manipulation behaviors by introducing hierarchy. Simulation results have shown that introducing hierarchy can elevate the use of RL from simple locomotion to complex navigation and object manipulation [12]. Thus, we propose doing the same for real-world settings. We separate policies into low-level *goal-conditioned* policies that are trained to perform locomotive behavior, and high-level *goal-proposing* policies that are trained to solve a task by iteratively directing the locomotion of the low-level policy. To properly scale the use of domain randomization, we propose to train each level of the hierarchy successively rather than concurrently. That is, the low-level policy is first trained with domain randomization chosen to be appropriate for its own goal-reaching task; the low-level is then fixed and the high-level policy is trained with randomization chosen to be appropriate for its task. The modularity introduced by hierarchy is thus key to scaling up domain randomization, since each level of the hierarchy need only be trained with randomizations appropriate for its level of abstraction, and these are often much fewer (and thus less aggressive) than the randomizations necessary when training a single policy from scratch. We call our training paradigm *hierarchical sim2real*.

Hierarchical sim2real lends itself well to legged robot tasks, in which it is natural to first learn a low-level policy to perform basic locomotion. Given a well-performing locomotive policy, a high-level policy can then perform much more complex tasks, effectively treating the robot as a simple point mass. Accordingly, we apply hierarchical sim2real to a number of challenging real-world quadrupedal tasks, going beyond simple locomotion to demonstrate zero-shot transfer of learned

2

policies for object avoidance and targeted object pushing. Ablation studies provide insight into our method and show the necessity of using both hierarchy and domain randomization in order to achieve good and efficient performance. Finally, our results culminate in the successful demonstration of a challenging multi-agent task, in which two quadrupeds must coordinate their movement to push a heavy block to a desired target location and orientation. This way, we are able to show the use of locomotion to perform coordinated multi-agent manipulation.

2 Related Work

Before elaborating on the specifics of our method, we review previous works looking at manipulation via locomotion, hierarchy, and sim2real.

Multi-Agent Manipulation via Locomotion   The dual relationship between manipulation and locomotion [14, 15] is evident in several previous works using legged robots for object manipulation [16]. Much work has continued to develop such behaviors in the presence of multiple agents; see [17] for a comprehensive review on such *multi-robot systems*. For example, [18] assess the ability of two robots to coordinate their behavior to transport an elongated box, similar to the *Coordinate* task we consider. While several later works aimed at similar real-world coordination problems use RL to some extent [19, 20, 21], we believe our work is the first to use RL for the manipulation via locomotion problem as a whole. Our policy operates directly on the joint motors of the robot and is imbued with minimal domain knowledge, whereas previous works often restrict the use of RL to only high-level planning.

Hierarchy   Hierarchy is key to the success of our method. Our hierarchical decomposition of quadrupedal tasks is natural, and is inspired by similar decompositions in the robotics literature. Many previous successful examples of quadrupedal or humanoid locomotion rely on a hierarchical decomposition of the problem, in which a high-level controller optimizes for path or task space trajectories while a low-level controller optimizes joint movement and foot placement [22, 23, 24, 25]. These previous works optimize each controller in the hierarchy using methods such as inverse kinematics [22], inverse dynamics [26], or planning [23]. In contrast, the focus of our method is on the use of model-free RL to optimize policies, which is potentially more scalable as it requires relatively little knowledge of the environment dynamics. While hierarchical model-free RL has enjoyed much recent success in simulation [12], fewer examples exist of real-world successes. One example is given by [27], in which hierarchical RL is applied to a real-world path following task. Unlike our method, this previous work uses a latent space hierarchy (as opposed to a goal-conditioned hierarchy) and trains in an end-to-end manner; thus, they are unable to take advantage of modularity with respect to domain randomization in training. Moreover, their real-world demonstrations focus on path following, whereas our tasks demonstrate much more complex learned behaviors.

Sim2Real   In simulated environments, model-free RL approaches to quadrupedal locomotion have long surpassed their real-world equivalents [28] thanks to the comparatively low cost of running large amounts of simulations [13]. In addition to the large amounts of trial-and-error necessary for training, deploying RL in the real world also requires a robust mechanism for measuring reward, which itself can be an expensive, manual, and poorly scalable procedure [29]. For this reason, we avoid fine-tuning in the real world [30, 31], train wholly in simulation, and perform a zero-shot transfer to the real world. To account for unknown differences between simulation and the real world we employ domain randomization, which has been used extensively to perform successful zero-shot sim2real transfer [6, 7, 8, 9].

