Cassie Learning Dodging Skills: A Hierarchical Reinforcement Learning Based Approach

Jesus Navarro
Koushil Sreenath, Ed.

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-254
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-254.html

December 14, 2021

Acknowledgement

This research work is funded by the National Science Foundation Graduate Research Fellowship Program. Much of the work of this project was done in collaboration with Zhongyu Li. Additionally, I would like to thank my research advisor Prof. K. Sreenath for his guidance and mentorship.

CASSIE LEARNING DODGING SKILLS: A HIERARCHICAL REINFORCEMENT LEARNING BASED APPROACH

Jesus Navarro

ABSTRACT

Bipedal locomotion presents a challenging set of control tasks that can be decomposed as (1) primitive and (2) task-specific high-level tasks. There exists a diverse set of data-driven and model-based controller optimization methods that enable legged robots to perform well on primitive (low-level) tasks such as walking, standing, and jumping. Learning high-level legged locomotion tasks is generally modeled as a two-layer optimization problem with a high-level and low-level control policy. We propose a similar and simple framework that uses Deep Reinforcement Learning (RL) to learn a high-level task given a fixed and high-performing low-level policy. State-of-the-art architectures are developed as end-to-end frameworks where both the parameters of low-level and high-level policies are optimized jointly during training. Our method decouples the learning procedure of the high and low-level tasks as disjoint optimization problems, and uses a curriculum learning based approach to optimize the high-level task. We demonstrate the learning framework by teaching Cassie, a bipedal robot, to dodge a rolling ball using a jumping and standing primitive controller.

1 INTRODUCTION

Primitive locomotion skills such as jumping, walking, and running are essential skills required for human-sized bipedal robots to interact with the real world. Data-driven approaches, such as evolution strategies and reinforcement learning, enable us to develop task-specific policies that perform well in a simulated environment. Methods such as Domain Randomization [1], enable the transfer of the controllers from their simulated environment to the real world while remaining robust and high-performing. Although traditional model-based approaches for humanoid locomotion also provide feasible and robust controllers, they are often highly constrained, overly-simplified, and require a high-level of parameter tuning [2; 3; 4].

While model-free approaches provide several advantages over model-based controllers, modern frameworks such as Deep Reinforcement Learning (RL) face many pitfalls and disadvantages. Deep Reinforcement Learning algorithms generally use neural networks to model the controller thereby introducing a large number of nested parameters into the system and usually require a lot of data and training. Augmenting the input or action space of the system may be necessary to learn in different environments, however, it requires either restarting the training process or applying some sort of transfer learning and fine tuning which may lead to poor performance. Due to this characteristic of neural networks, using a single policy for learning a hierarchical set of tasks is challenging. Bipedal robotic systems fall in this category where many tasks rely on low-level skills such as walking, jumping, or standing.

To address this problem Hierarchical Reinforcement Learning (HRL) makes use of multiple policy controllers to achieve learning high-level tasks but are typically developed as end-to-end frameworks where both low-level and high-level policies are jointly trained. This paper presents an HRL-inspired training framework that uses a stationary pre-trained low-level policy to learn high-level tasks. Specifically, we train a high-level policy to enable a bipedal robot to learn the high-level task of dodging a rolling ball by deciding between jumping and standing primitive skills. Unlike state-of-the-art HRL methods, our approach mitigates jointly training two policies.

2 Related Work

Hierarchical Reinforcement Learning (HRL) [5; 6; 7; 8] introduced the problem of learning high level and low-level tasks by decomposing the Markov Decision Process (MDP) into smaller sub-problems. More recently, [9] proposed a general two-layered hierarchical architecture where the low-level policies specialize sub-goals discovered and defined by the high-level policies. Hierarchical inspired approaches have demonstrated locomotion of a quadruped robot [10; 11] by representing the behavior of a complex robotic machine as a set of tasks formulated as least squares problems with a variety of priorities for motion, torque, and force optimization. An end-to-end Hierarchical control framework is proposed by [12] for quadruped locomotion over rough terrain. HRL is applied to legged locomotion in [13] to address challenging tasks that require a diverse set of primitive skills. The proposed HRL frameworks decompose the complex quadruped legged locomotion task into a set of high-level and low-level tasks, which are optimized using low-level and high-level policies.

Curriculum Learning [14; 15] (CL) is another method used to learn complex tasks by gradually increasing the complexity of the learning task throughout training. CL has demonstrated improvements across several different applications such as manipulation tasks, navigation, planning, and bipedal locomotion [16; 17; 18; 17; 19]. Early applications of CL for robotics manipulation [20] showed that gradually increasing the difficulty of tasks throughout training improved learning efficiency and overall performance. Similarly, [21] introduces a skill-discovery approach in continuous domains by constructing a chain of tasks that lead to end-of-task reward.

