Hierarchical Reinforcement Learning of Locomotion Policies in Response to Approaching Objects: A Preliminary Study

Shangqun Yu
Department of Computer Science
Brown University
Providence, RI 02906
syu68@cs.brown.edu

Sreehari Rammohan
Department of Computer Science
Brown University
Providence, RI 02906
sreehari@brown.edu

Kaiyu Zheng
Department of Computer Science
Brown University
Providence, RI 02906
kzheng10@cs.brown.edu

George Konidaris
Department of Computer Science
Brown University
Providence, RI 02906
gdk@cs.brown.edu

Abstract

Animals such as rabbits and birds can instantly generate locomotion behavior in reaction to a dynamic, approaching object, such as a person or a rock, despite having possibly never seen the object before and having limited perception of the object's properties. Recently, deep reinforcement learning has enabled complex kinematic systems such as humanoid robots to successfully move from point A to point B. Inspired by the observation of the innate reactive behavior of animals in nature, we hope to extend this progress in robot locomotion to settings where external, dynamic objects are involved whose properties are partially observable to the robot. As a first step toward this goal, we build a simulation environment in MuJoCo where a legged robot must avoid getting hit by a ball moving toward it. We explore whether prior locomotion experiences that animals typically possess benefit the learning of a reactive control policy under a proposed hierarchical reinforcement learning framework. Preliminary results support the claim that the learning becomes more efficient using this hierarchical reinforcement learning method, even when partial observability (radius-based object visibility) is taken into account.

Keywords: locomotion, reactive control, hierarchical reinforcement learning

1 Introduction

Animals have the ability to command their body—a complex kinematic system—quickly in response to environment stimuli [Pavlov, 2010]. When being approached by a person from an unexpected direction, a rabbit can immediately run away, even if the rabbit has never seen a person before. When being thrown a rock, a bird can very quickly fly away to avoid being hit, even if the rock is moving very fast and the bird does not notice until near collision.

Our work draws a connection to both this observation of nature and the progress of deep reinforcement learning (RL), where we noticed remarkable progress in using deep RL for locomotion [Peng et al., 2016, 2020; Tassa et al., 2018]. Many of the existing works in deep RL for locomotion control only consider controlling the agent from point to point. Works that do consider controlling an external object (e.g. dribbling) assume full observability of the object's properties permitted for the application of animation [Peng et al., 2017]. Other works present empirical evidence for the viability of training end-to-end locomotion policies directly from pixel input [Tassa et al., 2018], but they typically use third-person images with high sample complexity, which creates challenges for practical implementation. In this work, we focus on learning a reactive locomotion policy, which contrasts the settings in prior works where the agent is typically trained to proactively complete some task.

Our goal is to extend the success of deep reinforcement learning for locomotion control to settings where the agent must react to external objects under partial observability. As a first step, we consider a task where the agent must react to avoid being hit by an approaching ball (the "Dodge Ball Task"). The only reward signal the agent gets is a negative reward when being hit by the ball. Although reinforcement learning has achieved great improvements in domains such as games [Mnih et al., 2015] and continuous control for robotics [Gu et al., 2017], learning a policy in the continuous control setting with sparse rewards is still a major challenge in RL [Li et al., 2019]. Hierarchical Reinforcement Learning (HRL), with its structured policy and decomposition of problems into smaller subproblems [Levy et al., 2019], not only has shown strength under these challenging settings, but also provides a solution to reuse low-level skill modules on different tasks [Li et al., 2019]. Therefore, we investigate the benefits of having prior locomotion experience for this task under an HRL framework. Our preliminary results have shown that a two level feudal hierarchical agent with pre-trained low-level controller can solve the task with high sample efficiency while the end to end agent completely fails to learn with even five times the training samples.

2 Hierarchical Reinforcement Learning for Legged Reactive Control

A legged reactive control task can be formulated as a partially observable sequential decision-making problem. The environment state s can be broken into two parts, that is, $s = [s_o, s_h]$, where s_o represents the fully observable internal state of the agent, and s_h represents the state of the external object hidden to the agent. At each time step, the agent executes an action a to control its joint velocities, and it receives an observation, denoted $z = [s_o, \phi(s_h)]$, as a function of the state as a result of the action. The task is specified via a reward function $R(s)$. We are interested in the standard reinforcement learning objective, where the agent needs to maximize the cumulative reward as it interacts with the environment.

