Learning Agile Motor Skills on Quadrupedal Robots using Curriculum Learning

Zuoxin Tang
Georgia Institute of Technology
ztang315@gatech.edu

Donghyun Kim
University of Massachusetts Amherst
donghyunkim@cs.umass.edu

Sehoon Ha
Georgia Institute of Technology
sehoonha@gatech.edu

***Abstract*—Deep reinforcement learning offers an automated approach for developing complex motor controllers. While this method is convenient and widely used, it is known for its notorious high sample complexity, which hinders its use in many practical applications. One reason for the high sample complexity is the sparsity of reward signals, which are easy to design but slow down the convergence of policy gradient algorithms. In this work, we propose two different curriculum approaches for converting sparse reward signals to more dense formulations: environment-driven curriculum and task-driven curriculum. We demonstrate that our curriculum design allows us to learn a variety of agile locomotion skills and acrobatic motions for quadrupedal robots such as jumping, flipping, double-jumping, and wall-back-flipping.**

***Index Terms*—Legged Robot, Reinforcement Learning, Curriculum Learning**

I. Introduction

Achieving animal-level agility has been a long dream for roboticists. Unlike quadrupeds in nature that can gracefully execute a sequence of highly dynamic motions, robots typically demonstrate conservative movements to stay in the safe operation region constrained by actuator limits and sensor noise. One of the most notables examples is locomotion. There exists a clear gap between the locomotion skills of real quadrupedal animals, such as dogs or cats, and legged robots. Even with the recent advances in hardware and software, the motor skills of legged robots seem far from smooth and fluent animal movements that negotiate a variety of environments.

To reduce the gap between robots and animals, many researchers have proposed various approaches for achieving agile motor skills on legged robots. One popular approach is model-based planning that designs controllers based on the identified dynamics of legged robots. Typically, this approach decomposes the control problem into several layers, such as planning of simplified dynamics, footstep planning, feedback control design, model predictive control, joint-level control, and so on. Although it has been proven effective for many agile motor skills, such as jumping or flipping, it is known to require in-depth prior knowledge per each motor skill. On the other hand, recent deep reinforcement learning (deep RL) offers a bit more automated approach for developing control policies. By simply describing the desired behaviors using reward functions, algorithms automatically find the best weights for neural-network represented control policies. However, naive reward functions are often very difficult to learn due to their sparsity, which leads the optimizer to local minima.

In this paper, we investigate two different curriculum approaches for converting sparse reward signals into more dense formulations. The first is an environment-driven curriculum, where a curriculum parameter modifies the environment. One notable example is the jumping task: by gradually increasing the height of the obstacle, the RL agent can start to learn a policy for an easier task that gives us more meaningful reward signals. once it can overcome a shorter obstacle, we can gradually increase the height of the obstacle until it reaches the robot's capability. The second is a task-driven curriculum. In this approach, a curriculum parameter modifies the reward function. For instance, a robot can start to learn how to walk, and gradually increase the target speed. These two approaches change different entities in the formulation of the Markov Decision Process (MDP) but both can make the learning process much easier.

We evaluate two curriculum learning approaches on various agile motor tasks, including jumping, flipping, and double-jumping. For different control problems, we use one of the approaches and demonstrate that it allows us to learn effective control policies. We also investigate different observation spaces to analyze the learend policies.

II. Related Work

A. Proximal Policy Optimization

Proximal Policy Optimization (PPO) algorithm [15] is one of the most widely used policy gradient methods. It penalizes changes that make new policy far away from the old policy. Instead of using the original objective function,

$$L(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi'_\theta(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t [r_t(\theta) \hat{A}_t] \tag{1}$$

it uses a clipped objective function

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

Comparing to its predecessor, Trust Region Poliy Optimization (TRPO) [14], PPO is easier to implement and more scalable to large models and parallel implementation.

