

Using Deep Reinforcement Learning to Learn High-Level Policies on the ATRIAS Biped

Tianyu Li¹, Akshara Rai², Hartmut Geyer², Christopher G. Atkeson²

Abstract—Learning controllers for bipedal robots is a challenging problem, often requiring expert knowledge and extensive tuning of parameters that vary in different situations. Recently, deep reinforcement learning has shown promise at automatically learning controllers for complex systems in simulation. This has been followed by a push towards learning controllers that can be transferred between simulation and hardware, primarily with the use of domain randomization. However, domain randomization can make the problem of finding stable controllers even more challenging, especially for underactuated bipedal robots. In this work, we explore whether policies learned in simulation can be transferred to hardware with the use of high-fidelity simulators and structured controllers. We learn a neural network policy which is a part of a more structured controller. While the neural network is learned in simulation, the rest of the controller stays fixed, and can be tuned by the expert as needed. We show that using this approach can greatly speed up the rate of learning in simulation, as well as enable transfer of policies between simulation and hardware. We present our results on an ATRIAS robot and explore the effect of action spaces and cost functions on the rate of transfer between simulation and hardware. Our results show that structured policies can indeed be learned in simulation and implemented on hardware successfully. This has several advantages, as the structure preserves the intuitive nature of the policy, and the neural network improves the performance of the hand-designed policy. In this way, we propose a way of using neural networks to improve expert designed controllers, while maintaining ease of understanding.

I. INTRODUCTION

Reinforcement learning is a continuously learning and adapting framework that can generalize to multiple robots and task scenarios. Such a framework is essential for moving towards autonomous agents that can respond to changes in their environment and learn to perform well on new tasks. This potential has sparked great interest in studying reinforcement learning approaches on complex, real world problems. Recently deep reinforcement learning (DRL) has achieved very impressive results and beyond human performance on several complex long-horizon games such as AlphaGo [1] and even starting with no human input AlphaGoZero[2]. Similar progress has been achieved in the domain of continuous control too, dealing with very high dimensional state and action spaces. For example, Deep Deterministic Policy Gradients (DDPG) [3] and Trust Region Policy Optimization (TRPO) [4] can solve several control challenges in the Mujoco simulation environment [5].

¹Mechanical Engineering, Carnegie Mellon University, USA, tli3@andrew.cmu.edu

²Robotics Institute, School of Computer Science, Carnegie Mellon University, USA, arai@andrew.cmu.edu, hgeyer@cs.cmu.edu, cga@cs.cmu.edu

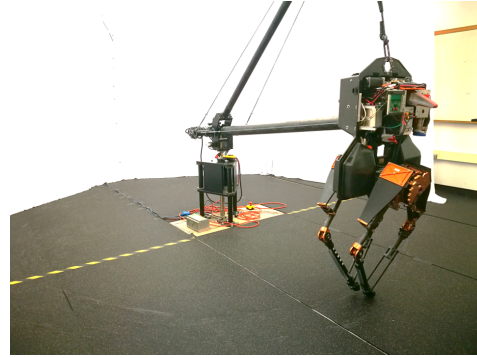


Fig. 1: Our test platform is CMU’s ATRIAS robot in a planar setup.

[6] show that a variant of the Proximal Policy Optimization (PPO) [7] can learn to control a very high degree of freedom humanoid robot in Mujoco, even in very challenging environments. Similarly, [8] use a variant of DDPG to learn to do dexterous manipulation with a high degree of freedom hand in simulation.

While simulation results for deep RL are very impressive, the simulation-hardware gap makes most controllers learned in simulation unsuitable to be implemented on hardware. This is partly because simulations are not perfect representations of real systems, but also because learning approaches often tend to exploit simulation inaccuracies to achieve better performance. Recent work learns parameters of expert controllers on hardware sample-efficiently, for example [9] and [10] use Bayesian Optimization. While these are promising, and robust even to hardware damage, they are limited by the controllers they can represent. On the other hand, high-dimensional neural network policies can approximate arbitrarily complex controllers, but too expensive to optimize on hardware. One approach to learn policies in simulation and deploy on hardware is domain randomization [11]. It can be used to learn robust neural network policies in simulation by applying random perturbations to the dynamics and other properties of the simulator. This approach has been applied to manipulation problems [12], as well as quadrupedal locomotion problems [13]. However, typical domain randomization can lead to controllers that perform worse than controllers trained without randomization [13], as well as make the learning problem much harder [14] taking over 100 years of simulation time to learn successful policies.