Recently, [22; 23] trained a set of walking-gait policies using a set of reference motions with different walking velocities and used policy distillation to construct a single policy capable of tracking velocity commands. Later, [16] introduced a tailored CL method that increases the learning difficulty using different target velocities with a single control policy.

Although CL and HRL methods have been shown to improve the learning efficiency and performance of tasks that make use of primitive skills, their implementation embed knowledge of the final desired behavior (end task). While knowledge of the end task is usually known, learning a new set of end tasks may require retraining from scratch.

2.1 Contributions

We propose a framework that trains a high-level policy network using reinforcement learning to implicitly learn high-level skills using a high-performing primitive policy for the low-level task. We apply our framework to teach a bipedal robot, Cassie, dodging skills. More specifically, we learn a high-level policy π_H that optimally selects which primitive skills to execute via the low-level policy π_L to avoid colliding with a rolling ball using a simple reward structure.

3 Preliminaries

We introduced the problem of enabling Cassie to learn the high-level task of dodging a rolling ball using primitive skills and discussed related work on Hierarchical Reinforcement Learning and Curriculum Learning. In this section, we will outline our approach by first introducing the Cassie simulation environment. We will then present Reinforcement Learning preliminaries and the Proximal Policy Optimization algorithm used to solve the RL problem.

3.1 Cassie Simulator

Cassie is a high-fidelity bipedal robot developed by Agility Robotics with 20 degrees of freedom and 10 actuators. It is an underactuated robotic system as it contains unactuated spring joints and several other unactuated degrees of freedom. Our methods and experiments are conducted using a simulated environment of Cassie in Mujoco [24].

We use a similar design as [25] to model the RL environment, where the MuJoCo model of Cassie represents the agent that the environment interacts with. The action space $\mathbf{a}_t \in \mathbb{R}^{10}$ specifies the 10 desired motor positions which are used as inputs into another low-level PD controller implemented in MuJoCo. Given a reference trajectory, $\mathbf{s}_t$ specifies the state input at time t of

the policy, which is decomposed as a sequence of observable robot states $q_t \in \mathbb{R}^{20}$, defined as, $\mathbf{s}_t = [q_t \quad q_{t+1} \quad q_{t+3} \quad q_{t+7}]$. We provide future robot states to help the policy infer system dynamics.

3.2 Reinforcement Learning

Reinforcement learning is generally modeled as a Markov Decision Process (MDP) defined as a tuple $\{S, A, p, \gamma, R\}$ representing the state space, action space, dynamic transition function, discount factor, and reward function of the system. Let us define a policy π_θ with parameters θ such that $\pi : \mathbb{S} \rightarrow \mathbb{A}$ where $S \in \mathbb{R}^n$ and $A \in \mathbb{R}^k$. The reward function $R : S \times A \rightarrow \mathbb{R}$ returns a scalar defined as the reward of the agent for taking an action a_t from state s_t at some time t in a specified environment.

The goal of reinforcement learning is to find an optimal policy π such that it maximizes the expected cumulative reward of an agent interacting with its environment, modeled by the MDP. Let us refer to the general objective function as $J(\theta)$, thus the RL framework seeks to maximize $J(\theta)$:

$$\max_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \big[\sum_{t=0}^{T} \gamma^t r_t \big] \tag{1}$$

where $\pi_\theta(\tau)$ represents the trajectory distribution imposed by the parameters θ and policy π_θ. The parameter $\gamma < 1$ is the discount factor and represents the time-value of the reward. There exists a set of novel off-policy and on-policy gradient algorithms for continuous Deep RL tasks [26; 27; 28; 29] that perform well for robotic applications. We make use of the Proximal Policy Optimization (PPO) algorithm [30] in our training framework.

3.2.1 Proximal Policy Optimization

PPO is a model-free on-policy optimization algorithm that uses a stochastic policy whose learned distribution parameters are used to sample actions. Specifically, $\pi_\theta(a_t|s_t)$ is modeled as a Gaussian distribution where the mean and variance of each action are learned. The stochasticity of the policy induces noise into the sampled action and thus results in efficient exploration of the agent throughout training. PPO has been applied to several continuous robotic tasks, including bipedal locomotion [31; 10]. One of the main advantages of PPO compared to other state-of-the-art Deep RL algorithms is relative robustness to hyper-parameter initialization and minimal fine-tuning. The objective function is defined as the following,

$$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t = \big[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \big] \tag{2}$$

where $\hat{A}_t$ is a sample based estimator of the advantage function used in the family of Actor-Critic RL algorithms, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ denotes the probability ratio between the new and old policy at time t. The probability ratio $r_t(\theta)$ measures the deviation of the new policy relative to the state of the previous policy. The clipping operation prevents the algorithm from taking a gradient step that will deviate the policy too far from the previous policy using $r_t(\theta)$.

