Building Adaptive Quadrupeds: Reinforcement Learning for High-Performance Jumping

Grayson Martin, Aryan Naveen, Pranay Varada
CS/Stat 184(0)
graysonmartin@college.harvard.edu
aryan_naveen@college.harvard.edu
pranayvarada@college.harvard.edu

Abstract

Agents acting in dynamic environments must be able to adapt to the conditions of these environments. Quadrupedal robots are particularly useful for simulating agents in these environments because of their flexibility and real-world applicability. This reinforcement learning project aims to teach a quadruped to jump over a moving obstacle, using Proximal Policy Optimization (PPO) and a two-stage curriculum learning framework. Stage I focuses on achieving the ideal jumping motion, while stage II integrates dynamic obstacles, utilizing a Markov Decision Process (MDP) modified to include motion dynamics and obstacle detection. Reference state initialization (RSI) and domain randomization are employed to enhance robustness and generalization. Simulations in IsaacGym demonstrated that two-stage curriculum learning improved upon direct training in minimizing obstacle collisions, particularly for obstacles at closer ranges. Timing remains a challenge when attempting to avoid obstacles initialized from farther away. This work highlights the value of curriculum learning methods in RL for robotic tasks and proposes future improvements in reward shaping and learning strategy for enhanced adaptability in dynamic environments.

1 Introduction

Throughout the field of reinforcement learning, there is significant demand for creating strategies to learn an optimal policy that can achieve desired reward across changing environments. Furthermore, it is essential that such a policy can *understand* the changes in these environments and act accordingly. After all, many environments in the real world are non-stationary, with varying conditions in weather and terrain acting on agents, for example. Adaptive policies are also suitable for long-term success because of their higher levels of robustness. In use cases from search-and-rescue operations to industrial inspections, such adaptability can be critical.

Our project aims to solve this problem of understanding changing environments through teaching a quadruped to react to a obstacle moving towards it with a random angle and velocity. Quadrupedal robots such as Boston Dynamics' Spot and ANYbotics' ANYmal are particularly advantageous for undertaking such a project for three reasons. First, there is extensive literature on reinforcement learning for quadruped locomotion, which means that we can iterate on different implementations of RL strategies in order to solve new problems. Specifically, while quadrupedal jumping may be well studied, our project aims to understand how an agent can react to random timing and position. Second, simulations of quadruped movement are both high-dimensional and visually easy to comprehend, making it clear what the agent has learned once the training process is complete. Third, there is a strong basis for quadruped simulation of and translation to real-world situations where changing

Submitted to Foundation Models for Science Workshop, 38th Conference on Neural Information Processing Systems (NeurIPS 2024).

environments may be at play; a desire to understand environments motivated our decision to work on this project.

To tackle the particular scenario of jumping over a moving obstacle, we consider several RL techniques. We begin with Proximal Policy Optimization (PPO) as the foundational approach, and subsequently integrate curriculum learning strategies into the training process to systematically optimize the quadruped's ability to master this (perhaps surprisingly) complex task. Curriculum learning breaks this task down into sub-problems, such that the quadruped first learns how to jump, and once it has mastered that skill, learns how to jump over a moving obstacle.

The remainder of the paper is structured as follows: Sections 1.1 and 1.2 detail the existing theory and literature that are pertinent to the experiments we carried out in this report, Section 2 details the modified MDP for our desired problem setting and task of jumping over dynamic obstacles. Sections 3 and 4 detail the selected approaches to solving this problem based on the pre-existing literature. And finally, in section 5 we detail the simulated results we observed and evaluate the learned behaviors performance for various approaches.

1.1 Preliminaries

Proximal Policy Optimization (PPO) is a policy gradient method that improves upon previous methods in the literature through its relative ease of implementation, requiring just first-order optimization, and greater robustness relative to optimization techniques such as Trust Regio Policy Optimization (TRPO) [6]. While the objective function maximized by TRPO is the expectation of the product of the advantage and a probability ratio measuring the change between the new and old policy at an update, PPO's objective function clips the probability ratio in this surrogate objective in order to prevent the policy from making unstable updates while simultaneously continuing to allow for exploration. Clipping also avoids having to constrain the KL divergence, making the process computationally simpler and enabling policy updates over multiple epochs. PPO's simplicity and stability make it a commonplace strategy for finding the optimal policy in RL, which is why we use it as a baseline from which we search for possible improvements, namely curriculum learning methods.