Domain randomization from scratch is especially challenging for under-actuated bipedal robots, as the basin of stability

around a controller is typically very small. As a result, it is very difficult to randomly explore the space of controllers to find successful controllers that stabilize a wide range of dynamic models, and other disturbances. Moreover, the learned controllers would typically be quasi-static in nature, similar to [11] in behavior, which is not very impressive in performance.

In this work, we study if a high-fidelity simulator and structured neural network controllers can eliminate the need for domain randomization in learning controllers for bipedal robots. We use a high-fidelity simulation of the ATRIAS robot [15] with ground height disturbances to learn a walking controller in simulation. Then we test these controllers on an actual bipedal robot and evaluate if they are capable of realizing a stable walking gait on hardware.

We experiment with different controller structures, and study their effect on transfer from simulation to hardware, without any domain randomization. We show that structured neural network controllers have a fast training rate in simulation as well as higher rate of transfer to hardware. The structure also gives the user the power to modify the controller, if required, without having to re-train the neural network from scratch.

The main contributions of this paper are as follows:

- 1) We demonstrate a successful neural network policy learned using deep reinforcement learning on a bipedal robot hardware
- 2) We study the effect of action space on the success of transfer between simulation and hardware
- 3) We present a way of using deep reinforcement learning and neural networks to improve user-designed policies, while maintaining their intuitive structure

II. LEARNING LOCOMOTION CONTROLLERS

In this section, we give a brief introduction of deep reinforcement learning and explain how we train our neural network policies in simulation.

A. Background

We formulate our locomotion control problem as a Markov Decision Process (MDP) which can be represented as a tuple: $M = \{S, A, r, \tau, T\}$. Here $S \subseteq \mathbb{R}^n$ is the set of states, $A \subseteq \mathbb{R}^m$ is the set of actions, $r : S \times A \rightarrow \mathbb{R}$ is the reward function, $\tau : S \times A \rightarrow S$ is the transition function which is robot's dynamics in our setting, T is the maximum episode length. We wish to find a policy $\pi : S \rightarrow A$ that can maximize the expected sum of the reward over a trajectory:

$$\eta(\pi) = \mathbb{E}_{\pi} \left[\sum_{i=1}^T r_i \right]$$

We use an actor-critic framework to solve this problem, similar to [8]. The policy is modeled using a neural network that takes as input the partial state of the robot and outputs desired control actions. A second neural network models the value function of the robot, which is the expected total reward from the current state.

B. Imitation Learning

Exploration is one of the challenges in RL methods; we need to try different actions in order to learn from and learn about their consequences. If we start from nothing, the learning process could be very slow [16] since the algorithms put a lot of effort on exploring state-action space. One way to reduce this issue is using behavior cloning [17]. Behavior cloning learns a policy over state-action pairs from expert trajectory [18] by finding the policy parameters θ that solve the following maximum-likelihood optimization problem [16]:

$$\underset{\theta}{\text{maximize}} \sum_{(s, a^*) \in \rho_D} \ln \pi_{\theta}(a^* | s) \quad (1)$$

where ρ_D indicates the expert's trajectory distribution, with a^* is actions that expert took, and s are the states visited.

Similarly, rewards accumulated during the expert's demonstrations can be used to pre-train a Q-function by minimizing TD error [19]. As Q-function is the guidance for policy update, a well pre-trained Q-function benefits actor training as well.

In practice, imitation learning does not guarantee that the cloned policy can work as good as the expert due to compounding error caused by distributional shift in states seen during training and testing [16], [20]. Moreover, the expert controller might not perform sufficiently well in settings where it is difficult to design a controller. For example, while our expert controller is very stable on flat ground walking, rough ground seems to present a challenge. This means that the learned policy actually has to be updated to perform well in this situation. Nevertheless, the expert is a good initialization for the policy and makes the learning process much faster.