4 Methods

In the following section, we will detail our proposed training framework and define the high-level action space, input space, and reward formulation of the high-level policy. We also discuss the role of the low-level control policy and the different training setups.

4.1 Training Framework

We define the pre-trained low-level policy as π_L with fixed parameters θ_L and the high-level policy network as π_H with learnable parameters θ_H. Similarly the action output of the low-level and

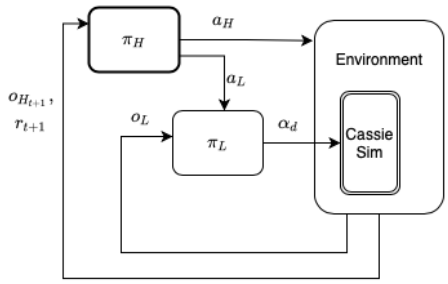

Figure 1: Diagram of our proposed training framework where the low-level policy π_L remains constant throughout training and outputs α_d, the desired joint angle positions of Cassie. The high level policy π_H outputs two sets of actions a_H and a_L where a_L is fed into the low-level policy and a_H is the selected jumping distance threshold. Note that a_L is considered an input into the low-level policy system and not an explicit component of the input observation o_L. The policy π_H receives a reward and new observation o_H from the environment.

high-level policy are denoted as a_L and a_H, respectively. We denote the observations and rewards corresponding to the high-level and low-level policy using the subscripts L and H, respectively. At each time t, we sample an action $a_H \sim \pi_H(a_H|s_H, s_L, \theta_H)$. Similarly, at each time t, we sample $a_L \sim \pi_L(a_L|s_L)$ for the low-level policy. For notation simplicity, the time subscript t is neglected and assumed. The action space of π_H is defined as $a_H = [a_{H_{1:3}} \quad a_{H_4}]$ and is composed of two parts: (1) action values for the high-level task a_{H_4}, and (2) action values used as inputs for the low-level policy $a_{H_{1:3}}$. We apply each action in the environment resulting in new states at time $t+1$, s'_L and s'_H and a reward scalar r_H for the high-level policy π_H. A diagram of our training framework is shown in Figure 1.

4.1.1 High-Level Action Space

The first three values of action $a_H \in \mathbb{R}^4$ specify inputs to the low-level controller which include: (1) reference trajectory selection a_{ref}, (2) desired x velocity, and (3) desired y velocity. Note that for the high-level task of dodging a rolling ball, the reference trajectory selection is composed of either standing or jumping reference motion, thus we can represent the reference selection action value as boolean values using a threshold value of 0. The last action value a_{H4}, represents the relative distance between the ball and Cassie at which the high-level policy should select the jumping reference motion. For clarity, we denote a_{H4} as a_{dist}.

4.1.2 High-Level State Space

The state space s_H is composed of the global position of the ball p_b, Cassie's observable states at time t, q_t, and a history of reference trajectory selections, ball position, and distance thresholds representing one second in simulation time. The frequency of the controller is 30Hz, so the size of each of the history lists is of length 10, thus the values are sampled at every third simulation step.

4.1.3 High-Level Reward Function

We formulate a very simple reward such that it outputs a 0 or 1 at each time step. We define the relative global Euclidean distance between the ball at time t as $d_{rel} = ||\mathbf{p}_b - \mathbf{q}_{b_{[0:3]}}||_2^2$. The selected jumping distance, or distance threshold, and reference trajectory selection of the high-level policy are denoted as a_{dist} and a_{ref}. The reward function is computed as follows:

$$R_h(t+1) = \begin{cases} 0 & d_t \leq a_{dist}, \quad a_{ref} \leq 0 \\ 1 & d_t \leq a_{dist}, \quad a_{ref} \geq 0 \\ 0 & d_t \geq a_{dist}, \quad a_{ref} \geq 0 \\ 1 & d_t \geq a_{dist}, \quad a_{ref} \leq 0 \end{cases}$$

The reward function defines the goal of the policy π_H, which is to select the standing reference trajectory when the relative distance of the ball and Cassie, d_{rel} is greater than the selected jumping distance threshold a_{dist} and select the jumping reference motion when the ball is within the selected distance threshold. We add an additional terminal condition when there is contact between Cassie and the rolling ball. The defined reward function and terminal condition allow the high-level policy to implicitly learn to accomplish the high-level task. The agent's incentive high-level policy will be encouraged to select optimal jumping distance thresholds so that the agent avoids the terminal condition and accumulates a higher episode reward.

We show in simulation that this simple reward formulation enables Cassie to learn dodging skills, but we determine undesired and unstable action output of a_{dist} while Cassie is in the jumping phase, which we discuss in the Conclusion section of the report.