In our preliminary investigation of the reactive control setting, the objective is to minimize the expected number of collisions $\mathbb{E}[n]$ by developing an optimal policy $\pi^*(a|z)$ for an agent in a world with a single object, which takes in the current state, and outputs joint velocities for the robot. To this end, we develop the Dodge Ball Task, shown in the center of Figure 1, implemented in MuJoCo. The agent (an 8 degree of freedom ant) must learn to dodge a ball which continuously re-spawns from different locations before being shot at the agent along a linear path. In the environment, we define a sparse reward function for our task based on whether a collision happens. Negative reward is only assigned when the robot is hit by the projectile. One episode lasts 1000 steps and a new ball is spawned randomly in a position that is 5 meters away from the agent and fired at the agent at a speed of 2 meters/second every 100 steps. An agent following the optimal policy will be able to dodge all balls and thus will have a final cumulative reward close to 0.

$$R(s) = \begin{cases} \text{-5} & \text{agent hit by ball} \\ 0 & \text{otherwise} \end{cases}$$

We propose a two level Feudal Reinforcement Learning agent shown in Figure 2. Feudal Reinforcement Learning (Feudal RL) is a type of Hierarchical Reinforcement Learning where a high-level controller sets a subtask that is executed by a lower-level controller [Dayan and Hinton, 1992; Pateria et al., 2021]. The state space $S = S_h \times S_o$ consists of the positions and velocities of the agent's joints as well as the position and velocity of the projectile ball. Both high-level and low-level controllers use SAC (soft-actor critic) [Haarnoja et al., 2018] as a base learning algorithm for the task objective.

The high-level controller receives the observation $z = [s_o, \phi(s_h)]$ and outputs a subgoal g to the low-level controller. A subgoal is a 2D point with coordinates relative to the agent. The low-level controller then receives this intent along with the fully observable part of the state s_o and outputs an action consisting of joint velocities for the 8-DOF agent.

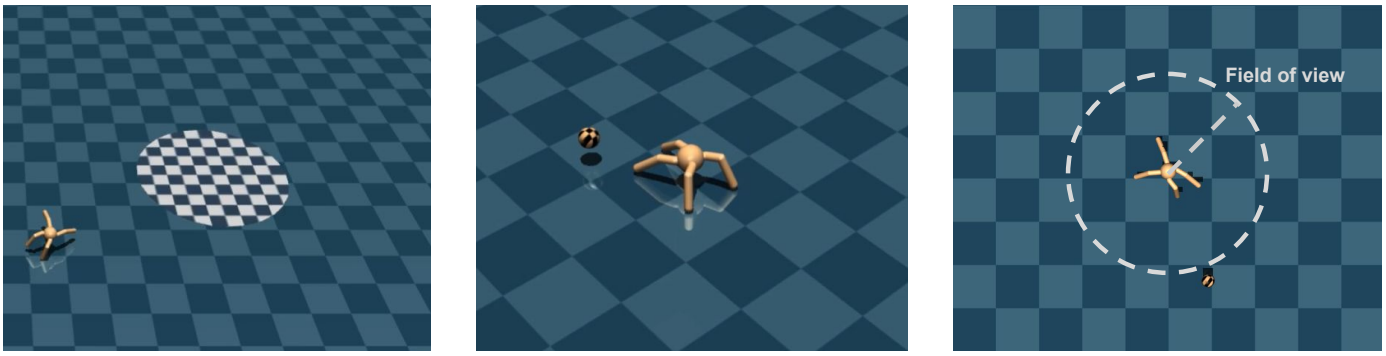

Figure 1: Left: The environment used to train the low-level controller. A target location is defined randomly near the agent (every 100 steps). The agent will gain reward proportional to the distance it traveled toward the goal. Center: In the dodge ball task, a single ball is spawned randomly in the environment (every 100 steps) and shot toward the agent along a linear trajectory. The agent receives a -5 reward every time it gets hit by the ball. Right: in the partially observable version of the dodge ball task, the agent does not receive information about the ball until it is within the field of view (a circle with 4 meter radius).

Similar to how animals possess prior locomotion skills such as running or flying, we first train the low-level controller in a different environment to gain basic locomotion skills for the agent. This environment, shown on the left side of Figure 1, requires the agent to move along a randomly specified direction vector within 300 steps, receiving reward proportional to the distance travelled in this direction. The agent receives an observation consisting of the internal configuration (joint positions and velocities) along with a subgoal g with the normalized direction vector. After training, the low-level controller is able to move the agent forward along an arbitrary direction specified by g.