Traditionally, tuning the randomization parameters for a successful sim2real transfer is a manual trial-and-error process, but a few recent works attempt to improve this. For example, [11] propose a Bayesian optimization method to optimize the distributions, which may scale poorly with the number of randomizations. Another work [10] uses meta-learning to learn a small set of randomization settings which simultaneously maximize the utility of the learned policy and the diversity of the settings. Our approach is markedly different from and much simpler than these existing approaches; we propose to use hierarchy to introduce modularity into the domain randomization process, which is very sensible for quadrupedal locomotion. One can first focus on randomizations which allow for learning a robust locomotive policy; once a basic locomotive policy is found and fixed, many of the randomizations used to train it are no longer relevant, or may be replaced with simpler variants.

3

1. Pre-train π_{lo} to reach randomly sampled goals g from some fixed distribution; i.e, train basic locomotive policy in simulation with appropriate domain randomizations (e.g., joint frictions, random terrain).

2. Freeze π_{lo} and train π_{hi} in simulation to direct π_{lo} to solve more complex task. Use appropriate domain randomization (e.g., noisy high-level actions).

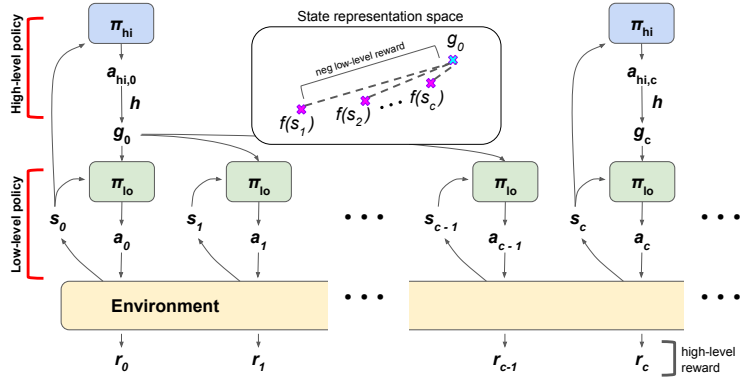3. Transfer full hierarchical policy to the real world in a zero-shot fashion.

Figure 3: We propose to solve tasks using a hierarchical policy in which a high-level policy π_{hi} produces high-level actions a_{hi} which are transformed to goals g that a lower-level policy π_{lo} is trained to reach. In quadrupedal locomotion tasks, this is a natural decomposition: the low-level policy may be trained to produce behaviors to reach various goals; i.e., $g := (g_x, g_y)$ is a desired point and the state representation $f(s)$ is simply the x, y coordinates of the quadruped. The high-level policy then solves a task by iteratively directing the low-level to a sequence of goal locations.

Learning a high-level policy on top of this basic locomotive policy will utilize distinct, and usually fewer, domain randomizations.

3 Hierarchical Sim2Real

We now describe our modular approach to solving quadrupedal manipulation via locomotion. Although this paper focuses on applications to quadrupeds, our approach is general, and can be applied to other problems involving mobile robots. Therefore, we begin with a general overview of the method and subsequently describe the details of applying it to quadrupedal locomotion tasks. In these descriptions we consider a standard Markov decision process (MDP) setting [32], in which each step in the environment consists of a state observation s, an action a sampled from the policy, a reward r, and a next state s'. We wish to find an optimal behavior policy to maximize cumulative discounted future reward, with discount given by some $0 \leq \gamma < 1$.

3.1 Overview

The paradigm of hierarchical sim2real proposes to break up a task into low-level and high-level controls which can be successively learned by separate policies. For ease of exposition, we restrict our description to two-level hierarchies, although the paradigm is generalizable to hierarchies of depth greater than two. Taking inspiration from recent successes in simulation [12, 33], we focus on *goal-conditioned hierarchies* in which the low-level policy is trained to reach randomly sampled *goal* states. The high-level policy is then trained to solve the original task by iteratively setting goals for the low-level policy. See Figure 3 for a visual diagram of this structure. In general, goals are not specified in the raw observation space but rather in a lower-dimensional representation space, with mapping from raw observations to low-dimensional representations given by some function f. In some settings, the function f is easy to hard-code (e.g., in tasks involving quadrupedal locomotion, it is natural to choose $f(s)$ to be the x, y coordinates of the robot). In other settings, the function f may be learned via unsupervised objectives [33].