B. Curriculum Learning

Curriculum Learning (CL) was introduced for Supervised Learning earlier [2], however it becomes an important technique in reinforcement learning and robot motion control recently. This method usually start with an easy task and evolve gradually into a difficult task. For example, [19] designed curriculum as assistive force introduced by a proportional-derivative (PD) controller, and its coefficients become the curriculum space. A window in that space is sliding from the part where PD controller does most of the work to another extreme where PD controller is disabled. Some other work [18] uses an adaptive curriculum update algorithm to reduce number of hyper-parameters to tune and show that adaptive curriculum outperforms naive curriculum. On top of curriculum design, [1] introduces domain randomization in its curriculum to counter uncertainties existing in the real world such as sensor error and motor delays. Our project is mainly inspired by [19]. To shape the behavior of robot, we also change environment gradually. With proper reward design, we successfully train policies that can perform highly dynamic skills.

C. Robot motion control

In the area of robot motion control, Model-free RL algorithm is often used to train a fixed policy that map from robot observation to its action (or decision). Previous work [5], [11], [12] has shown the power of neural network polices that can map from high dimensional observation to high dimensional action. The advantage of this method is that, after training, the execution time of policy is short. We can also mask some of the observation to fit our test environment while using privileged information in our training environment [3]. Another category of robot motion control algorithm is model-based algorithm, where an optimization is performed during execution. There are many works that use this method, such as [8], [9], [17], or use a combination of both model-free and model-based algorithms [4], [10]. In this paper, we focus on model-free RL algorithm.

III. Methods

A. Reinforcement learning

Reinforcement learning (deep RL) aims to learn a policy that maximizes the expected sum of rewards [16]. We formulate the visual navigation task as Partially Observable Markov Decision Processes (POMDPs), $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, r, p_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{O}$ is the observation space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the transition function, r is the reward function, p_0 is the initial state distribution and γ is a discount factor. We take the approach of model-free reinforcement learning to find a policy π, such that it maximizes the accumulated reward:

$$J(\pi) = \mathbb{E}_{\mathbf{s}_0,\mathbf{a}_0,\ldots,\mathbf{s}_T} \sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t), \tag{3}$$

where $\mathbf{s}_0 \sim p_0$, $\mathbf{a}_t \sim \pi(\mathbf{o}_t)$, $\mathbf{o}_t \sim c(\mathbf{s}_t)$ and $\mathbf{s}_{t+1} = \mathcal{T}(\mathbf{s}_t, \mathbf{a}_t)$.

B. Curriculum Learning Approaches

Curriculum learning is a technique for solving complex problems by gradually ramping up the task difficulty from simple to the end. The effectiveness of the algorithm has been proven in many robotic applications, including locomotion, grasping, and navigation. In curriculum learning, the difficulty of a task is tuned by a task parameter μ that is ranged from 0 to 1. Intuitively, we can imagine that a task with the parameter $\mu = 0$ is an easy task while a task with the parameter $\mu = 1$ is the hardest task. For instance, the locomotion task with $\mu = 0$ is to walk on flat terrain while the locomotion with $\mu = 1$ is to negotiate a challenging terrain with many obstacles. We utilized two types of curriculum in this study.

The first is an environment-driven curriculum, where the task parameter μ directly modifies the environment, such as the size of the obstacles in the environment. In the example of the jumping task, we map the task parameter μ with the size of the obstacle. We start from the small height which can be easily jumped over by the robot, and gradually increase the height when the robot can successfully overcome the given height. We use this approach for the tasks that have obstacles in the robot's path.

The second is a task-driven curriculum, where the task parameter μ modifies the reward function. For instance, the task parameter of basic locomotion training is mapped to the target velocity term in the reward function and we progressively increase the target speed from low to high. In the other tasks like flipping jumps, the sample principles are applied. We guide the training to learn flipping motion first and gradually add the safety penalty in the final posture to encourage the robot to learn safe landing.

Another important design factor of curriculum learning is how to advance the task parameter μ. Instead of using a fixed scheduling technique, we increase the task parameter by a small amount when the robot can successfully execute the current task. We measure the success of the task by comparing the episodic reward against some threshold, $\bar{R}$, which is separately configured for each task.

IV. Experiments

We use RaiSim [7] and PPO [6] for training and testing. The RaiSim supports vectorized simulation and has fast simulation speed. The dynamics simulation runs in 400 Hz and policy control update frequency is set to 100 Hz. The agent will be give its IMU information (direction of gravity in its local coordinate system), information from its motor encoder (including joint position and velocity), position and velocity of robot's trunk. We test four different tasks: running jump, backflip, barrel roll, and double jump. The different reward design of each task will be described below. Similar to [11], we introduce early termination and termination penalty in the training and reward. Finally, to avoid jerky motion, we add a low-pass filter [13] (cutoff frequency is lower than 7.5 Hz) in the joint controller.