C. Proximal Policy Optimization (PPO)

We used PPO [6], [7] as our deep RL algorithm. PPO is a stable on-policy method that uses importance sampling and a clipped objective function to update policy parameters. As a result, the updated policy is close to the initializing policy and the training is stable.

III. CONTROLLERS DESCRIPTION

In this section, we first describe a feedback based reactive policy which is also our expert controller, used for initializing our neural network policies. Then we introduce two types of neural network controllers using a similar state and action representation as the feedback based reactive policy. The first neural network controller has a general architecture, with little structure, while the second has a very similar structure to the expert.

A. Feedback Based Reactive Policy (Expert Controller)

This feedback based reactive policy [21], is designed for controlling the CoM height and torso pitch in stance, and the

leg placement location in swing:

$$F_x = K_{pt}(\theta_{des} - \theta) + K_{dt}(-\dot{\theta}) \quad (2)$$

$$F_z = K_{pz}(z_{des} - z) + K_{dz}(-\dot{z}) \quad (3)$$

$$x_p = k(v_{act} - v_{tgt}) + 0.5 \cdot v \cdot T \quad (4)$$

Here, F_x is the desired horizontal ground reaction force (GRF), K_{pt} and K_{dt} are the feedback gains on the torso pitch angle θ and velocity $\dot{\theta}$. F_z is the desired vertical GRF, K_{pz} and K_{dz} are the feedback gains on the CoM vertical height z and velocity \dot{z} . θ_{des} and z_{des} are desired the torso lean and vertical CoM height. x_p is the desired swing leg foot placement location, v_{act} is the current measured horizontal velocity of CoM and v_{tgt} is the target velocity. T is the swing time and the term $0.5 \cdot v \cdot T$ is a feedforward term similar to the Raibert controller [22]. Our controller tries to maintain a fixed CoM height and torso pitch in stance, and regulate a forward velocity through leg placement in swing.

The desired GRFs (F_x, F_z) are then sent to an inverse dynamics model of ATRIAS to generate desired motor torques that realize the GRFs [23]. These torques are tracked with low level motor velocity-based feedback loop that generates the desired current in the motors.

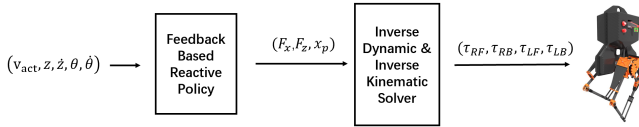


Fig. 2: Pipeline of the expert controller for controlling ATRIAS.

The swing trajectory starts from the current leg position and terminates at the calculated foot position x_{fp} with ground-speed matching. Inverse kinematics then translates this trajectory to desired joint positions and velocities. The joints are controlled by sending a velocity commands to the robot.

This feedback based reactive policy can walk in simulation without any perturbations. However, this controller is not robust enough, since in an environment with ground height disturbances or torque noise it often falls after a few steps. In addition, this controller cannot be directly implemented on hardware. Due to differences between the simulation and hardware such as error in models and parameters, that do not transfer, this controller is not robust enough to overcome these differences. This implies that further training is essential to achieve a better performance. We discuss two different types of controller that can be trained to obtain a better and more robust policy.

B. Neural Network Policy

In our first setting, we create a general neural network policy without much structure. Similar to the expert controller, we design the action space of the neural network to be horizontal and vertical ground reaction forces (GRFs) F_z ,

F_x and the swing leg foot place location x_p . The input space is the state of the CoM: $(x, v_{act}, z, \dot{z}, \theta, \dot{\theta})$. This means that the structure of the neural network policy becomes:

$$\pi(x, v_{act}, z, \dot{z}, \theta, \dot{\theta})_{NN} \rightarrow (F_z, F_x, x_p) \quad (5)$$

The stance and swing control pipeline is similar to the expert and shown in Figure 3.