Goal-conditioned hierarchies are desirable because low-level policies can be trained before any reward signal is observed from the original task, or even before the task is known. That is, the low-level policy is trained in an unsupervised fashion based solely on intrinsic (goal-reaching) rewards. Accordingly, training is performed in two phases, one for each level of the hierarchy. First, the low-level policy $\pi_{\mathrm{lo}}(a|s, g)$ is trained to reach goals randomly sampled from some distribution $\mathcal{G}$. That is, we train the low-level policy via RL to maximize cumulative reward given by,

$$r_{\mathrm{lo}}(s, a, s', g) := -D(f(s'), g) + r_{\mathrm{aux}}(s, a, s', g), \tag{1}$$
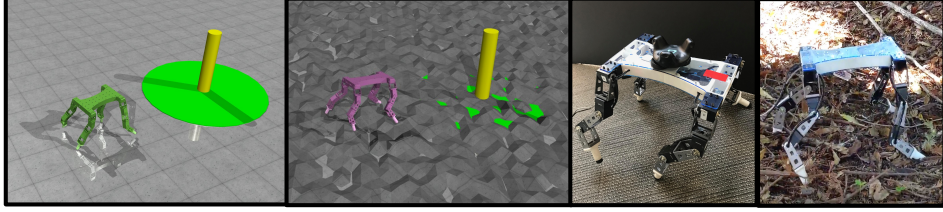
4

Figure 4: The low-level policy is trained to perform simple goal-reaching in simulation (left). We apply a variety of domain randomizations, including randomized height fields, which we found to be most helpful (middle). These randomizations lead to a robust locomotion policy in a variety of real-world environments, including both indoor and outdoor terrain (right).

where D is some distance function, g is sampled from $\mathcal{G}$, and r_{aux} is an optional additional auxiliary intrinsic reward. Appropriate domain randomization should be applied during simulated training in order to encourage suitable zero-shot transfer of goal-reaching behaviors in the real world (we elaborate on the randomizations we use for quadrupedal locomotion in the next subsection).

Given a pre-trained low-level policy, the high-level policy is then trained via RL to solve the task through iterative goal-proposing. More specifically, a high-level policy $\pi_{\text{hi}}(a_{\text{hi}}|s)$ is trained to take in a current observation s and return a high-level action a_{hi}. The high-level action is transformed to a goal $g := h(a_{\text{hi}})$ for the low-level via some pre-defined, fixed mapping h (in simple settings, h can be the identity, $g := a_{\text{hi}}$). The low-level is then run for some number of steps to generate behavior $\tau = \{(s_i, a_i, r_i, s_i')\}_{i=0}^{c-1}$ sampled from the environment, where $s_0 := s$ and $a_i \sim \pi_{\text{lo}}(\cdot|s_i, h(a_{\text{hi}}))$. Following previous work [12], we use a fixed c ($c = 10$). The reward for the high-level is given by,

$$r_{\text{hi}}(s, a_{\text{hi}}, \tau) := \sum_{i=0}^{c-1} \gamma^i r_i. \tag{2}$$

Any standard RL algorithm may be used to optimize the policy π_{hi} with respect to this task reward. As for the low-level, appropriate domain randomization should be applied during high-level training. As we will see concretely in the case of quadrupedal tasks, the domain randomization required for high-level training is often different from the domain randomization used for the low-level. It is often also much simpler than if a non-hierarchical policy is trained to solve the task from scratch. Indeed, once one has access to a suitable goal-reaching low-level policy, this policy is fixed, and thus there is no need to employ randomizations relevant to these low-level primitives (e.g., randomizations on the joints of a walking robot or terrain surface properties). Rather, the randomizations can focus more on high-level properties of the environment, such as agent and object locations or dimensions.

3.2 Quadrupedal Manipulation via Locomotion

Hierarchical sim2real as a paradigm for solving real-world tasks lends itself well to applications involving quadrupeds. In these settings, a low-level policy may be trained to perform basic locomotion, while a high-level policy is trained to modulate these locomotive primitives in order to solve a much more complex task, involving long time-horizons, object interaction, and multiple agents.

