Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot

Sehoon Ha, Joohyung Kim, and Katsu Yamane

***Abstract*— In this paper, we present an automated learning environment for developing control policies directly on the hardware of a modular legged robot. This environment facilitates the reinforcement learning process by computing the rewards using a vision-based tracking system and relocating the robot to the initial position using a resetting mechanism. We employ two state-of-the-art deep reinforcement learning (DRL) algorithms, Trust Region Policy Optimization (TRPO) and Deep Deterministic Policy Gradient (DDPG), to train neural network policies for simple rowing and crawling motions. Using the developed environment, we demonstrate both learning algorithms can effectively learn policies for simple locomotion skills on highly stochastic hardware and environments. We further expedite learning by transferring policies learned on a single legged configuration to multi-legged ones.**

I. Introduction

Some creatures in nature can change their number of limbs in various situations. In a recent study on harvestmen [1], also known as "daddy longlegs", researchers showed that harvestmen voluntarily release legs during a predator encounter and recover their locomotion capability of speed and steering control after a short period of time. Even without this ability of autotomy, most of creatures are able to locomote after changing their body configurations, by means of learning to adjust themselves. Motivated by this wonder of nature, we implemented and investigated learning locomotion skills of a robot hardware while varying its configuration.

The process of designing robot walking motions largely relies on manual efforts of engineers who can discover appropriate gaits for the given morphology from their prior knowledge. Although experienced engineers can design surprisingly effective gaits for a wide range of robots, it is less practical for reconfigurable robots [2] which can have a huge number of different morphologies. For example, our modular legged robot, Snapbot [3], can be configured into 700 different forms by attaching three types of legs to one or more of the six attachment terminals of the hexagonal base unit. While some configurations, such as symmetric quadrupeds, are easy to find appropriate motions, many configurations are not intuitive to find the best gait especially when they are highly asymmetric. Therefore, automated design of motions is highly preferable when the robot can be easily reconfigured into many different morphologies.

Recently, researchers have demonstrated sampling efficiency of deep reinforcement learning (DRL) techniques on various continuous control problems. A recent benchmark [4] extensively compares the performances of the state-of-the-art DRL algorithms on many tasks including swimming, hopping, walking, and running in simulation. In contrast, deep learning on hardware has been investigated mostly for manipulation tasks [5], [6] and rarely for legged motions. In our experience, even a simple crawling task becomes much more challenging on real hardware due to various unmodeled or approximated dynamics, such as link deformation, joint slackness, and contacts. These unexpected properties result in highly asymmetric and stochastic dynamics, which are difficult to simulate by simply inserting artificial Gaussian noises into simulation.

Sehoon Ha, Joohyung Kim, and Katsu Yamane are with Disney Research, Pittsburgh, PA 15206, USA katsu.yamane@gmail.com

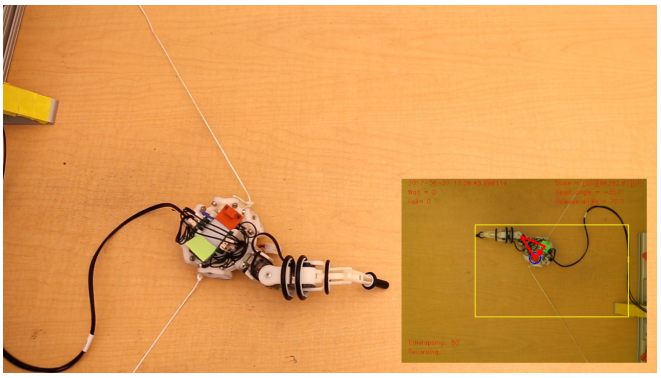

Fig. 1. We implemented an automated framework for training a control policy directly on hardware of a reconfigurable robot. (Figure) The robot platform, Snapbot, in the learning environment. (Bottom Right) The vision image for computing the reward.

Although DRL serves as an automated learning framework, it might be still time consuming if we train all controllers from scratch. In the worst case, learning algorithms may not be able to find the optimal policies for some complex configurations with many degrees of freedom. A common strategy to alleviate this problem is to exploit learned policies on simple configurations as prior knowledge that can guide learning for more complex robots. Although various advanced techniques for learning and transfer of hierarchically represented policies have been proposed [7], [8], we test the simple approach of initializing the policy using previously learned parameters.