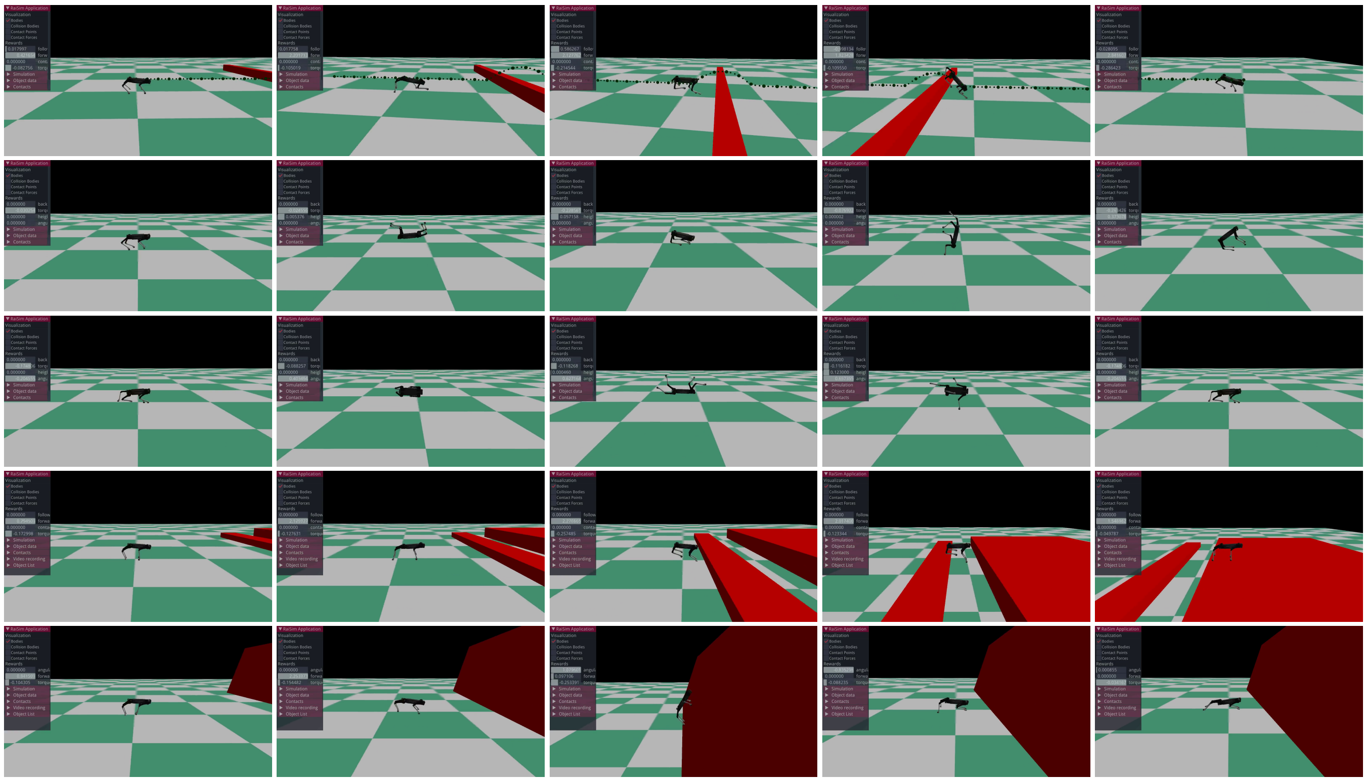
Fig. 1. **Dynamic motion demonstration.** Various agile motor tasks are learned using the proposed curriculum learning algorithms. From the top, the demonstrated motions are running jump, back flip, barrel roll, double jump, and wall backflip.