3.2.1 Training

We train the low-level to reach goals in x, y space; i.e., $f(s)$ returns the x, y (ground) coordinates of the torso of the quadruped. During low-level training we sample goals (g_x, g_y) as,

$$(g_x, g_y) := (u \cos v, u \sin v), \text{ where } u \sim \text{uniform}(1, 2), v \sim \text{uniform}(-0.5, 0.5). \tag{3}$$

We reward the low-level with negative L2 distance to the goal; i.e., $D := ||\cdot||_2$. To encourage better walking behavior, we define r_{aux} to augment the reward with an additional three terms,

$$r_{\text{aux}}(s, a, s', g) := r_{\text{upright}}(s') + r_{\text{heading}}(s', g) + r_{\text{bonus}}(s', g). \tag{4}$$

The first term r_{upright} encourages the quadruped to remain upright; i.e., penalizes it when the torso's z-coordinate is too low. The second term r_{heading} encourages the quadruped to face the goal point; i.e., rewards it for having a high normalized dot-product between $g - s'$ and the torso's heading. Lastly, the third term r_{bonus} augments the reward with a bonus when the quadruped is within 0.5m of
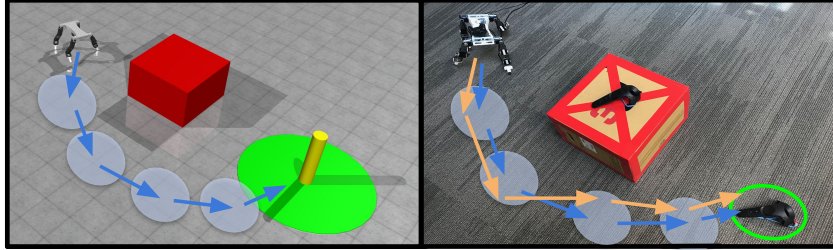
Figure 5: The high-level policy directs the agent by setting relative position goals. In this way, it combines locomotive primitives to solve a more complex task, as in this example trajectory for the *Avoid* task. To account for potential unknown gaps between low-level behavior in simulation and reality (blue vs. orange arrows), during training we pollute the high-level actions with random noise.

the goal. We run episodes for length 40, corresponding to 4 seconds of real-world time, terminating the episode if the quadruped falls.

Once the low-level is trained, we fix it and train the high-level to direct the low-level to solve a more complex task over a longer horizon (200 steps in simulation, 20 seconds in the real world). In the case of a single mobile agent, the high-level actions are 2-dimensional. A high-level action $a_{\mathrm{hi}} := (a_{\mathrm{hi},0}, a_{\mathrm{hi},1})$ is transformed to a low-level goal via the polar mapping,

$$h(a_{\mathrm{hi}}) := (u(a_{\mathrm{hi}}) \cos v(a_{\mathrm{hi}}), u(a_{\mathrm{hi}}) \sin v(a_{\mathrm{hi}})), \tag{5}$$

where, u, v are linear functions: $u(a_{\mathrm{hi}}) := 0.5 + 0.3 \cdot a_{\mathrm{hi},0}$; $v(a_{\mathrm{hi}}) := 0.2 \cdot a_{\mathrm{hi},1}$.

To ensure that the low-level generalizes to potentially very different environments, when activating the low-level policy, we zero-out the components of each observation s which correspond to the 6 positional and rotational degrees of freedom of the torso. Thus, high-level actions always correspond to *relative* goals (similar to previous work in simulation [12]).

In multi-agent environments we allow the high-level policy to control multiple mobile agents concurrently in order to coordinate their behavior. In these settings, the high-level actions are $2n$-dimensional, where n is the number of quadrupedal agents, and h is defined analogously to Equation 5. The modularity enabled through hierarchy in these multi-agent settings is clear. Rather than interact with the low-dimensional action space of a multi-agent environment directly, the imposition of hierarchy allows us to first train a single low-level policy and then train a high-level which effectively treats each agent as a point mass. Hierarchy here also makes the domain randomization problem more tractable, avoiding the duplication that will otherwise be necessary for training two agents from scratch.

We learn each policy by fitting a neural network using natural policy gradient [34, 35]. Each policy is a feed-forward neural network with two hidden layers, each of dimension 32 and using tanh activations. We train the low-level for 15000 iterations using batches of 100 trajectories. The high-level is then trained for 5000 iterations using batches of 200 trajectories.