In this paper, we present an automated environment for deep reinforcement learning of locomotion tasks consisting of a vision tracker and a reseting mechanism. Using this environment, we apply two state-of-the-art learning algorithms: Trust Region Policy Optimization (TRPO) [9] and Deep Deterministic Policy Gradient (DDPG) [10] directly on the

hardware of a reconfigurable legged robot. As a preliminary study, we simplify the problem by limiting the target motion to rowing or crawling with up to three legs. First, we train a neural network policy for crawling of a single-legged configuration. Then we apply the policy for single-legged locomotion to multi-legged configurations by initializing the policy parameters with the learned policies. The experimental results demonstrate that both algorithms are able to learn effective control policies directly on hardware for three different types of legs. The results also indicate that we can further expedite learning by reusing the learned parameters for more complex morphologies.

II. Related Work

Reconfigurable robotic systems have recently drawn a lot of attention as a paradigm for achieving more versatile and resilient motions [11], [12]. For example, Romanishin *et al.* [13] developed a self-reconfigurable cubic robot system where each modular unit can move independently using a self-contained flywheel mechanism. Kalouche *et al.* [2] designed a legged robot that can be reconfigured with a set of modules including series-elastic actuators, sensors, links, and end-effectors, which is more similar to our modular robot. These early studies have motivated various commercial modular robot products, such as [14], [15]. In addition, it is worth mentioning the self-recovering algorithm of Bongard *et al.* [16] when the robot faces unexpected damage on its leg.

There exists a large body of research devoted to finding locomotion controllers for new robots and virtual creatures in the fields of robotics, machine learning, and computer animation. One of the most popular approaches is to optimize time-discretized trajectories by minimizing target physical costs (e.g. energy usage) while satisfying dynamics constraints. For instance, Wampler and his colleagues proposed a space-time optimization technique [17] for generating locomotion controllers for various creatures from a small number of inputs and adapting locomotion styles to new animals. The optimized trajectories can further be used to guide search algorithms for learning complex behaviors, especially for challenging control problems with complex and discontinuous dynamics [18].

In recent years, various advances have been made on deep reinforcement learning (DRL) algorithms to effectively train complex high-dimensional control policies. Mnih *et al.* [19] achieved human-level control on classic Atari games by improving the performance of Deep Q-Network (DQN) with two important techniques: *experience replay* and *target network*. Lillicrap *et al.* [10] further extended DQN to problems with high dimensional continuous state and action spaces with the actor-critic approach and deterministic policy gradient (DPG) [20]. Schulman *et al.* [9] proposed a learning algorithm called Trust Region Policy Optimization (TRPO) that guarantees near-monotonic improvement of control performance under a few theoretic assumptions.

Although DRL has been successfully applied to simulated environments, the learned control policies are likely to fail on hardware due to discrepancy between the dynamics of simulation and real worlds. To bridge the gap, researchers often applied model-learning algorithms that iteratively update the simulation model by building a probabilistic transition model from collected data samples to make it closer to real environment [21], [22]. Another approach is to directly train policies on the target hardware without using simulation at all. Due to many practical reasons, this approach has been first examined on manipulation tasks rather than legged locomotion. For instance, a few researchers [5], [6] developed a camera-to-servo controller for robotic manipulators by training a convolutional neural network that evaluates the probability of successful grasp. For robots with floating bases, Yamaguchi *et al.* [23] trained crawling motions on a real spider robot using Q-learning with discretized actions. Asada *et al.* [24] accelerated Q-learning on hardware of a mobile robot by ordering input states from easier ones to difficult ones. Recently, Luck *et al.* [25] demonstrated sample-efficient learning of the sand-swimming motion controller for a sea-turtle robot where the policy is represented as a linear combination of basis functions. In this paper, our goal is to design a locomotion controller in the form of neural networks, which is more commonly used in the literature of deep reinforcement learning.

III. Problem Formulation

In this section, we will describe the details of our control problem in the following order: Section III-A describes our robot and learning platform, Section III-B provides a mathematical formulation of the problem, and Section III-C presents the policy representation and learning algorithm.

A. Robot and Automated Learning Environment

Our robot can be reconfigured by attaching and detaching various types of legs (Fig. 2) to the base unit with magnetic mechanical coupling. The base unit has a hexagonal shape where each face has a socket at the center for attaching a limb and a triangular supporting structure at the bottom for reducing the friction force from the ground. Although it has an on-board microprocessor and battery (Fig. 2 top), we tethered it to a desktop computer because the microprocessor is not powerful enough to perform the learning computation, and the battery cannot support the whole learning process that usually takes a few hours. There exists three types of legs that each leg has two to three Dynamixel XL-320 position controlled servos: a roll-pitch leg (Type A), a yaw-pitch leg (Type B), and a roll-yaw-pitch leg (Type C). The robot can identify the configuration in real time and select appropriate gaits depending on the configuration among manually-designed rowing, crawling, and walking gaits.