1.2 Literature review

Atanassov, Ding, Kober, Havoutis and Santina use curriculum learning to stratify the problem of quadrupedal jumping into different sub-tasks, in order to demonstrate that reference trajectories of mastered jumping are not necessary for learning the optimal policy in this scenario [1]. This increases the adaptability of such a policy, because it is learned by the robot on its own, enabling it to generalize better to unseen real-world scenarios. Another important component of achieving the optimal policy is reward shaping, which Kim, Kwon, Kim, Lee, Oh tackle in a stage-wise fashion in the context of a humanoid backflip [3]. By developing customized reward and cost definitions for each element of a successful backflip, a complex maneuver like this is segmented into an intuitive fashion that translates well to real-world dynamics.

2 Problem formulation

We utilize a Markov Decision Process (MDP) as the underlying sequential decision-making model for our RL problem. The MDP is described as a tuple where $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, P, \rho, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta(S)$ is the transition operator with $\Delta(S)$ as the family of distributions over $\mathcal{S}$, $\rho \in \Delta(S)$ is the initial distribution and $\gamma \in (0, 1)$ is the discount factor. The overall objective of reinforcement learning is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the cumulative infinite-horizon discounted reward:

$$\mathbb{E}_{s_0 \sim \rho, \pi} \left[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \mid s_0 \right] \tag{1}$$

A given policy π has a value function under transition dynamics P defined as the expected cumulative reward from a given state: $V_P^\pi(s) = \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \mid s_0 = s]$, and the state-action value function under transition dynamics is similarly defined as $Q_P^\pi(s, a) = \mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \mid s_0 = s, a_0 = a]$.