4.2 Low-level Controller Policy

The lower-level control policy used in this paper was developed following a model-free RL framework approach [32] and is trained with Domain Randomization [1] to improve the robustness of the policy to enable sim2real transfer. The approach uses jumping gait reference motion trajectory developed using the Hybrid-Zero-Dynamics approach and is used to teach the RL controller to output the desired motor positions to track the reference trajectory in simulation.

The jumping reference trajectory was modified to encourage Cassie to learn how to jump bilaterally at different velocities and was trained by randomly sampling the desired bilateral velocities. In parallel, the policy learned a nominal standing reference trajectory which enabled Cassie to perform multiple jumps with varying standing times in between jumps.

The parameters of the low-level control policy are not updated throughout our training framework and are run in parallel with the high-level policy. Note that the low-level policy entails an additional state observation of its own used to sample $a_L \sim \pi_L(a_L|s_L)$. Our training framework specifies the inputs to the low-level policy as desired parameters that are implicitly included in the state observation. In other words, the inputs to the low-level policy defined by our training framework $a_{H_{0:3}}$ are not explicitly fed into the state observation of the lower-level control policy but instead are set as defined parameters of the agent.

4.3 Training

In this section, we describe in detail different experimental setups conducted. We trained three different models in simulation:

1. Random ball x position, constant v_x
2. Random ball x position, random v_x
3. Random ball x and y position, random velocity magnitude

We used Open AI's Baseline Proximal Policy Optimization algorithm implementation, an on-policy RL algorithm that performs well for continuous tasks, to train our high-level policy. Both the high-level and low-level policies are running at 30Hz in simulation. The maximum number of steps per episode is set to 1000 simulation steps and defines our maximum episode reward to 1000.

For (1) we set the the initial ball position $\mathbf{p}_0 = [p_x, 0, 0]$ where $p_x \sim \mathcal{U}[3.5, 10]$ (meters) and fix the initial ball velocity, $\mathbf{v}_0 = [1.75, 0, 0]$. For case (2), the ball position is sampled randomly in the same way as (1), but randomly sample $v_x \sim \mathcal{U}[1.75, 10]$ (meters per second) and set $\mathbf{v}_0 = [v_x, 0, 0]$.

Lastly, for setup (3) at the beginning of each training episode, the initial radial ball position and velocity are uniformly sampled, $r_0 \sim \mathcal{U}[3.5, 10]$ and $||\mathbf{v}_0||_2^2 \sim \mathcal{U}[1.75, r_0]$. The initial velocity magnitude of the ball has an upper bound defined as the initial radial position of the ball to avoid infeasible initial conditions such as having a very high-velocity magnitude and small initial radial positions which will likely lead to a collision. We also sample $\phi \sim \mathcal{U}[0, 2\pi]$ to fully define the initial position of the ball. The position is then transformed into Euclidean coordinates to set the position in the simulator.

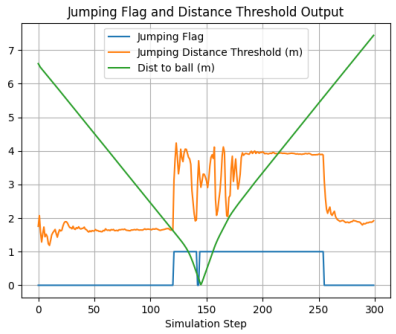

Figure 2: High-level policy action selection plots at each simulation step for the reference selection (jumping flag), distance threshold, and the relative position of the ball and Cassie.

4.3.1 Jumping Distance Threshold

We impose a constraint on the action value a_{dist}, which specifies the jumping distance threshold by scaling the output by 10 meters. Therefore the range of possible distance thresholds is between 0 and 10 meters. We clip the lower limit from 0 meters to 1 meter since small jumping distance thresholds are not feasible. Table 1 displays the range and definitions of the randomized parameters during the start of each training episode.

4.3.2 Selecting Reference Motion of Low-Level Policy

Since the high-level and low-level policies are running in parallel, we face the issue of the high-level policy selecting the standing reference shortly after initializing the jumping phase early in training. Since its physically infeasible to switch between jumping and standing gaits at high frequencies, the jumping command is latched onto the lower-level policy until it has terminated jumping. However, the reward signal considers the raw output and is used to optimize the high-level policy to select the correct desired reference trajectory at each simulation step.

5 Experiment Results

In this section we illustrate the results of the training setups discussed in the Methods section. We include snapshot sequences of Cassie dodging a rolling ball and include action behavior plots for a set of initial conditions. We vary the initial positions and velocity magnitude of the rolling ball.