We set up an autonomous learning environment with a vision-based tracking system and a resetting device for bringing the robot back to the initial position (Fig. 1 and Fig. 3). The vision system is implemented using a consumer-grade web camera mounted at a height of approximately 90 cm (Fig. 3). It tracks two visual feature points (green and red) on the base unit to reconstruct the global position and orientation of the robot.

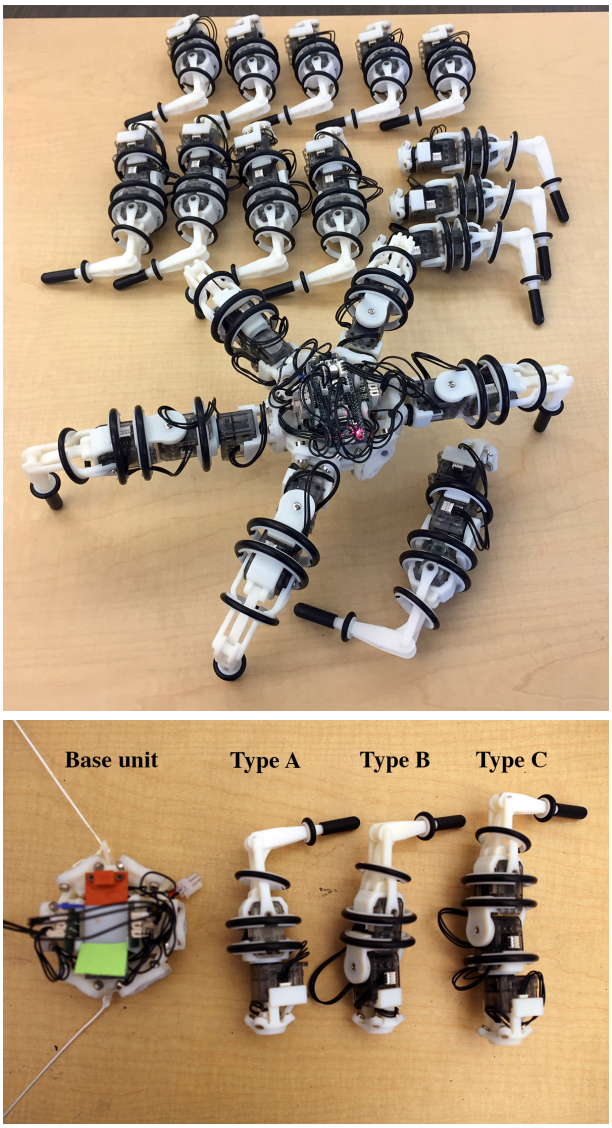

Fig. 2. (Top) Standalone Snapbot and its legs. It has a microcontroller and a battery in the base unit. (Bottom) The base unit and three types of modular legs.

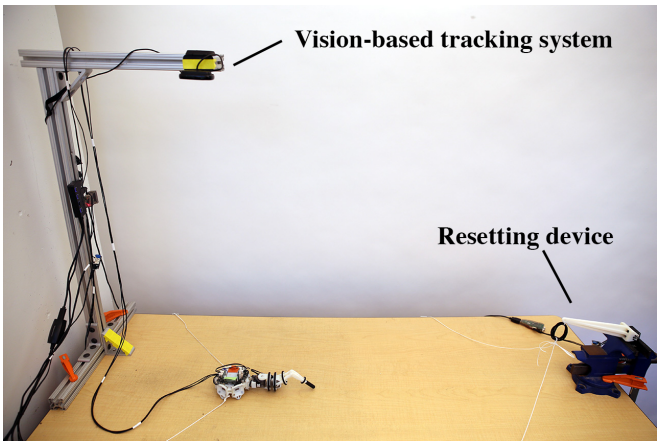

Fig. 3. The autonomous learning environment with a resetting device.

Resetting is an important functionality for the fully automated learning environment. We implement it using a simple one-DoF lever mechanism that can pull the robot back to its original position. The resetting lever is 25 cm long and connected to two points on the robot's base unit via two 1.5 m-long cables. We placed rubber bands between the cables and the lever to allow a little bit of slackness. At the beginning, the lever is at its rest pose which provides approximately 30 cm by 40 cm of free space to the robot. When the vision system detects the robot being out of the boundary, it pulls the robot back to the designated initial position by rotating 45° degrees and gets back to the rest pose.