3.2.2 Domain Randomization

In addition to simplifying the training of complex behaviors, hierarchy also introduces modularity into the domain randomization process. For low-level domain randomization, we randomize a number of parameters commonly used in sim2real methods [8]: joint damping, joint friction, actuator gain, total mass (gravitational force), and surface friction. As in previous works, we also apply random forces and torques to the torso (randomly applying anew every 10 simulated steps) and use sticky actions (repeat the previous action) with probability 0.2. Finally, we replace the floor with a randomly generated height field; i.e., we replace the terrain with a randomly generated triangular mesh (see Figure 4). In MuJoCo [13], this corresponds to simply adding an hfield asset to the model and randomizing its parameters. We are not aware of previous work which utilizes this randomization. In our experience, we found this randomization to be the *most crucial* for producing robust walking gaits, and hope that future quadrupedal locomotion research can make further use of this finding. See the appendix for more randomization details.

Once the low-level is sufficiently trained, it is effectively a robust moving point mass from the perspective of the high-level, so the next stage of training requires significantly fewer domain randomizations. Still, although the pre-trained low-level policy is robust, it is not perfect, and there

6

remains a non-negligible gap between its behavior in simulation and the real world. For example, a 'walk forward' command may result in real-world behavior that is slightly biased to the left or right due to hardware imperfections or terrain variability; see Figure 5. Rather than re-use the various randomizations employed for low-level training, we instead use a single, simple randomization to account for this gap: We add noise to the high-level actions (uniformly sampled between -1 and 1) before passing them to the low-level. This is another example where the benefits of hierarchy for sim2real applications are concrete. Training a non-hierarchical policy on these more complex, longer horizon tasks would potentially involve the cumbersome tuning of various short-term and long-term randomization parameters. In contrast, when hierarchy is imposed, the quadruped is simply a noisy point mass from the perspective of the high-level policy, and thus appropriate domain randomization is much simpler to apply. In the multi-agent task, in addition to adding noise to the high-level actions, we also randomize the object dimensions.

With these domain randomizations, the resulting hierarchical policy is able to successfully transfer to the real world in a zero-shot fashion.

4   Experiments

We present experimental results of applying hierarchical sim2real on several real-world tasks, building up to a demonstration of coordinated multi-agent object manipulation. We begin by describing the hardware and the tasks on which we evaluate.

4.1   Hardware

For our quadrupedal robot we use D'Kitty, a robust and affordable locomotion platform (4200 USD) designed and built in-house. The robot consists of a 3D-Printed torso and four legs, each comprised of three Dynamixel Servo Motors that provide feedback on joint angle and angular velocity. We attach an HTC Vive tracker to the torso to further observe the global position and orientation of the robot. In tasks involving objects (blocks), we additionally use an HTC Vive controller attached to the object to observe its global position and orientation.

4.2   Tasks

We consider three tasks (see Figure 2 for a visual diagram):

- **Avoid**. The quadruped must walk to a target location while avoiding a block object. We provide the agent a reward at each step consisting of (1) a term measuring negative L2 distance of the agent to the target, (2) a bonus for being within 0.5m of the target location, (3) a term which penalizes the agent for being within 0.5m of the block location, and (4) a term which encourages the agent to be upright (same as r_{upright} in Equation 4).
- **Push**. The quadruped must push a block object to a desired target location. We provide the agent a reward at each step consisting of (1) a term measuring negative L2 distance of the agent to the block, (2) a term measuring negative L2 distance of the block to the target, (3) a bonus for when the block is within 0.5m of the target location, and (4) a term which encourages the agent to be upright (same as r_{upright} in Equation 4).
- **Coordinate**. Two quadrupeds must coordinate to push a long block such that the two ends of the block match two desired target locations. We provide the agent a reward at each step consisting of (1) the sum of the negative L2 distances of each end of the block to the desired target locations, (2) a bonus for when both ends of the block are withing 0.3m of the desired target locations, and (3) a term which encourages both agents to be upright (the sum of r_{upright} for each agent).

Each task is run for 200 steps (corresponding to 20 seconds in the real world) without termination. The task observations include the joint angles, joint angular velocities, global position, and global orientation of each quadruped. In *Avoid* and *Push*, the observations include positions of the target and block locations relative to the quadruped. In *Coordinate*, the observations include the orientation of the block, the positions of each target location relative to each end of the block, and the positions of each end of the block relative to each quadruped.

In simulation, the positions of the quadrupeds, objects, and target locations are initialized randomly. In the real world, we measure the success rate of the hierarchical policy on the three tasks (whether it reached the target or not) based on fixed initializations. See the appendix for further details.