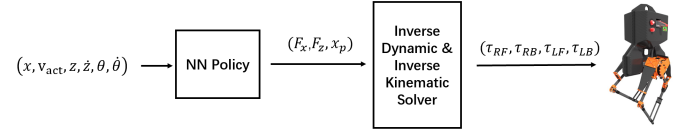


Fig. 3: Pipeline of our first method for controlling ATRIAS. Here τ_{RF} , τ_{RB} , τ_{LF} , τ_{LB} are torques in the right leg's front and back and left leg's front and back motors.

While this policy still has the information about dynamics constraints of the robot through the inverse dynamics and inverse kinematics blocks, it loses the well-defined structure of the expert. This means that the output of the neural network is unpredictable, and the user does not have a lot of control over the behavior of the robot. Roboticists typically prefer controllers that can be understood and predictable, as well as can be modified if needed. In our second controller, we try to emulate this shared autonomy, which lets the user control the overall behavior of the robot, and the neural network helps improve the user-designed policy through deep RL.

C. Neural Network in the Heuristic Policy

In our second setting, instead of directly predicting the vertical and horizontal ground reaction force (F_z, F_x, x_p) as well as desired swing leg foot place location x_p , we use a neural network as part of the feedback based reactive stepping policy, described in Section III-A. While the original policy has a fixed desired torso lean θ_{des} , desired CoM height z_{des} and fixed structure for x_p , we learn these as a function of the state of the CoM using a neural network. The neural network now takes the state of the CoM as input, and predicts the desired torso pitch, CoM height and an offset on the footstep location.

$$\pi(x, v_{act}, z, \dot{z}, \theta, \dot{\theta})_{HP} \rightarrow (\theta_{NN}, z_{NN}, x_{NN}) \quad (6)$$

When inserted into the expert controller, this gives a structured neural network policy, where neural network outputs are used as part of a heuristic policy. It is worth noting that this policy is capable of generating the same outputs as the general neural network policy. For example, for any desired F_x profile, it is always possible to find a θ_{NN} given K_{pt} , K_{dt} , θ and $\dot{\theta}$. The pipeline is illustrated in Figure 4.

$$F_x = K_{pt}(\theta_{NN} - \theta) + K_{dt}(-\dot{\theta}) \quad (7)$$

$$F_z = K_{pz}(z_{NN} - z) + K_{dz}(-\dot{z}) \quad (8)$$

$$x_p = k(v_{act} - v_{tgt}) + 0.5 \cdot v \cdot T + x_{NN} \quad (9)$$

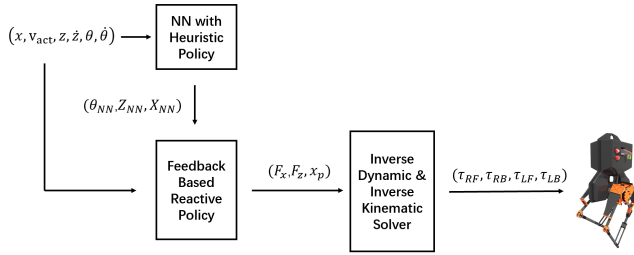


Fig. 4: Pipeline of our second method using a heuristic policy and neural network for controlling ATRIAS.

There are several benefits to having such structure in our policies. First and foremost, the user can predict the behavior and understand the outputs of this policy. This is important for policies implemented on robots to ensure safety of the robot and its environment. Moreover, if the user wants to test a slightly different setting than the simulation, she can easily tune other parameters of this policy, keeping the neural network fixed. Lastly, if the policy fails on hardware, it is easy for the user to understand why that might be, and even possible to fix other parts of the controller without re-training the neural network.

IV. EXPERIMENT

In this section we describe our experimental platform - the ATRIAS robot, followed by our simulation and hardware experiments. Our experiments compare a general neural network policy with a more structured policy, where the neural network helps modulate an expert controller. We compare both policies in simulation and hardware.