B. Partially Observable Markov Decision Process

We formulate the control problem as a Partially Observable Markov Decision Process (POMDP) to illustrate the decision making problem with unknown state variables. Formally, a POMDP is described by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \Omega, q, \gamma)$ where each term respectively denotes the set of states, the set of actions, the probabilistic transition function, the reward function, the set of observation, the probabilistic observation function, and the discount factor.

The state $\mathbf{s} \in \mathcal{S}$ is defined as a three dimensional vector with the the global position ($\mathbf{p}_t = [x_t, y_t]^T$), and orientation ($\theta_t$) of the base unit in the horizontal plane, which are sampled at 5 Hz. The action $\mathbf{a} \in \mathcal{A}$ is defined as the target servo position $\bar{\mathbf{q}}$, where the control loop runs at 30 Hz. The stochastic transition dynamics function $p(\mathbf{s}_{t+\Delta t}|\mathbf{s}_t, \mathbf{a}_t)$ is intrinsically determined by the dynamics of the real robot and environment. Note that our definition of the state vector is a small subset of the robot's *real* state variables selected for calculating the reward. The real system has many more unmodeled state variables such as momenta, slackness of joints, current levels, cables, unevenness of the terrain, and so on.

The reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ is a very important term that governs the behavior of learning. Our reward function consists of three terms: *travel distance*, *direction*, and *effort*. The *travel distance* term is the main objective that describes the travel distance in a single gait cycle, which is defined as

$$r^{tr}(\mathbf{s}_t, \mathbf{a}_t) = k(t)\big((\mathbf{p}_t - \mathbf{p}_0) \cdot \mathbf{d}(\theta_0)\big) \quad (1)$$

where $\mathbf{d}(\theta)$ returns the direction vector from the orientation θ. The $\mathbf{p}_0$ and θ_0 are the position and direction at the beginning of the cycle. Simply using the distance results in discontinuous reward because the travel distance can be only measured at the end of a cycle. To smooth the landscape of the reward function, we introduce a scaling parameter $k(t)$ that returns 10.0 if the t is at the end of the gait cycle (2 seconds, 0.5 Hz) and returns 1.0 otherwise. The direction term measures the closeness to the original heading direction as

$$r^{dir}(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{d}(\theta_t) \cdot \mathbf{d}(\theta_0). \quad (2)$$

Finally, the effort term is defined as:

$$r^{eff}(\mathbf{s}_t, \mathbf{a}_t) = -|\mathbf{a}_t - \mathbf{q}_t|^2 \quad (3)$$

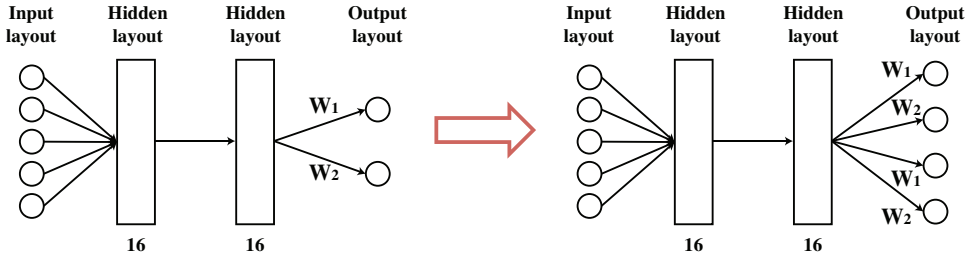

Fig. 4. To speed up learning, we initialize the neural network for the multi-legged configuration with a copy of the learned network for the single-legged configuration, duplicating the output neurons and their weights as necessary.

to penalize the uneven motion trajectories where $\mathbf{q}_t$ is the current servo positions. The reward function is a weighted sum of the three terms:

$$r(\mathbf{s}_t, \mathbf{a}_t) = w_1 r^{tr}(\mathbf{s}_t, \mathbf{a}_t) + w_2 r^{dir}(\mathbf{s}_t, \mathbf{a}_t) + w_3 r^{eff}(\mathbf{s}_t, \mathbf{a}_t) \quad (4)$$

where we use the constant weights $w_1 = 0.1$, $w_2 = 0.01$, $w_3 = 0.01$ for all experiments.