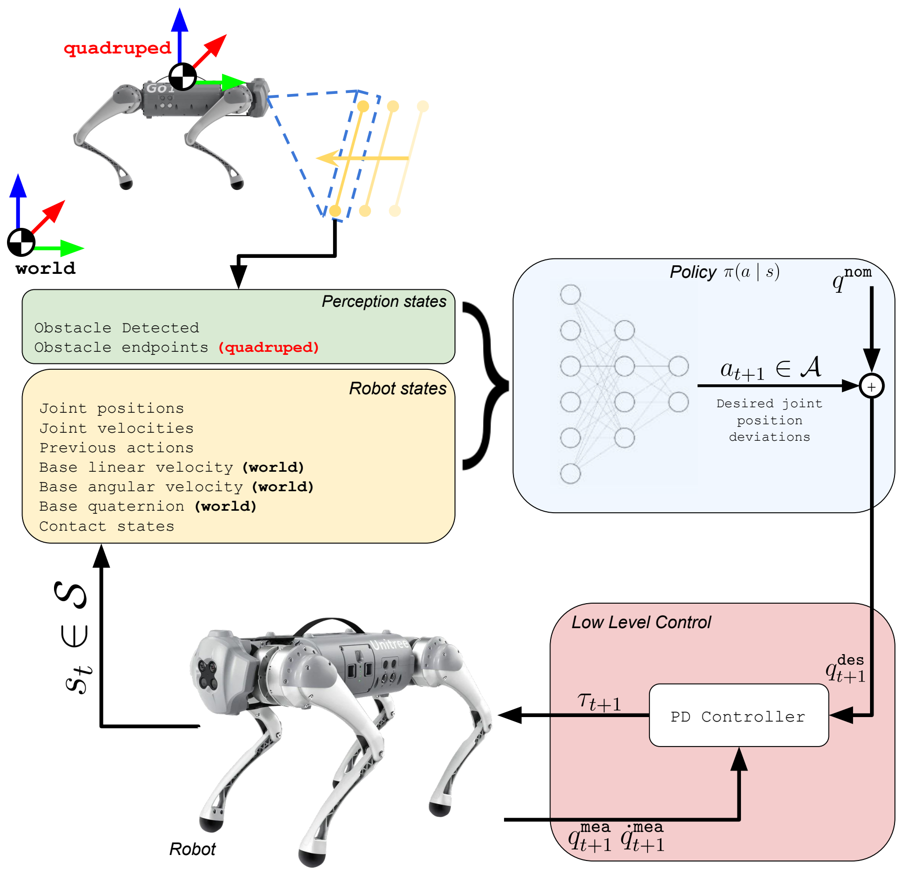


Figure 1: Overview of the hierarchical control framework for a quadruped robot. The trained policy π yields desired joint position deviations from the nominal joint positions that are then fed into a low-level PD controller that producing necessary torques τ for each joint.

2.1 Quadruped jumping obstacle avoidance MDP formulation

As outlined in Section 1, in this report we are interested in extending the work of [1] in order to enable a quadruped to jump over dynamic obstacles. This enhancement requires not only integrating additional learning strategies, such as curriculum learning, but also modifying the underlying MDP formulation to account for the quadruped's perception module's feedback as shown in Figure 1.

State Space. As proposed by [1], we leverage a memory of previous observations and actions in order to enable the agent to implicitly reason about its own dynamics and interaction with the environment as visualized in Figure 2. By leveraging a concatenation of the previous N observations, we are able to allow π to reason about dynamics and predict much more effectively. As visualized in Figure 2, similar to [1], the state space $\mathcal{S}$ contains a historical window of the robot's base linear velocity $\mathbf{v} \in \mathbb{R}^{3\times N}$, base angular velocity $\omega \in \mathbb{R}^{3\times N}$, measured joint positions $\mathbf{q} \in \mathbb{R}^{12\times N}$, measured joint velocities $\dot{\mathbf{q}} \in \mathbb{R}^{12\times N}$, previous actions $\mathbf{a}_{t-1} \in \mathbb{R}^{12\times N}$, the base orientation $\bar{q} \in \mathbb{R}^{4\times N}$ and the foot contact states $\mathbf{c} \in \mathbb{R}^{4\times N}$. However, given our objective to train policies that can react appropriately to dynamic environments, we introduce additional components to our state space $\mathcal{S}$. Specifically, we provide the policy with information about whether an obstacle was detected i^{ζ} as well as the midpoints and endpoints of the obstacle ζ in the quadruped's coordinate frame. Similar to the windowed history of dynamics for robot states, we apply the same principle to the detection states to allow the policy to also reason about the dynamics of the surrounding environment. As illustrated in Figure 2 these observations are concatenated together to yield a $60N$ state space dimensions that encodes everything we believe is necessary to learn the behaviors of jumping over dynamic obstacles.

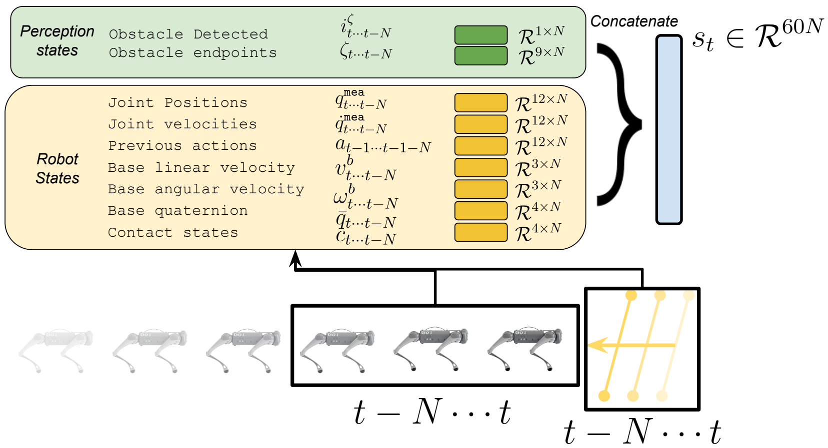


Figure 2: Overview of the state representation used for training our jumping over dynamic obstacle policies. The state vector $s_t \in \mathbb{R}^{60N}$ is constructed by concatenating perception states (obstacle detection and endpoints) and robot states (joint positions, velocities, previous actions, base velocities, quaternion, and contact states) over the past N timesteps. This temporal sequence enables the policy to learn dynamic behavior by incorporating historical information.