A. ATRIAS robot

The ATRIAS robot (Figure.1) is our test bipedal robot platform. It weights about 64kg, with most of its mass concentrated around its torso, and its rotational inertia about its center of mass (CoM) is about 2.2kgm^2 . In this work, we only focus on planar walking, enforced with the help of a boom. Details of the robot and our simulator can be found in [15]. The simulator is designed in MATLAB Simulink environment and is high-fidelity. It has non-linear contact models, as described in [15] for realistic contact forces, as well as detailed actuator dynamics, like spring deflections and gear dynamics.

B. SIMULATION EXPERIMENTS

We trained our neural network controllers in simulation before implementing on hardware. This is because we use Proximal Policy Optimization (PPO) as our learning algorithm. PPO needs to collect a large amount of data points at each iteration. When our current policy is not good enough, we would not be able to collect much data since the robot might fall at the very beginning of experiment. In order to collect enough number of data points, we need to do a large number of trials, making it near impossible to train on hardware. However, since we train our policies

in simulation, the resulting controllers might work well in simulation environment but when implemented on hardware, they might fall. In order to overcome this issue, we add ground height disturbances to learn robust controllers in simulation.

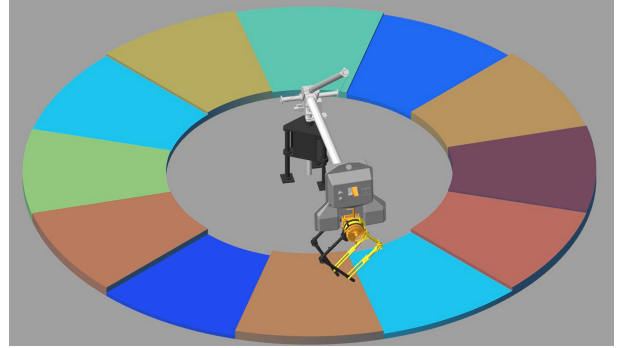


Fig. 5: Simulation of ATRIAS with ground height disturbance. Different colors indicate different ground height disturbances. Maximum ground height disturbance is ± 10 cm.

The reward function is defined focusing on matching the desired walking speed and preventing large angular velocity of the torso. We also add a large penalty if the controller falls:

$$\text{reward}_t = \begin{cases} -C_1 \cdot (v_{act,t} - v_{tgt})^2 - C_2 \cdot (\dot{\theta}_t)^2 + 1, & \text{if walk} \\ -C_3 * T_{sim}, & \text{if fall} \end{cases} \quad (10)$$

where $v_{act,t}$ is the vector is the actual forward velocity of the robot at time t , v_{tgt} is the target velocity, $\dot{\theta}_t$ is the angular velocity of torso pitch, T_{sim} is the simulation time, C_1, C_2, C_3 are positive fixed parameters. This kind of cost function is similar to prior work on optimizing locomotion controllers, and it can easily distinguish points that walk from points that fall. The longer the controller can survive, the closer to the target speed, the less torso pitch the higher total reward the controller can get. In this setting, $C_1 = 1$, $C_2 = 0.3$, $C_3 = 0.01$. In our observation, often the highest scoring controllers in simulation tend to exploit simulation inaccuracies. For example, some controllers tend to jerk the torso to achieve speeds closer to the target speed. While these perform well in simulation, they tend to fail on hardware.

1) *Neural Network Policy*: The first set of simulation experiments were with the Neural Network Policy (NN Policy), described in Section III-B. The first step for Neural Network Policy was behavior cloning. After initialization, our NN policy could walk on flat ground, but it had difficulties with ground height disturbances. This was unsurprising because the 'expert controller' also cannot walk on ground with ground height disturbance. Hence, we had to train the policy further to achieve acceptable performance.

We trained 10 different NN policies, the rewards for which are shown on Figure 8. The rewards shown is the averaged reward of one iteration which consisted on 30,000 data points. As shown in the plot, the reward keeps growing as

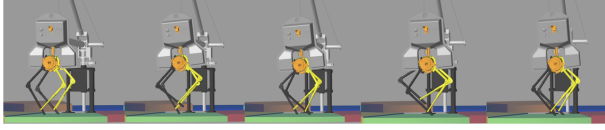


Fig. 6: Sequence of frames from a single walking cycle (left to right).

training goes on, starting from the average cost of the expert. This shows that PPO achieves a very stable training with little to no forgetting of the initial expert.