Because we aim for finding a periodic open-loop motion, the only observable variable is the phase variable t. Although this single variable is sufficient to learn the effective gaits, we can further accelerate learning by providing a binary encoding of the phase variable to distinguish the sub-phases that is inspired by the work of Peng *et al.* [8]. $\Phi_i(t) \in \{0, 1\}$ is 1 if and only if t is in its sub-phase interval, e.g. $\Phi_1(t)$ is 1 if $0 \leq t < 0.25$. The probabilistic observation function q simply returns the corresponding time observation at the queried moment. In our experiments, the binary encoding accelerates learning two to four times compared to the case when we only provide the phase t. Finally, we simply set the discount factor γ as 1.

C. Policy Representation and Learning Algorithms

A policy $\pi : \mathbf{o}_t \rightarrow \mathbf{a}_t$ is a stochastic function that finds the optimal actions for the given observation at time t. The policy is represented as a neural network that has two fully connected hidden layers with 16 $tanh$ activated neurons per each layer, which results in 388 parameters in total for a two DoFs configuration. For TRPO that trains a stochastic policy as input, we use the Normal distribution where its mean is the output of the neural network and the covariance is maintained with an additional set of parameters.

Once the policy is trained for a single leg morphology, we may be able to expedite the learning process by transferring the knowledge to multi-legged configurations. By assuming all the legs have the same joint configuration, we can initialize a new policy for multi-legged locomotion by duplicating output neurons and corresponding connections (Fig. 4). This initialization provides a good starting policy that is already able to move forward. For stochastic policies, we reset the covariance parameters to 0.3 for encouraging exploration.

Algorithm 1: Trusted Region Policy Optimization. TRPO [9] is an on-policy batch learning algorithm for solving large scale nonlinear control problems, which theoretically guarantees monotonic improvement of the reward. For each iteration, it roll-outs several trajectories using a probabilistic control policy and collects data samples on the *advantage* that is the difference between the return and the baseline. Then it carefully updates the policy parameters by solving an optimization problem with a constraint on the Kullback-Leibler divergence between the new and old policies to stay within the *trust region* that does not decrease the expected rewards. For more details, please refer to the original paper [9].

Some researchers reported that TRPO can be less data efficient [10] or shows early convergence of the policy network [26] for some scenarios. It might be because TRPO does not explicitly learn a critic network, or the constraint on a KL divergence does not allow large updates of the policy. However, its near-monotonic updating behavior provides us a couple of preferable properties when we apply the algorithm to real robots. First, it is able to near-monotonically increase the accumulated reward without falling into local minima. This property is very important for our problem because many policies are not able to move the robot at all in any directions, which results in a large basin of attraction for bad local minima. In addition, the algorithm samples new trajectories only in the neighborhood of the trajectories of the previous iteration. Therefore, it is easier to stop or roll back learning when we observe bad motions at the development stage.

Algorithm 2: Deep Deterministic Policy Gradient. DDPG [10] is an actor-critic approach based on the deterministic policy gradient proposed by Silver *et al.* [20]. Unlike batch learning algorithms, it continuously updates the policy parameters while exploring the input environments. It can be also considered as an extension of Deep Q Network (DQN) algorithm of Mnih *et al.* [19] to support high-dimensional continuous action spaces. The authors demonstrated that DDPG can efficiently learn competitive policies on various control problems in simulated environments. We refer [10] for a more detailed description of the algorithm.

IV. Experimental Results

We run all the learning and control codes on a single core of 3.40GHz Intel i7 processor. For both learning algorithms, we use the implementations in RLLAB [4]. The learning environment (called "gym" in RLLAB) is implemented with