	Hierarchical Sim2Real	No high-level randomization	No hierarchy, with randomization	No hierarchy, no randomization
Avoid	**73.3 ± 6.7%**	56.7 ± 18.6%	0 ± 0%	0 ± 0%
Push	**73.3 ± 12.0%**	70.0 ± 5.8%	0 ± 0%	20.0 ± 5.8%
Coordinate	**90.0 ± 5.7%**	80.0 ± 15.3%	0 ± 0%	6.7 ± 3.3%

Table 1: We present success rates (with standard error) on the three tasks comparing our method (hierarchical sim2real) to a number of ablations. Each percentage is the result of 30 attempts – ten attempts per three independently trained models. These ablations show that hierarchy is crucial to success on these tasks. With no hierarchy (right two ablations), success rates fall precipitously. Once hierarchy is employed, domain randomization on the high-level (uniform noise added to high-level actions) can provide a modest improvement.

	All randomizations	No height field	No randomizations
Distance Travelled	**1.26 ± 0.43**	0.85 ± 0.36	0.44 ± 0.14
Percentage of Falls	**0%**	**0%**	80%

Table 2: We present the performance of the low-level policy at basic locomotion. We run the low-level policy to walk forwards for five seconds, and show the average distance travelled (in meters, with standard error) and the rate of falling. Each statistic is the result of 30 attempts – ten attempts per three independently trained models. Without randomizations, the policy is unusable in the real world, due to high rate of falling. With randomizations, the rate of falling goes to zero. We find the most impactful randomization to be the height field, which improves the average distance travelled by about 50%.

4.3 Results

After training the low-level and high-level in simulation, we evaluate the full hierarchical policy in the real world. We present the results in Table 1. To ensure confidence in our metrics, we measure success over 30 trials, where each set of 10 is a separately trained model.

Our results present the hierarchical sim2real approach compared to a number of ablations. In 'No hierarchy, with randomization' and 'No hierarchy, no randomization,' we train non-hierarchical policies in simulation with and without randomization, respectively. While both of these policies perform poorly in the real world, the variant with domain randomization surprisingly performs even worse. Although not shown (see the appendix), we note that *in simulation* the 'No hierarchy, no randomization' ablation can reasonably solve all the tasks, while the 'No hierarchy, with randomization' cannot. We therefore conclude that the randomizations which allow for learning robust basic locomotion are too aggressive to learn more complex locomotion, and lead to an overly conservative policy, as shown by our results. This is strong evidence of the need to apply hierarchy in a sim2real context. Hierarchy introduces modularity into the sim2real process with real-world benefits.

The two hierarchical ablations – 'Hierarchical Sim2Real' (our proposed method) and 'No high-level randomization' – achieve success rates much higher than either of the non-hierarchical ablations. Once a low-level basic locomotive policy is learned, the domain randomizations necessary for low-level training can be discarded, and a high-level policy can be easily trained. Even with no randomizations ('No high-level randomization'), high-level training in simulation achieves good zero-shot results in the real world. When adding basic high-level randomizations (as described in Section 3.2.2), we are able to further improve the success rates, reaching 90% success on *Coordinate*.

We provide further insight into low-level training in Table 2. The randomizations described in Section 3.2.2 lead to a robust locomotive policy. In comparison, a policy trained with no randomizations may perform well in simulation, but in the real world often falls shortly after initiating movement. We found height field randomization to be the most useful element leading to a robust policy. It alone provides a 50% boost in performance.

5 Conclusion

We have presented successful zero-shot transfer of policies trained in simulation to perform difficult locomotion and manipulation via locomotion tasks. The key to our method is the imposition of hierarchy, which introduces modularity into the domain randomization process and enables the learning of increasingly complex behaviors. We intend to continue our efforts in future work, further exploiting hierarchies to demonstrate more difficult tasks with a greater number of cooperating agents.

References

[1] R. D. Alexander. The evolution of social behavior. *Annual review of ecology and systematics*, 5(1):325–383, 1974.

[2] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *CoRR*, abs/1812.11103, 2018. URL http://arxiv.org/abs/1812.11103.

[3] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 3, pages 2619–2624. IEEE, 2004.

[4] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[5] 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 preprint arXiv:1804.10332*, 2018.

[6] F. Sadeghi and S. Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[7] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[8] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[9] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

[10] B. Mehta, M. Diaz, F. Golemo, C. J. Pal, and L. Paull. Active domain randomization. *arXiv preprint arXiv:1904.04762*, 2019.

[11] F. Ramos, R. C. Possas, and D. Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv preprint arXiv:1906.01728*, 2019.