Action Space. As is standard in quadruped policies, our policy generates the desired twelve actuated joint angles $\mathbf{q}^{\texttt{des}} \in \mathbb{R}^{12}$ for control. Specifically, the policy learns the deviation from the nominal joint positions $\mathbf{q}^{\texttt{nom}} \in \mathbb{R}^{12}$. Additionally, it is standard for the output actions to be smoothened utilizing a low-pass filter and then scaled before being added to $\mathbf{q}^{\texttt{nom}}$ to compute the $\mathbf{q}^{\texttt{des}}$ for the motor servos. As visualized in Figure 1, a low level controller is utilized to compute the necessary joint torques to attain the computed setpoints.

Reward. It is important to note that our reward formulation is largely inspired by [1]. They contend that in order to combat local minimas, such as standing behaviors without jumping to avoid energy penalties, rather than naively summing them, we should multiply the positive components of the reward by the exponent of the squared negative component:

$$r_{\texttt{total}} = r^+ \exp\left(-1||r^-||^2/\sigma\right)^4$$

This allows for the observed reward to remain positive, where incurred penalties scale down the observe reward to improve stability. Specifically, the agents are rewarded based on the phase of the jump behavior they are in: (1) stance, (2) flight, and (3) landing. [1] claimed that this reward formulation encouraged desired behavior to be found quicker based on the fact that it is built around a reference desired behavior. We observe similar advantages as a consequence of this formulation, as initial experiments with a constant reward formulation observed successful continuous mini jumps but never large athletic jumps. Additionally, reward components are divided into *sparse* and *dense* types, where the former is given once per episode and the latter is given per each simulation iteration.

1. **Dense Rewards:** Several dense rewards considered in this formulation, inspired by [1] is tracking a desired linear velocity, and yaw angular velocity while in flight. Additionally squatting down to a height of 0.2m before taking off. Additionally, to encourage tuck behaviors (as seen in Figure 6) a foot clearance reward is added to minimize the z-distance between each foot and the centre of mass while in the air. We also introduce penalties for energy used (based on commanded torques) to encourage learned policies that are effecient and do not waste energy.
2. **Sparse Rewards.** We additionally award several rewards/penalties based on the final result of the jump. For instance, max height and whether the agent jumped in the episode are rewarded. Moreover, if the agent prematurely terminates we penalize the reward observed.

Reasons for premature terminations include (1) falling over, (2) collision between body links, (3) orientation error larger than π, and a couple others.

Note that, while we decided to remain close to the formulation derived in [1], given that quadruped jumping is a unique behavior that requires extensive attention to detail, we do include key modifications in latter stages of our curriculum learning to enable the timing component necessary to avoid the obstacle, as outlined in Section 4.

3 State initialization and domain randomization

An important aspect of such an MDP formulation is selecting the *initial state distribution*, which we will represent $\rho(\mathcal{S})$. When applying learning methods to agents such as quadrupeds, for many tasks it is convenient to initialize the agent in a static state in the learning process. However, for certain tasks such as quadruped jumping, such an initial state distribution is undesirable when a lack of diverse and informative initial states discourages the agent from exploring desired trajectories. Consider the quadruped jumping scenario in which termination penalties are applied to the reward function when the quadruped falls over. Having not yet learned how to stick a landing, the agent sees that jumping high leads to a large penalty and stops attempting to learn to jump high. Further, static initialization can make it difficult for a policy to learn that certain states have high rewards. In the quadruped jumping example, if the quadruped is always initialized on the floor, the policy never sees that height off of the floor is associated with high positive rewards.

To combat such scenarios, Peng, Abeel, Levine, and van de Panne introduce a strategy of *reference state initialization (RSI)* [5]. This method of state initialization is an imitation learning technique in which the agent's initial state is sampled from the reference trajectory it is trying to learn. More formally, for reference expert trajectory $\tau_{ref} = \{s_0^{ref}, (s_1^{ref}, a_1^{ref}) \dots (s_{H-1}^{ref}, a_{H-1}^{ref})\}$, $\rho(\{s_0^{ref} \dots s_{H-1}^{ref}\})$ is given by some distribution across the values of s^{ref}. The agent then encounters desirable states along the expert trajectory, even before the policy has acquired the proficiency needed to reach those states [5].