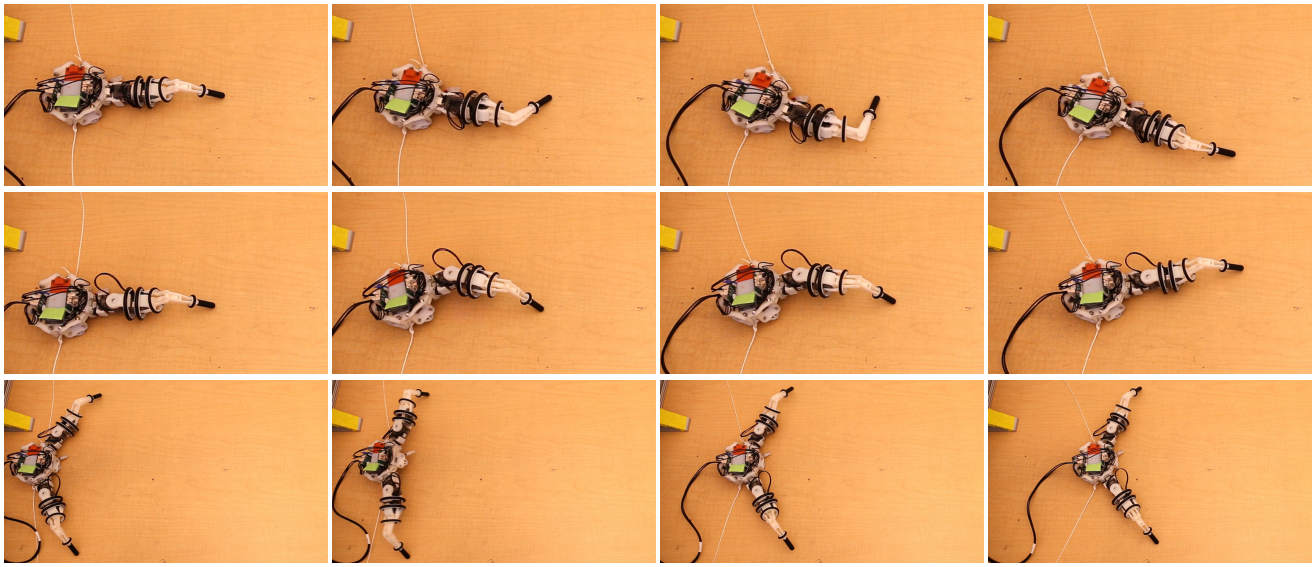

Fig. 5. (Top) The motion with one Type A leg. (Middle) The motion with one Type B leg. (Bottom) The motion with two Type B legs.

Dynamixel SDK for controlling servo motors and OpenCV for tracking the robot. A single learning process takes approximately three hours to run 60 iterations where each iteration evaluates 32 trajectories. For TRPO, we set the batch size to 320 (32 gait cycles) and the step size to 0.05. For DDPG, we used an epoch length of 320 (32 gait cycles), a mini-batch size of 32, and a learning rate of 10^{-3} and 10^{-4} for the actor and critic respectively.

In this section, we investigate the following research questions:

- Can the state-of-the-art DRL algorithms directly train a policy on hardware?
- Can we expedite learning by transferring policies to complex scenarios?

A. *Learning of One-legged Locomotion*

We first train policies for one-legged locomotion, which results in motions similar to rowing or crawling. The results of learning for all three types of legs are presented in Fig. 6. Because the target environment is highly stochastic and learning curves vary a lot, we plot all three trials rather than presenting the average and median per iterations. In general, both TRPO and DDPG are able to successfully train policies directly on hardware, which often outperform manually designed motion gaits. For instance, the maximum reward for the type A leg was 60.3, which is greater than the reward of a manually designed gait 34.8. We also observe some unexpected locomotion strategies. For instance, one of the policies for the Type A leg propels itself only using the pitch joint without the roll joint, while another policy for the Type B leg swings multiple times per one cycle. Please refer to Fig. 5 and the supplementary video for the motions of the learned policies.

We also observe differences between TRPO and DDPG on the three problems. First, TRPO robustly learns all policies with near-monotonic improvement on the cumulative reward