[12] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *NeurIPS*, 2018.

[13] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[14] T. Asfour, J. Borràs, C. Mandery, P. Kaiser, E. E. Aksoy, and M. Grotz. On the dualities between grasping and whole-body loco-manipulation tasks. In *Robotics Research*, pages 305–322. Springer, 2018.

[15] I. Mordatch, Z. Popović, and E. Todorov. Contact-invariant optimization for hand manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 137–144. Eurographics Association, 2012.

[16] O. A. Silva, P. Sigel, W. Eaton, C. Osorio, E. Valdivia, N. Frois, and F. Vera. Crabot: A six-legged platform for environmental exploration and object manipulation. In *Proceedings of the 4th Congress on Robotics and Neuroscience*, number 2312, pages 46–51, 2018.

[17] E. Tuci, M. H. Alkilabi, and O. Akanyeti. Cooperative object transport in multi-robot systems: A review of the state-of-the-art. *Frontiers in Robotics and AI*, 5:59, 2018.

[18] M. J. Mataric, M. Nilsson, and K. T. Simsarin. Cooperative multi-robot box-pushing. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 3, pages 556–561. IEEE, 1995.

[19] X. Olivares, J. Ruiz-del Solar, and J. Zagal. On-line learning of an object manipulation behavior for legged robots. 2004.

[20] Y. Sanz, J. de Lope, et al. Applying reinforcement learning to multi-robot team coordination. In *International Workshop on Hybrid Artificial Intelligence Systems*, pages 625–632. Springer, 2008.

[21] Y. Wang and C. W. De Silva. Multi-robot box-pushing: Single-agent q-learning vs. team q-learning. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3694–3699. IEEE, 2006.

[22] S. Feng. Online hierarchical optimization for humanoid control. 2016.

[23] M. Zucker, J. A. Bagnell, C. G. Atkeson, and J. Kuffner. An optimization approach to rough terrain locomotion. In *2010 IEEE International Conference on Robotics and Automation*, pages 3589–3595. IEEE, 2010.

[24] M. Hutter, H. Sommer, C. Gehring, M. Hoepflinger, M. Bloesch, and R. Siegwart. Quadrupedal locomotion using hierarchical operational space control. *The International Journal of Robotics Research*, 33(8):1047–1062, 2014.

[25] L. Sentis and O. Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(04):505–518, 2005.

[26] A. Herzog, L. Righetti, F. Grimminger, P. Pastor, and S. Schaal. Balancing experiments on a torque-controlled humanoid with hierarchical inverse dynamics. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 981–988. IEEE, 2014.

[27] D. Jain, A. Iscen, and K. Caluwaerts. Hierarchical reinforcement learning for quadruped locomotion. *arXiv preprint arXiv:1905.08926*, 2019.

[28] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[29] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[30] I. Mordatch, N. Mishra, C. Eppner, and P. Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 242–248. IEEE, 2016.

[31] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[32] M. L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[33] O. Nachum, S. Gu, H. Lee, and S. Levine. Near-optimal representation learning for hierarchical reinforcement learning. *ICLR*, 2019.

[34] A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade. Towards Generalization and Simplicity in Continuous Control. In *NIPS*, 2017.

[35] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

A Task Details

For all tasks, the low-level action space is joint angles; i.e., it directs the motors via desired joint angles. Thus, the low-level action space is 12-dimensional (3 motors per each of the 4 legs).

For *Avoid* we use a simulated block of dimensions $0.6 \times 0.6 \times 0.4$ meters. The target location is set at the beginning of each episode as $(r_T \cos \theta_T, r_T \sin \theta_T)$, where $r_T \sim \text{uniform}(1.5, 2.5)$ and $\theta_T \sim \text{uniform}(-1, 1)$. The block location is then set to $(r_B \cos \theta_B, r_B \sin \theta_B)$ where $r_B = \max\{0.6, r_T \cdot \text{uniform}(0.3, 0.8)\}$ and $\theta_B = \theta_T + \text{uniform}(-0.5, 0.5)$.

For *Push* we use a simulated block of dimensions $0.6 \times 0.6 \times 0.4$ meters. The target location is set at the beginning of each episode as $(r_T \cos \theta_T, r_T \sin \theta_T)$, where $r_T \sim \text{uniform}(1.5, 2.5)$ and $\theta_T \sim \text{uniform}(-1, 1)$. The block location is then set to $(r_T \cos \theta_T + r_B \cos \theta_B, r_T \sin \theta_T + r_B \sin \theta_B)$ where $r_B \sim \text{uniform}(0.6, 1.2)$ and $\theta_B \sim \text{uniform}(\pi/3, 5\pi/3)$. If this results in the block and agent being within 0.5m, we re-sample.