5.1 Non-Random Velocity Model

Figure 3 demonstrates a snapshot sequence of simulations using our trained high-level policy network with no randomization of the initial ball velocity. Note that in this training setup, we also fix the desired y velocity of Cassie to be 0 throughout training and testing. The left and right snapshot of Fig. 3 illustrates the high-level policy outputting the standing reference position. Conversely, the 3 inner snapshots show instances of our high-level policy selecting the jumping reference trajectory and successfully jumping over the ball.

We run an additional simulation test for 300 total sim steps and plot the reference trajectory selection a_{ref}, jumping distance threshold a_{dist}, and distance between the ball and Cassie d_{rel} which are shown in Figure 2. Given the reward function description we define the desired output of a_{ref} as,

$$a_{ref} = \begin{cases} 0 & a_{dist} \leq d_{rel} \\ 1 & a_{dist} \geq d_{rel} \end{cases} \tag{3}$$

The output a_{ref} should be equal to 0 when $a_{dist} < d_{rel}$ and 1 when $a_{dist} \geq d_{rel}$. As shown from Fig. 2, the output a_{ref} of our high-level policy behaves as expected. Note the interesting behavior of the output a_{dist}, such that the initial intersection of a_{dist} and a_{ref}, there is a steep increase in

a_{dist} which also contains a higher variance. Also note that although we expect $a_{ref} = 0$ when $a_{dist} < d_{rel}$, which occurs after the second intersection of a_{dist} and a_{ref}, $a_{ref} = 1$ for several time steps. This occurs due to Cassie still being in the jumping phase.

5.2 Random Ball Velocity And X-Position Model

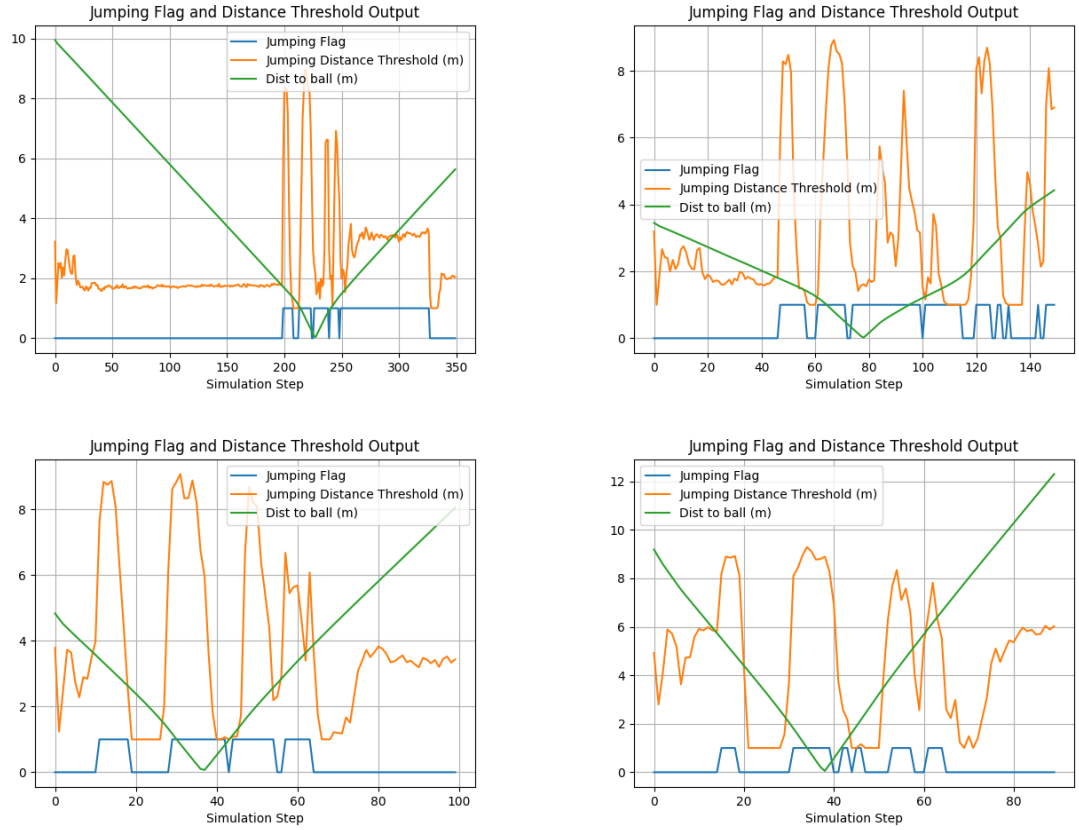

Figure 4: a_{dist}, a_{thresh}, and d_{rel} plots for models trained with initial ball conditions: 1.5 m/s 10m (top left), 1.5 m/s 5m (top right), 5 m/s 5m (bottom left), and 9.5 m/s 9m (bottom right)

The high-level policy is trained using random initialization of the position and velocity of the ball in x-direction (in front of Cassie). Specifically, we sample the radial distance (in meters) $\mathbf{r}_0 \sim \mathcal{U}[3, 10]$, $\phi = 0$, and $||\mathbf{v}||_2^2 \sim \mathcal{U}[1.75, 10.0]$. We ran a set of different tests to demonstrate the high-level's ability to find the optimal parameter a_{dist} such that the high-level task is accomplished by varying the ball and velocity initialization. Figure 4 shows similar value plots of a_{ref}, a_{dist}, and d_{rel} of the four tests described in Table 4.