Next, the low-level controller is "attached" to the high-level controller in the sense that the high-level controller outputs latent intents (similar to a subgoal direction g) which are passed into the low-level controller to compute the primitive action a. Note, because the low-level controller is trained agnostic to the high-level task, this module can be re-used – all that needs to be re-trained is the high-level controller.

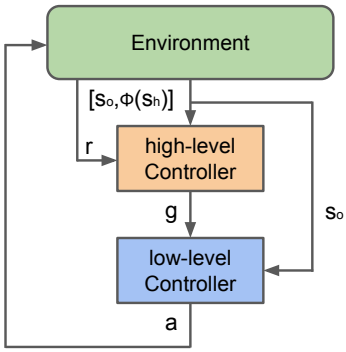

Figure 2: The high-level controller receives a state s from the environment and outputs a subgoal g to the low-level controller consisting of the direction vector it wants the agent to move along. The low-level controller takes in this subgoal and the internal state s_o and outputs a primitive action for execution in the environment.

3 Preliminary Experiments

3.1 Experiment Setup

We evaluate the performance of our algorithm in both fully and partially observable settings (Figure 1). In the fully observable setting, the agent will receive the complete information of the state s, including $[s_o, s_h]$. The full state of the ball s_h is a 6D vector that contains the 3D position (relative to the agent) and velocity of the ball (relative to the world frame). In the partially observable setting, the observation function ϕ_h is defined such that the agent can only "see" the ball once it is within a 4-meter radius (in all other circumstances 0 is populated for the elements corresponding to the obstacle in the low-dimensional state s_h). We use a standard SAC end-to-end trained agent (no low-level controller) as a baseline for comparison. Each agent is trained for 3 seeds (each lasting 2000 epochs on both environments). In each epoch, the agent collects 2000 samples of interaction from the environment and performs 200 gradient update steps.

3.2 Results and Discussion

Our results in Figure 3 show that under sparse reward, feudal HRL agents in both the fully observable and partially observable settings are able to learn reactive behavior well. We observe that the high-level controller learns to assign the direction it wants to go in as a subgoal to the low-level controller, and the low-level controller moves the robot toward the desired direction based on the subgoal, which allows the agent to avoid the ball. It's worth noting that the behavior of the HRL agent is slightly different between the partially observable setting and fully observable setting. Under the fully observable setting, the agent tends to react earlier to the ball when it is approaching compared to the agent under partial observability (Figure 4). This matches our

intuition since under partial observability, the agent can not "see" the ball until it is within 4 meters. As a result, the HRL agent under partial observability converges to a slightly lower average return. End-to-end agents struggle to learn anything meaningful under the sparse reward setting and end up with jittering policies for locomotion after extensive training.

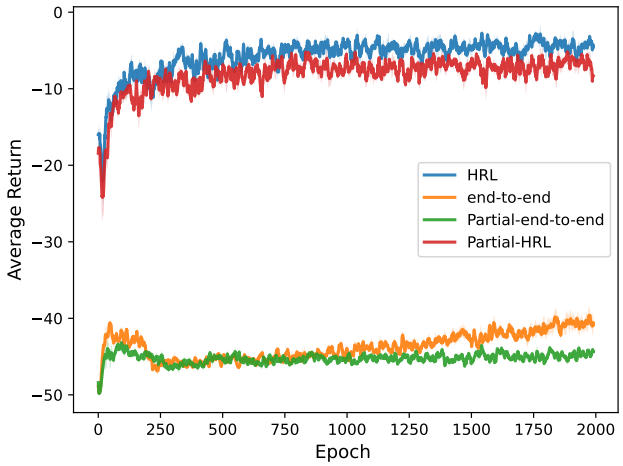

Figure 3: All agents are trained on 3 seeds. The results show that the HRL agents converged within the first 250 epochs to a cumulative reward close to 0, indicating that the agents are able to dodge most balls. Meanwhile, the agents trained end-to-end in both the partially observable and fully observable setting were not able to learn a reactive policy.