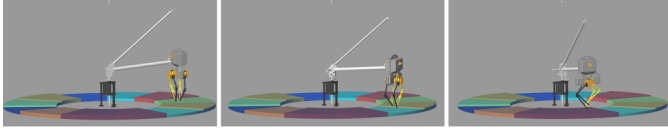


Fig. 7: A time lapse of Neural Network Policy walking around the boom with ground height disturbance in simulation.

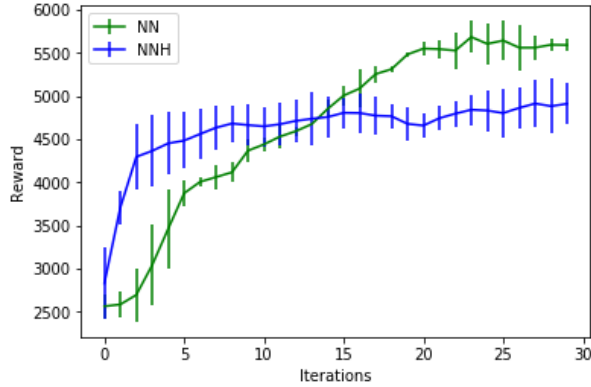


Fig. 8: Training plot of Neural Network Policy (green) and Neural Network with Heuristic Policy (blue). Each of them is averaged of 5 trials data. In our training, we collected 30000 data in one iteration. For each simulation episode, we simulated 10 seconds if the controller keeps walking, and in simulation our time step was 0.001s. Thus, we could collect no more than 10000 data in a single episode, which also means in one iteration we needed data at least from 3 different episodes. By collecting data from more than one episodes, we could average the total rewards to determine the performance of the controller, also eliminating ‘lucky’ runs.

The NN policy achieved good reward improvement compared to the initialization. After training, the NN policy could walk with ground height disturbance with a reasonable success rate. Figure 6 and Figure 7 shows the NN policy walking with ground height disturbance.

2) *Neural network with heuristic policy:* The second set of simulation experiment uses a neural network in a heuristic policy, described in section III-C. By initializing this policy to the target values of the ‘expert controller’, the initial policy

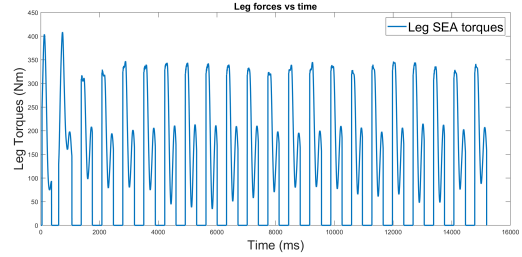


Fig. 9: A plot of the stance leg SEA torques measured during one run of the NN policy on hardware. The torques are 0 during swing and follow a double hump pattern in stance. Note the high initial torques in the first two steps as compared to the rest of the walking cycle.

was able to walk on flat ground but cannot walk on rough ground.

The training plot of Heuristic Policy is shown on Figure 8. As shown on the plot, the Heuristic Policy reward improves faster than the NN policy but reaches its peak in very few iterations. It then remains at that reward for the rest of iterations. The initial fast improvement is expected because as the heuristic policy uses the structure of the feedback-based reactive stepping policy, the search for reasonable parameters is simple. Hence, walking controllers are found much faster. However, the same structure also imposes a strong bias on the search, and as a result, it is hard to find controllers that can perform as well as the pure NN controller in simulation. However, as we describe later, even though the best reward achieved by Heuristic controllers is lower, the rate of transfer to hardware is much higher than the NN policy.

C. HARDWARE EXPERIMENTS

We tested 5 NN policies and 5 heuristic policies in our first set of hardware experiment, with the reward function described in Function 10.