In their quadruped jumping formulation, Atanassov, Ding, Kober, Havoutis and Santina use a modified version of RSI in which they sample a random height and upward velocity from a predefined range rather than using an explicit reference trajectory [1]. Since there is no reference trajectory, we want an intelligent choice of $S_{init} \subset S$ such that $s_0 \sim \rho(\mathcal{S}_{init})$ for the given task. In our case, $s_0 \sim \mathcal{U}(\mathcal{S}_{init})$ where $\mathcal{U}$ represents the uniform distribution and $\mathcal{S}_{init}$ takes the range of values shown in table 1 for stage I training. It should be noted that for stage I training, all components of the state space that are properties of the obstacle are set to 0. To highlight the importance of RSI, Figure 4 demonstrates the impact of implementing RSI in the first stage.

Table 1: Initialization Ranges for Quadruped State Space Variables in Stage I

State Space Variable	$\mathcal{S}_{init}$ min	$\mathcal{S}_{init}$ max
Height (m)	0	0.3
z velocity (m/sec)	-0.5	3

In addition to using this modified RSI for the quadruped itself, we further utilize randomization of obstacle states in s_0 in the second stage of learning outlined in section 4. Ranges for $\mathcal{S}_{init}$ are given in table 2 for stage II training. Here RSI is necessary so that the quadruped learns to jump over obstacles moving at various speeds and with any distance or orientation. An interpretation of the obstacle characteristics is given in Figure 3.

In addition to RSI, a related technique called *domain randomization* is often utilized for policies training in a simulation environment in an attempt to allow for maximum real-world generalization and decrease the sim-to-real gap. This concept was introduced by Tobin, Fong, Ray, Schneider, Zaremba, and Abbeel, wherein instead of training a model on a single simulated environment, the simulator is randomized to expose the model to a wide range of environments at training time[7]. The original reinforcement learning quadruped environment we built upon includes zero-shot domain randomization for ground friction, ground restitution, additional payload, link mass factor, center of

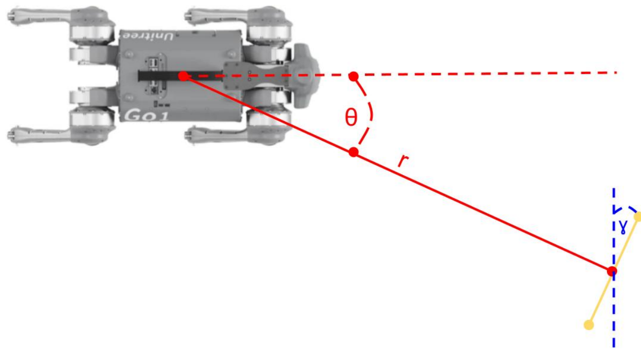


Figure 3: State Space Variables Related to the Obstacle

Table 2: Initialization Ranges for Obstacle State Space Variables in Stage II

State Space Variable	$\mathcal{S}_{init}$ min	$\mathcal{S}_{init}$ max
Obstacle distance r (m)	3	7
Obstacle direction θ (rad)	0	2π
Obstacle orientation γ (rad)	$\frac{\pi}{2} - \theta - \frac{\pi}{3}$	$\frac{\pi}{2} - \theta + \frac{\pi}{3}$
Obstacle velocity (m/sec)	3.5	7

mass displacement, episodic latency, extra per-step latency, motor strength, joint offsets, PD gains, joint friction, and joint damping[1].