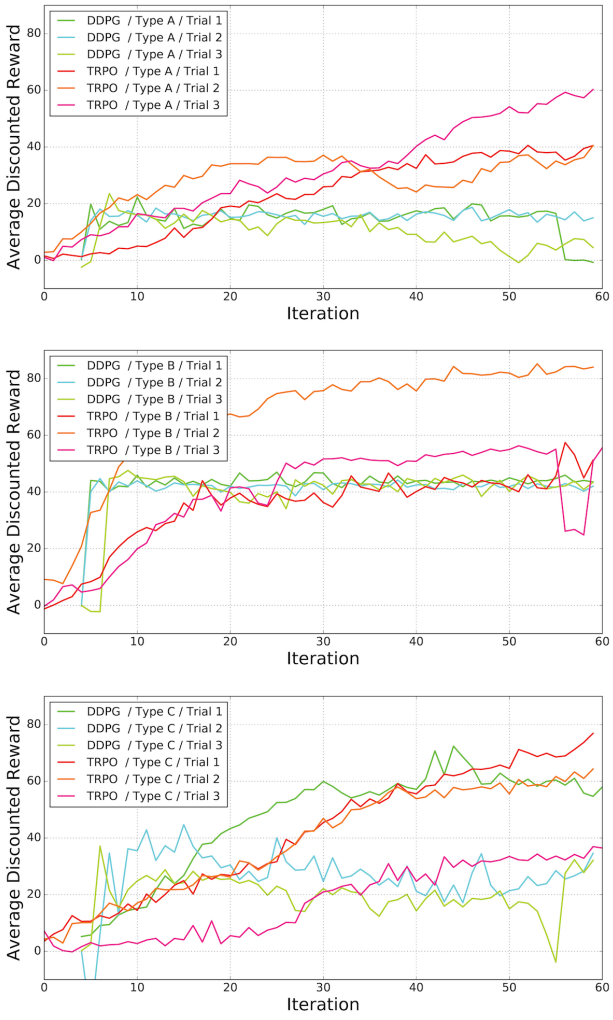

Fig. 6. Learning curves for single-legged locomotion with three types of legs (three trials per leg type). (Top) The Type A leg with roll-pitch joints. (Middle) The Type B leg with yaw-pitch joints. (Bottom) The Type C leg with roll-yaw-pitch joints.

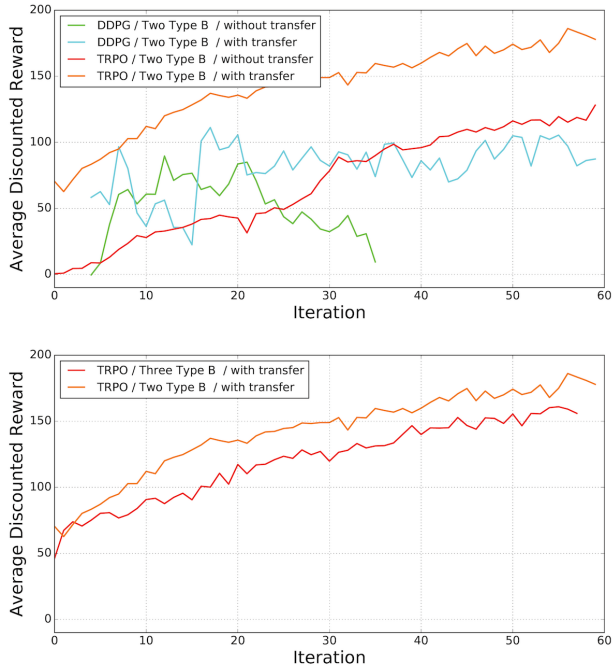

Fig. 7. Learning curves of multi-legged locomotion. (Top) Comparison between learning with and without transferred initial policy parameters. (Bottom) Comparison between two and three Type B legs.

as its original purpose. However, its local improvement scheme sometimes prevents it from finding the global optimal policies, as we can observe the different final rewards in the case of the type B leg. On the other hand, DDPG is very sample efficient for the Type B leg as shown by the quick learning of effective policies. However, DDPG often falls into local minima and takes long time to escape, such seen in the learning curves for the type A leg. We believe that the roll joint in the type A leg increases the difficulty of the problem because the pitch joint cannot touch the ground at the maximum roll angle.

B. Learning of Multi-legged Locomotion

We also tested the learning framework on multi-legged locomotion with up to three legs. The first task is crawling with the two Type B legs. The top plot in Fig. 7 compares the learning curves for the two algorithms with and without using the transferred policy parameters (Section III-C). As we expected, the learned policy serves as a good initial solution: both algorithms achieve better rewards within the same number of iterations. Interestingly, TRPO with the transferred parameters learns to swing its legs three times per cycle, which was the fastest gait we observed (Fig. 5). For one trial of DDPG, we terminated learning at the 35^{th} iteration because the robot started to flip itself upside down and the current version of the reset device cannot flip it back over. We also tried to train locomotion with the three Type B legs (Fig. 7, Bottom), but the resulting motion does not use the middle leg effectively.

V. Discussion and Future Work

In this paper, we implemented an automated learning environment and applied two state-of-the-art learning algorithms, TRPO and DDPG, for finding crawling gaits of modular legged robot hardware. We implemented a fully automated learning environment that computes the reward from a consumer-grade web camera and resets the robot position using a simple lever mechanism. The experimental results indicated that both TRPO and DDPG can successfully train policies on hardware within three hours. In general, TRPO near-monotonically increases the reward while convergence rates of DDPG vary a lot with respect to the problem. We also demonstrated that learning of multi-legged locomotion can be accelerated by transferring policy parameters learned on a single-legged configuration.

Due to the limited sensing capability, we only conducted experiments on simple open-loop crawling motions. It will be interesting to consider more complex controllers and rewarding terms to achieve more complex behaviors. For instance, we can implement an IMU-based feedback controller to train walking or running motions. Alternatively, we can collect the height of the robot using the depth camera and provide a reward term that encourages the transition from crawling to walking.