As shown in Fig 4. and listed in Table 1, the policy learns to output larger values of a_{dist} for high ball velocities, resulting in earlier jumping times and thus avoiding collisions. Conversely, the high-

$Test\#$	$\mathbf{r}_0$	$\|\|\mathbf{v}\|\|_2^2$	a_{dist}
Test 1	10 m	1.5 m/s	1.9m
Test 2	3 m	1.5m/s	1.9m
Test 3	5 m	5.0 m/s	2.8m
Test 4	9.5 m	9.5 m/s	6.0m

Table 1: Initial radial position and velocity magnitudes of the ball for different test cases for the model trained with random x and y positions and random velocity magnitudes.

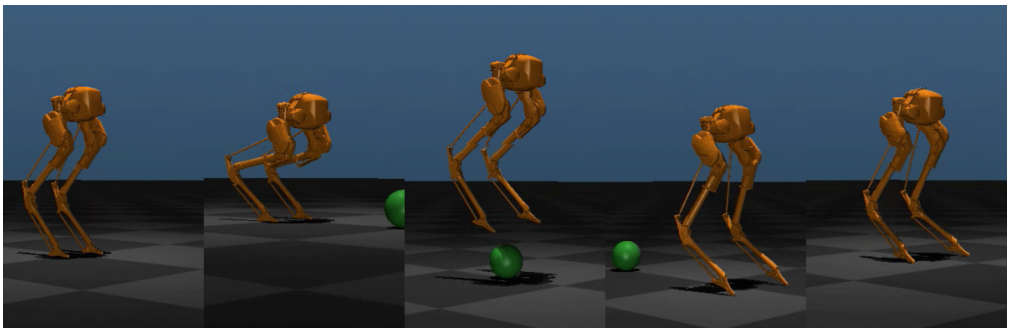

Figure 3: A snapshot sequence of Cassie successfully hopping over the ball using setup (1) where the initial ball position and velocity are constant in the x direction throughout training. The high-level policy (left) is selecting the standing reference trajectory chooses the jumping reference trajectory using the learned threshold (second image from left). After completing the jump, the high-policy selects the standing reference trajectory for the remainder of the rollout.

level policy tends to output smaller values of a_{dist} when the magnitude velocity of the ball is small. These values are depicted in the plots by the leftmost intersection of the a_{dist} and d_{rel} plot curves and explicitly shown in Table 1.

A notable difference between the random and non-random trained high-level policies is the behavior of a_{dist}. After the initial time that $d_{rel} \leq a_{dist}$ there is significantly larger variance in the behavior of a_{dist} for the randomized trained policy. The larger variance causes a fluctuation in a_{dist}, and unlike the non-random policy, a_{dist} fails to lie above d_{rel} throughout the jumping phase.

5.3 Random velocity, x, and y Positions

To demonstrate the learned behavior of the fully randomized training setup, Figure 4 shows a set of snapshot sequence images of the simulated rollout episodes with different initial ball positions. Specifically, the initial radial distance and velocity magnitude of the ball are randomized for evaluation, and θ is varied using intervals of $\frac{\pi}{4}$. Note that the velocity vector is fixed to be in the direction of Cassie, and in this training setup, the velocity parameters v_x and v_y for the low level policy are learned as opposed to the previous setups where v_y was set to zero.

As depicted by Figure 4, the high-level policy accomplishes the task of dodging the ball. Interestingly, the policy learns to dodge the ball by jumping out of its trajectory, as opposed to jumping directly over it as in previous training setups.

5.4 Discussion

The high-level policy was capable of learning inputs to the low-level policy such that Cassie accomplishes the high-level task of dodging the ball. Our initial training setups (1 and 2), fixed $v_y = 0$ which forced our policy to learn inputs that resulted in Cassie hopping directly over the ball. The policy was also able to learn the optimal distance threshold to accomplish this task. Unfortunately, during the jumping phase, we encountered unstable behavior of the policy which would cause catastrophic issues if the jumping flag was not 'latched' onto the low-level policy during the entire jumping phase.