A. Running Jump

In the running jump task, the robot is asked to run forward and do one obstacle clearance. Our reward function is designed to: 1. reward forward velocity 2. penalize use of joint torque 3. penalize deviation from guiding trajectory 4. penalize contact with obstacle. 5. penalize early termination. The guiding trajectory is a manually set cubic spline so that the robot will do running jump instead of stopping in front of the obstacle and go around it. At the beginning of curriculum, the obstacle does not present, and robot is trained to run forward. The curriculum is set as follows: once the reward of the agent reaches a certain threshold, the obstacle will grow a small amount of height, until it reaches our desired height.

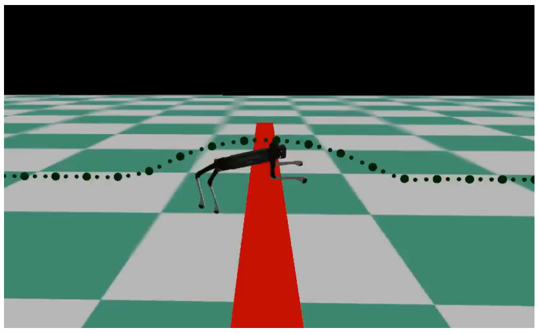
Fig. 2. Running jump

B. Backflip

In this task, the robot performs backflip on the ground. Reward function is set as followings: 1. reward angular velocity perpendicular to its sagittal plane 2. penalize angular velocity along its trunk direction 3. penalize use of joint torque 4. penalize early termination. The curriculum is designed as two phase: first we set early termination to be zero. Once the robot is able to do a complete flip, we gradually introduce early termination penalty so that the robot can land safely.

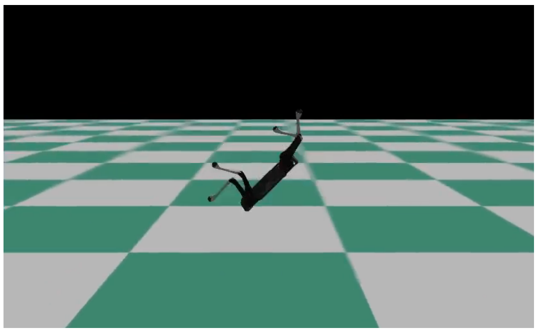
Fig. 3. Backflip

C. Barrel Roll

Barrel Roll is similar to backflip except that it will flip around its front direction. Minimum tuning on the weight of each reward is conducted to encourage natural barrel roll.

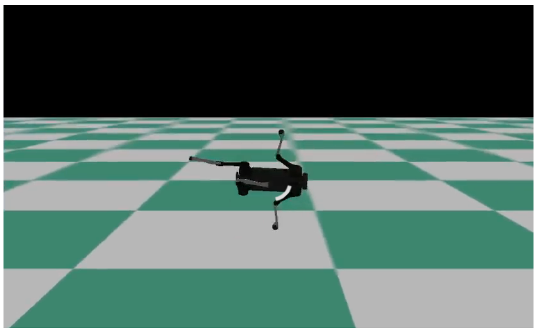

Fig. 4. Barrel Roll

D. Double jump

In the double jump task, the robot needs to jump twice in order to reach a higher platform. To reduce the task difficulty, we limit the robot in 2D, namely we reduce the DoF of robot trunk to be three and force the movement of robot's limbs to be symmetric along both sides of sagittal plane. The reward function consists of two parts: 1. reward forward velocity 2. penalize early termination.

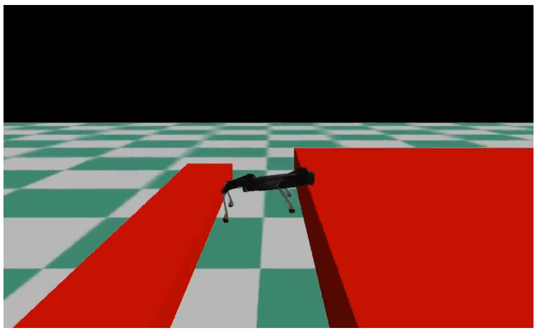

Fig. 5. Double jump

E. Wall Backflip

In this task, we teach the robot to kick off the wall and perform a backflip. We design a curriculum to negotiate a wall with different angles. Once the robot lands on the wall, the wall exerts the force on the robot to help its backflip. This is done until the robot successfully executes the task.