For *Coordinate* we use a simulated block of dimensions $0.3 \times 1.4 \times 0.3$ meters. The block location is initialized to always be in front of the agents. The two target locations are then set to be $(r_T \cos(\theta_T) \pm 0.55 \cos \theta_B, r_T \sin \theta_T \pm 0.55 \sin \theta_B)$, where $r_T \sim \text{uniform}(1, 1.5)$, $\theta_T \sim \text{uniform}(-1, 1)$, $\theta_B \sim \theta_T + \pi/2 + \text{uniform}(-0.5, 0.5)$.

During real-world evaluations, we initialize the tasks as follows:

- **Avoid**. The quadruped is initialized facing a box 1m away. Taking the robot's heading as the x-axis, we place the target location at $(2, \pm 0.5)$ (i.e., have the time at $(2, 0.5)$ and half the time at $(2, -0.5)$. Success is measured as reaching within 0.5m of the target without moving the box.
- **Push**. The quadruped, box, and target initializations are the same as for Avoid. Success is measured as pushing the box within 0.5m of the target location.
- **Coordinate**. The two quadrupeds are initialized facing a long, horizontal box. The target location is set to be a 1.5m frontwards translation of the box. Success is measured as both ends of the box reaching within 0.3m of the target position of the ends of box.

B Domain Randomization

The code snippets below show the specific randomizations we employed for low-level training.

Listing 1: Dynamics model randomizations for low-level training.

```
dof0 = 6  # six global observations
dofn = dof0 + 12  # twelve joints
self.sim.model.dof_damping[dof0:dofn] = \
    self.np_random.uniform(low=0.9, high=1.1)
self.sim.model.dof_frictionloss[dof0:dofn] = \
    self.np_random.uniform(low=0.001, high=0.005)
functions.mj_setTotalmass(
    self.sim.model,
    self.np_random.uniform(low=1.6, high=2.0))
self.sim.model.geom_friction[:] = self.np_random.uniform(
    low=[0.8, 0.003, 0.00005], high=[1.2, 0.007, 0.00015])
self.sim.model.actuator_gainprm[:, 0] = self.np_random.uniform(
    low=4, high=6)
self.sim.model.actuator_biasprm[:, 1] = \
    -self.sim.model.actuator_gainprm[:, 0]


# regenerate height map and upload to GPU
self.sim.model.hfield_data[:] = self.np_random.uniform(
    low=0, high=.050, size=np.shape(self.sim.model.hfield_data))
if (len(self.sim.render_contexts) > 0):
    functions.mjr_uploadHField(self.sim.model,
                               self.sim.render_contexts[0].con, 0)
```

C   Training and Ablation Details

By default, each policy is a feed-forward neural network with two hidden layers, each of dimension 32 and using `tanh` activations. For the non-hierarchical ablations we increase the hidden dimension to 64. For the non-hierarchical ablation with randomization, we apply all the randomizations used by default for low-level training and additionally a small (uniform between -0.1 and 0.1) noise on the torso x, y coordinate observations.

D   Simulation Results

	Hierarchical Sim2Real	No high-level randomization	No hierarchy, with randomization	No hierarchy, no randomization
Avoid	**$86.5 \pm 5.8\%$**	$76.7 \pm 3.3\%$	$0 \pm 0\%$	$33.3 \pm 20\%$
Push	**$93.3 \pm 8.8\%$**	$90.0 \pm 11.5\%$	$0 \pm 0\%$	$86.7 \pm 6.7\%$
Coordinate	$80.0 \pm 3.3\%$	$66.7 \pm 3.3\%$	$0 \pm 0\%$	**$86.7 \pm 30\%$**

Table 3:   We present success rates (with standard error) on the three tasks in simulation. Each percentage is the result of 30 attempts – ten attempts per three independently trained models. In simulation we find that hierarchy helps, but less so than in the real world, especially on the *Coordinate* task, for which a non-hierarchical policy performs best. This is potentially because our low-level policies are not trained in the presence of blocks. We also see that learning a policy from scratch (no hierarchy) with domain randomization is futile. In order to successfully transfer learned behaviors to the real world, both hierarchy and domain randomization are necessary.