Conversely, training the model with randomized initial x and y ball position, the policy learned a different maneuver to successfully dodge the ball. While it is not an unexpected behavior given that it is easier to dodge something by moving out of the way as opposed to jumping over it, it is surprising that the high-level policy implicitly learned the constraints of the low-level policy (i.e. maximum feet position during jumping) and found a more feasible optimal point.

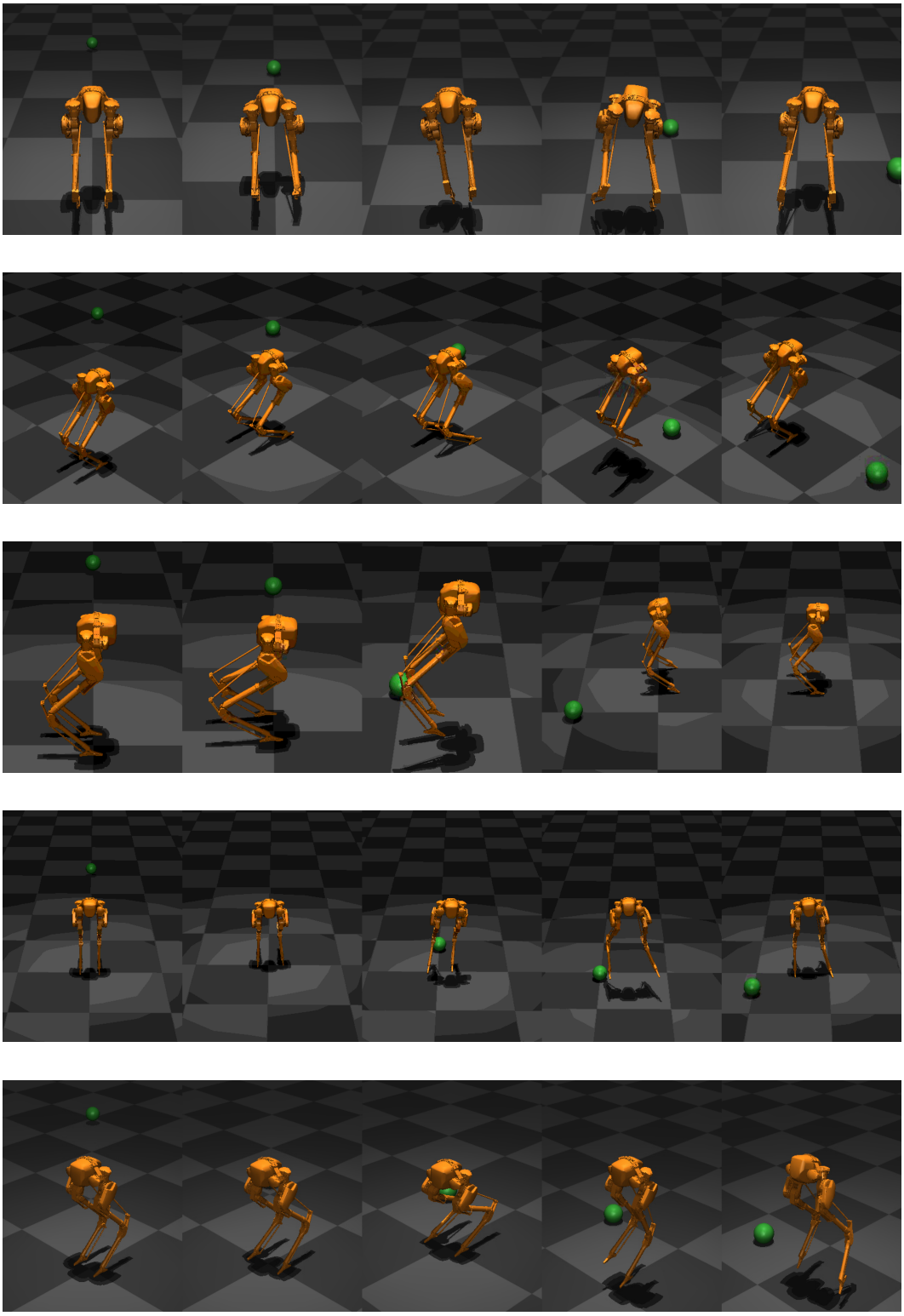

Figure 4: Snapshot sequences of Cassie dodging a rolling ball trained with randomized ball position and velocity magnitudes. There are a total of 5 snapshot sequences each corresponding to a different relative direction of the ball and Cassie, θ. We preset θ to $0, \frac{\pi}{4}, \frac{\pi}{2}, \pi, \frac{5\pi}{4}$ ordered from top to bottom with random radial distances and velocity magnitudes. In all cases, the policy has learned to dodge the ball by maneuvering out of the way. Specifically, the high-level policy is generating higher v_y values which manages to successfully dodge the ball.