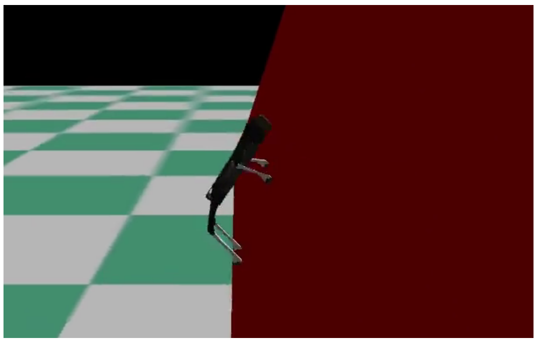

Fig. 6. Wall Backflip

F. Results and Discussion

Our curriculum learning algorithm display success on all our four tasks. To our surprise, backflip and barrel roll are much easier than running jump and double jump in terms of training samples. This is counter intuitive because for human, jumping over obstacles is easy but doing flipping is difficult. We suspect that this is because of the training cost of each sample. Doing backflip is much more dangerous than doing running jump and therefore in practice. This shows the value of researching on sim-to-real transfer where we can ignore much of the sampling risk. Another hypothesis is that, flipping is easier because of its shorter task horizon.

We also do experiments on whether certain observation is vital to the agent. We tried remove global position of trunk in running jump task to see how it will affect the result. The environment steps vs. obstacle height curve of this experiment is shown in Figure 7, where ro means reduced observation, fo means full observation, sn means small network (two hidden layers with 128 neurons each layer), ln means large network (two hidden layers with 256 neurons each layer). This curriculum curve shows that removing some important observation from the agent can hinder training but increasing the policy network can somehow remedy the result. Besides, as we expect, small network will finish the first stage (no obstacle exists) earlier while it takes some more samples for larger network until obstacle starts to grow.

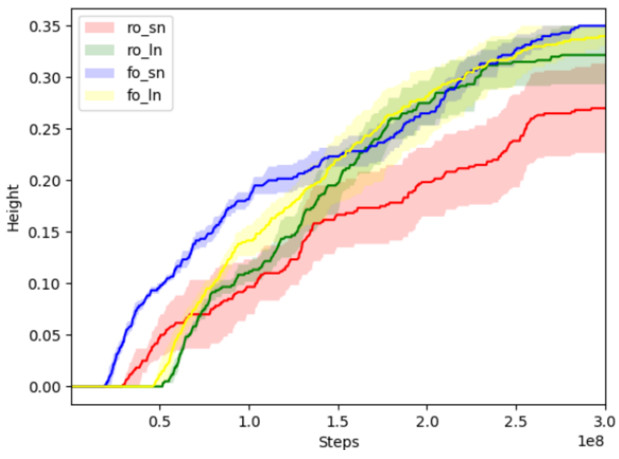

Fig. 7. Experiments on full/reduced observation

V. Conclusion and Future Work

In this project, we use curriculum learning and reinforcement learning to train robot to do highly dynamic skills in simulation environment. With the power of vectorized environment and fast simulation, we are able to train policies successfully for different skills of robot within reasonable amount of time. One of the possible future work of this project is to concatenate different skills together, for example, doing multiple obstacle clearances or doing a wallflip, a combination of running and flipping. Further more, it is also interesting to investigate how we can use a higher level meta policy to control different skills so that robot can do tasks with longer horizon.

References

[1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[3] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[4] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[5] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[6] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[7] Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902, 2018.

[8] Donghyun Kim, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. *arXiv preprint arXiv:1909.06586*, 2019.

[9] Taesoo Kwon, Yoonsang Lee, and Michiel Van De Panne. Fast and flexible multilegged locomotion using learned centroidal dynamics. *ACM Transactions on Graphics (TOG)*, 39(4):46–1, 2020.

[10] Sergey Levine and Vladlen Koltun. Guided policy search. In *International conference on machine learning*, pages 1–9. PMLR, 2013.

[11] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[12] 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.

[13] Bernd Porr. Dsp iir realtime c++ filter library. https://github.com/berndporr/iir1, 2020.

[14] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[16] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[17] Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

[18] Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. Allsteps: Curriculum-driven learning of stepping stone skills. In *Computer Graphics Forum*, pages 213–224. Wiley Online Library, 2020.

[19] Wenhao Yu, Greg Turk, and C Karen Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.