There is a large mismatch between simulation and hardware starting conditions for our system. Starting from rest is a significant challenge for bipedal robots, especially underactuated robots like ATRIAS. ATRIAS is incapable of balancing by standing in place, and needs to continue stepping to stay upright. Typically, a start-up routine is designed for these robots to reach some initial walking gait, and then the controller to be tested is initiated. For example [24] start by executing a stepping motion in air, and a human holds on to the robot as its lowered onto the ground. We train our controllers to start from rest in both simulation and hardware, to avoid having to design a start-up routine. However, our simulation leg is not initially loaded, so the simulation has to make an initial push to keep the CoM from going too low. On the other hand, on hardware the robot starts in position control in the air and then is lowered. So as stance starts, there is already sufficient force in the leg, and the initial push (from simulation) makes the robot leave ground. Such a controller falls on hardware. This behavior is

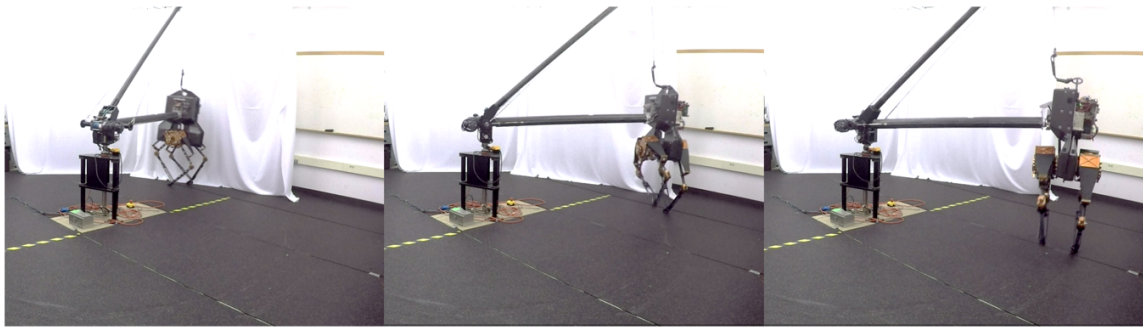


Fig. 10: A time lapse of ATRIAS walking around the boom during a run of a NN with heuristic policy.

illustrated in Figure 9 where the first two steps on hardware experience higher leg forces than the rest of the steps.

1) *Neural network policy*: 2 out of 5 of the NN policy could walk on hardware. But all of the NN policies shared the same issue that at the beginning of walking they would jump up. This behavior can also be seen in our simulation. The initializing ‘expert controller’ has a jump up behavior at the beginning of walking, and our NN policy also inherits this behavior. In simulation, this behavior would not lead to falling, but on hardware, ATRIAS would fall as soon as it tries to start walking because of the initial jump. On hardware, 2 out of 5 controllers could overcome the initial disturbance and then started normal walking pattern. 3 out of 5 controllers, however could not recover from this disturbance. This led to a 40% rate of transfer of learned policies between simulation and hardware.

2) *Neural network with heuristic policy*: 4 out of 5 controllers trained with neural network as part of the heuristic structure were able to walk on hardware. There are a few reasons for the success of the NN with heuristic structure policy, over the pure NN policy. Firstly, the initial discrepancy between simulation and hardware leg force is already compensated for by the heuristic structure. In the simulation, the NN predicts a desired height, and the feedback on this desired height results in a high leg force, as the actual height of the robot is lower. On hardware, the actual height of the robot is close to this desired height, so it automatically reduces the initial leg force. Unlike pure NN policy, where the forces were high on both simulation and hardware, now the initial forces are only high when the actual state of the robot is different from the desired. This structure in our policy, hence, makes the generalization from simulation to hardware very simple and intuitive.

Secondly, since the structure of the policy is readable by an expert, the expert can still edit the policy after being trained in simulation. For example, if the NN was trained on a lower velocity but the expert wanted to run the robot on a higher velocity, the heuristic structure allows to directly edit this target. Similarly, we observed that some of our controllers were falling because of speeding, and we increased the feedback gain k on velocity feedback, described in Equation 9. This modification significantly improved the success rate of policies learned in simulation on hardware.

These experiments highlight the advantages of structure in learned policies. While the NN helped us improve our heuristic policy, as shown in Figure 8, by learning a state-dependant desired height, pitch and footstep location, it also kept the intuitive structure of the heuristic. So, it was able to better reject disturbances between simulation and hardware, as well as be modified for slightly different test situations, without needing re-learning. This allowed a 80% rate of transfer of learned policies between simulation and hardware.