Although we demonstrated the reuse of policies by initializing starting policies with the parameters learned for more simple configurations, our investigation on transfer is limited to the similar motions with the same type of legs. One future work is to decompose the motion into hierarchical tasks with different levels of abstraction, such as end-effector manipulation, task-space locomotion, and high level decision. As reported in the work of [7] and [8], hierarchical policy representation may allow more smooth knowledge transfer between various morphologies, by directly replacing the low-level controllers. However, we must be aware of the possibility that artificial selection of policy hierarchy may prevent the robot from utilizing its full capability by excluding locomotion strategies that do not fit the hierarchy.

Both algorithms come with their own limitations: sample inefficiency for TRPO and instability for DDPG. Most notably, DDPG is not robust to noise in the data caused by, for example, invalid contacts due to joint slackness or unexpected motions due to communication errors. For instance, a small amount of noise in a good rowing motion can result in a backward movement that receives a large negative reward. Therefore, the learning algorithm for hardware should be less vulnerable to noisy data to avoid degrading the quality of the final policy.

The automated learning process sometimes generated unexpected behaviors. In one instance, the motion included flips to exploit a bug of the vision tracking system where the robot location is miscalculated when the markers are invisible. Although this behavior did not cause any damage to the robot, unexpected behaviors can be very detrimental especially for legged robots with higher center of mass heights. Therefore, direct learning on hardware may require

conservative algorithms that can ensure safety rather than being sample efficient.

REFERENCES

[1] M. Domnguez, I. Escalante, F. Carrasco-Rueda, C. E Figuerola-Hernndez, M. Ayup, M. Umaa, D. Ramos, A. Gonzlez-Zamora, C. Brizuela, W. Delgado, and J. Pacheco-Esquivel, "Losing legs and walking hard: Effects of autotomy and different substrates in the locomotion of harvestmen in the genus prionostemma," vol. 44, pp. 76–82, 04 2016.

[2] S. Kalouche, D. Rollinson, and H. Choset, "Modularity for maximum mobility and manipulation: Control of a reconfigurable legged robot with series-elastic actuators," in *Safety, Security, and Rescue Robotics (SSRR), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 1–8.

[3] J. Kim, A. Alspach, and K. Yamane, "Snapbot: a reconfigurable legged robot," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017.

[4] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *arXiv preprint arXiv:1504.00702*, 2015.

[6] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3389–3396.

[7] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver, "Learning and transfer of modulated locomotor controllers," *arXiv preprint arXiv:1610.05182*, 2016.

[8] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

[9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] M. Yim, W.-M. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. S. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 43–52, 2007.

[12] M. Yim, P. White, M. Park, and J. Sastra, "Modular self-reconfigurable robots," in *Encyclopedia of complexity and systems science*. Springer, 2009, pp. 5618–5631.

[13] J. W. Romanishin, K. Gilpin, and D. Rus, "M-blocks: Momentum-driven, magnetic modular robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 4288–4295.

[14] MOSS, "http://www.modrobotics.com/moss/," 2016.

[15] TOIO, "https://first-flight.sony.com/pj/toio," 2017.

[16] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.

[17] K. Wampler, Z. Popović, and J. Popović, "Generalizing locomotion style to new animals with inverse optimal regression," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 49, 2014.

[18] S. Levine and V. Koltun, "Learning complex neural network policies with trajectory optimization." in *ICML*, 2014, pp. 829–837.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[20] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 387–395.

[21] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[22] J. Tan, Z. Xie, B. Boots, and C. K. Liu, "Simulation-based design of dynamic controllers for humanoid balancing," in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 2729–2736.

[23] A. Yamaguchi, J. Takamatsu, and T. Ogasawara, "Dcob: Action space for reinforcement learning of high dof robots," *Autonomous Robots*, vol. 34, no. 4, pp. 327–346, 2013.

[24] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Machine learning*, vol. 23, no. 2, pp. 279–303, 1996.

[25] K. S. Luck, J. Campbell, M. A. Jansen, D. M. Aukes, and H. B. Amor, "From the lab to the desert: Fast prototyping and learning of robot locomotion," *arXiv preprint arXiv:1706.01977*, 2017.

[26] S. Amarjyoti, "Deep reinforcement learning for robotic manipulation-the state of the art," *arXiv preprint arXiv:1701.08878*, 2017.