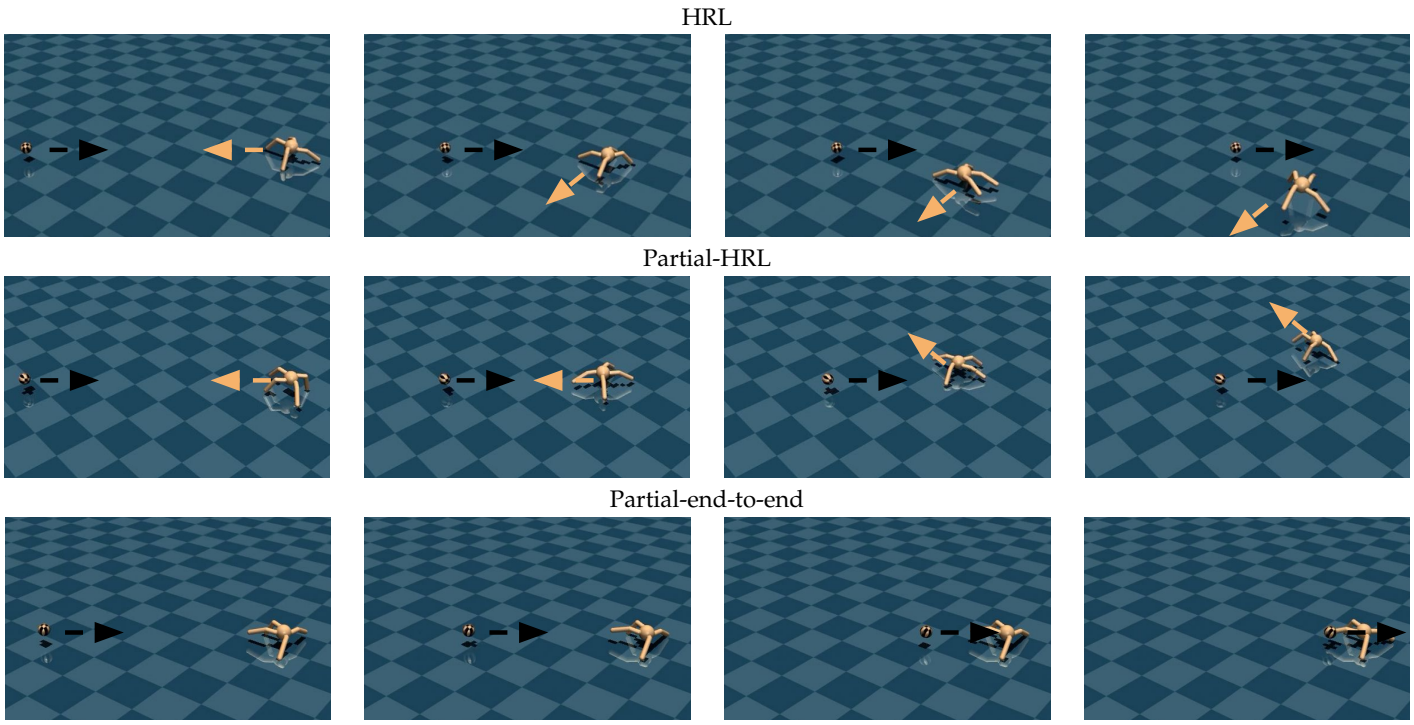

Figure 4: It takes less time for the HRL agent in the fully observable environment (1st row) to react when the ball is approaching compared to the HRL agent in the partially observable environment (2nd row), while the end-to-end agent fails to learn an effective policy in both environments. (3rd row)

3.3 Conclusion and Future Work

To study the task of reactive control, we developed a ball-dodging environment in MuJoCo that involves a subset of the challenges real world robots face such as learning a locomotion policy and acting under partial observability. We also presented a two level feudal hierarchical reinforcement learning framework for the environment and empirically demonstrated significant improvements over an end-to-end trained RL agent. We plan to extend our HRL framework to tasks that involve a variety of kinematic systems with more realistic assumptions of the agent's perception capabilities in the partial observable setting (e.g. optical flow). We also hope to increase the complexity of approaching objects (increasing the number and changing the dynamics) to create a more challenging domain. Eventually, we hope to realize our framework on a real quadrupedal robot.

References

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

Shixiang Shane Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

Andrew Levy, George Dimitri Konidaris, Robert W. Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations (ICLR)*, 2019.

Siyuan Li, Rui Wang, Minxue Tang, and Chongjie Zhang. Hierarchical reinforcement learning with advantage-based auxiliary rewards. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1407–1417, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/nature14236.

Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv.*, 54(5):109:1–109:35, 2021.

P Ivan Pavlov. Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex. *Annals of neurosciences*, 2010.

Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. In *Robotics: Science and Systems*, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.