V. CONCLUSIONS AND DISCUSSION

In this work, we used deep reinforcement learning to learn two neural network policies to control the ATRIAS biped in simulation, and study their effectiveness at transferring to hardware. One of the policies uses a general neural network, while the second builds on the structure of a heuristic policy. We show that introducing structure into neural network policies can vastly improve the transfer rate of policies between simulation and hardware. A structured policy can guarantee the safety of the robot, as well as, allow a user to intervene if the controller fails. In our experiments, such a controller led to a 80% rate of transfer between simulation and hardware, as compared to 20% for a traditional NN policy. These results show that incorporating neural networks into heuristic policies can help improve the performance of the policy, and increase the rate of transfer of NN policies between simulation and hardware.

In our hardware setup, the initial condition in simulation and hardware was very different, which led to a lot of controllers failing on hardware. This is a common problem in locomotion systems, especially bipeds. One way of learning more robust policies to such disturbances can be to use cost functions that encourage conservative controllers. We conducted such experiments with a cost that penalized high torques and found that the transfer improved to a 100% for NN policy with heuristic structure. In general, it is worth studying control frameworks that can switch between conservative and high performing controllers, based on the mismatch between simulation and hardware. For example, we might want to start with a conservative controller, and then transition to a high-performance controller once we have achieved a cyclic gait. We leave this for future work.

REFERENCES

- [1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, p. 484, 2016.
- [2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, p. 354, 2017.
- [3] 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.
- [4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International Conference on Machine Learning*, 2015, pp. 1889–1897.
- [5] 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*. IEEE, 2012, pp. 5026–5033.
- [6] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Eslami, M. Riedmiller *et al.*, “Emergence of locomotion behaviours in rich environments,” *arXiv preprint arXiv:1707.02286*, 2017.
- [7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [8] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, “Towards generalization and simplicity in continuous control,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6550–6561.
- [9] R. Antonova, A. Rai, and C. G. Atkeson, “Deep kernels for optimizing locomotion controllers,” in *Conference on Robot Learning*, 2017, pp. 47–56.
- [10] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, “Robots that can adapt like animals,” *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.
- [11] I. Mordatch, K. Lowrey, and E. Todorov, “Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids,” in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5307–5314.
- [12] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Sim-to-real transfer of robotic control with dynamics randomization,” *arXiv preprint arXiv:1710.06537*, 2017.
- [13] 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.
- [14] OpenAI, :, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, “Learning Dexterous In-Hand Manipulation,” *ArXiv e-prints*, Aug. 2018.
- [15] W. C. Martin, A. Wu, and H. Geyer, “Robust spring mass model running for a physical bipedal robot,” in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 6307–6312.
- [16] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, “Learning complex dexterous manipulation with deep reinforcement learning and demonstrations,” *arXiv preprint arXiv:1709.10087*, 2017.
- [17] D. A. Pomerleau, “Efficient training of artificial neural networks for autonomous navigation,” *Neural Computation*, vol. 3, no. 1, pp. 88–97, 1991.
- [18] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.
- [19] A. Sendonaris and C. G. Dulac-Arnold, “Learning from demonstrations for real world reinforcement learning,” *arXiv preprint arXiv:1704.03732*, 2017.
- [20] S. Ross and D. Bagnell, “Efficient reductions for imitation learning,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 661–668.
- [21] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. G. Atkeson, “Bayesian optimization using domain knowledge on the atrias biped,” *arXiv preprint arXiv:1709.06047*, 2017.
- [22] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.
- [23] A. Wu and H. Geyer, “Highly robust running of articulated bipeds in unobserved terrain,” in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 2558–2565.
- [24] C. Hubicki, J. Grimes, M. Jones, D. Renjewski, A. Spröwitz, A. Abate, and J. Hurst, “Atrias: Design and validation of a tether-free 3d-capable spring-mass bipedal robot,” *The International Journal of Robotics Research*, vol. 35, no. 12, pp. 1497–1521, 2016.