6 Conclusions and Future Work

We used a combination of hierarchical and curriculum reinforcement learning to train a high-level policy using a fixed low-level policy that enabled Cassie to learn a high-level task that relied on primitive skills. Specifically, we used this approach to teach Cassie how to dodge a rolling ball by selecting the optimal reference trajectory and velocity parameters at each point in time. In addition to reference trajectory selection and desired velocity, the high-level policy learned an optimal distance at which the jumping trajectory should be selected. This skill was achievable by using a relatively simple reward function.

An interesting future direction of this work is imposing a set of different geometries and object trajectories into the environment. For example, we could consider a long rolling cylinder which would require learning a specific jumping direction and embedding the geometry of the object into the observation space. Another exciting future direction is using advanced adversarial path trajectories such that the high-level policy learns to continually dodge an object that is trying to collide with it.

6.1 Acknowledgements

This research work is funded by the National Science Foundation Graduate Research Fellowship Program. Much of the work of this project was done in collaboration with Zhongyu Li. Additionally, I would like to thank my research advisor Prof. K. Sreenath for his guidance and mentorship.

References

[1] 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," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[2] J. W. Grizzle, C. Chevallereau, R. W. Sinnet, and A. D. Ames, "Models, feedback control, and open problems of 3d bipedal robotic walking," *Automatica*, vol. 50, no. 8, pp. 1955–1988, 2014.

[3] M. Vukobratovic and B. Borovac, "Zero-moment point - thirty five years of its life." *I. J. Humanoid Robotics*, vol. 1, pp. 157–173, 2004.

[4] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion, part 1: Theory and application to three simple gait models," *The International Journal of Robotics Research*, vol. 31, pp. 1094–1113, 2012.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[6] M. Hauskrecht, N. Meuleau, L. P. Kaelbling, T. L. Dean, and C. Boutilier, "Hierarchical solution of markov decision processes using macro-actions," in *UAI*, 1998.

[7] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, p. 341–379, 2003.

[8] Z. Li, A. Narayan, and T. Leong, "An efficient approach to model-based hierarchical reinforcement learning," in *AAAI*, 2017.

[9] B. Bakker and J. Schmidhuber, "Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization," in *Proceedings of the 8-th Conference on Intelligent Autonomous Systems, IAS-8*, 2004, pp. 438–445.

[10] W. Yu, G. Turk, and C. K. Liu, "Learning symmetric and low-energy locomotion," *ACM Transactions on Graphics*, vol. 37, no. 4, p. 1–12, 2018.

[11] M. Hutter, H. Sommer, C. Gehring, M. Hoepflinger, M. Bloesch, and R. Siegwart, "Quadrupedal locomotion using hierarchical operational space control," *Int. J. Rob. Res.*, vol. 33, no. 8, p. 1047–1062, 2014.

[12] M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, and S. Schaal, "Learning, planning, and control for quadruped locomotion over challenging terrain," *I. J. Robotic Res.*, vol. 30, pp. 236–258, 2011.

[13] D. Jain, A. Iscen, and K. Caluwaerts, "Hierarchical reinforcement learning for quadruped locomotion," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7551–7557, 2019.

[14] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*. Association for Computing Machinery, 2009, p. 41–48.

[15] T. Sanger, "Neural network learning control of robot manipulators using gradually increasing task difficulty," *IEEE Trans. Robotics Autom.*, vol. 10, pp. 323–333, 1994.

[16] D. Rodriguez and S. Behnke, "Deepwalk: Omnidirectional bipedal gait by deep reinforcement learning," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3033–3039, 2021.

[17] S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *J. Mach. Learn. Res.*, vol. 21, pp. 181:1–181:50, 2020.

[18] A. Karpathy and M. van de Panne, "Curriculum learning for motor skills," in *Advances in Artificial Intelligence*, L. Kosseim and D. Inkpen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 325–330.

[19] 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.

[20] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. PP, pp. 1–34, 2020.

[21] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining." vol. Vol. 22, 2009, pp. 1015–1023.

[22] Z. Xie, G. Berseth, P. Clary, J. W. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1241–1246, 2018.

[23] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. van de Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Proc. Conference on Robot Learning (CORL 2019)*, 2019.

[24] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[25] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement learning for robust parameterized locomotion control of bipedal robots," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2811–2817, 2021.

[26] 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*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 1. Bejing, China: PMLR, 2014, pp. 387–395.

[27] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3–4, p. 229–256, 1992.

[28] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018.

[29] 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*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 2015, pp. 1889–1897.

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[31] N. M. O. Heess, T. Dhruva, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *ArXiv*, vol. abs/1707.02286, 2017.

[32] Z. Li, C. Cummings, and K. Sreenath, "Animated cassie: A dynamic relatable robotic character," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3739–3746, 2020.