4 Obstacle avoidance curriculum learning

Curriculum learning, generally attributed to Bengio, Louradou, Collober, and Weston, is a sequential method of model training wherein the model is first taught a simple task or building block and then goes on to be trained on more difficult problems that require the building blocks [2]. This method of training a network is an intuitive model for how humans learn complex tasks: students first learn basic math, then use those tools to learn algebra, then use those tools to learn calculus, and so on. Such a learning procedure has been shown to be especially useful for reinforcement learning [4]. Sample efficiency is often vastly improved as the agent learns useful representations and behaviors early in training before moving to more difficult tasks. Stable partial solutions reduce high variance in returns and prevent the agent from getting stuck in poor local optima when faced with complex versions of the task. Because the agent has a sequence of diverse learning experiences, it often generalizes better to new or slightly different tasks.

In the context of quadruped tasks involving jumps, Peng, Abeel, Levine, and van de Panne use curriculum learning in the following manner: stage I teaches the quadruped how to jump, stage II teaches the quadruped how to jump to a desired position and orientation, and stage III teaches the quadruped how to jump onto or over platforms [1]. For our task, we utilize the same stage I training to teach the quadruped how to jump in place. However, instead of making the robot motion more difficult such as the original experiments, we make the next stage more difficult by introducing a flying obstacle to the environment. Accordingly, the reward function is updated to include a sparse penalty and termination for collision with the obstacle. The sparse reward calculation is given in equation 2, where ζ represents the obstacle and the reward scale is -50.

$$r_{col} = \mathbb{1}\left\{ \min_{f \in \text{quadruped feet}} \left\| (f_x, f_y, f_z) - (\zeta^{(x)}, \zeta^{(y)}, \zeta^{(z)}) \right\| \leq 0.1 \right\} \tag{2}$$

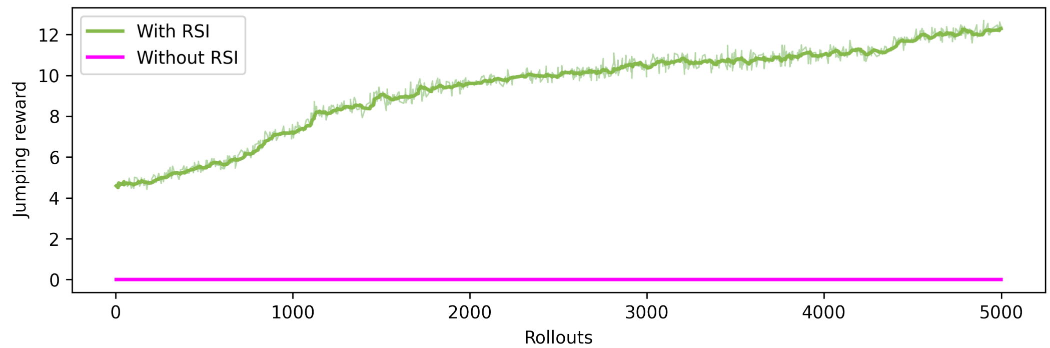


Figure 4: Stage I Training with and without RSI

Another important distinction is that, unlike the original experiments, our agent is not randomly returned to stage I while training later stages. In the original work, the authors wanted to maintain a more "ideal" jumping motion while learning the later tasks, but our objective mostly values clearing the flying obstacle. A "worse" jumping form is more desirable here, as taking the time to enter an athletic stance sometimes allows a fast-moving obstacle to hit the quadruped before it is able to leave the ground. To reflect this, we decrease the reward scale for crouching in an athletic position before jumping in the second stage from 5 to 1. The reward for crouching to the desired height for an athletic stance is given in equation 3.

$$r_{squat} = 0.6 \exp\left(-\frac{(\text{height} - 0.2)^2}{0.001}\right) \tag{3}$$

5 Experiments

In our experimental setup, we primarily tested two different strategies for discerning the optimal policy for jumping over a moving obstacle. Firstly, our "control" was training without curriculum learning (20,000 iterations). This process essentially skipped our aforementioned stage I, such that the quadruped would go straight to attempting to master jumping over a moving obstacle. The performance of the curriculum learning-free method served as a baseline for our second strategy: 5,000 iterations in stage I – with no dynamic obstacle present – such that the quadruped would learn to jump in place, followed by 15,000 iterations in stage II with the dynamic obstacle in place, through which the quadruped would ideally use its ability to jump to learn how to jump over this obstacle. We chose these particular numbers of iterations because the policy converged after 5,000 iterations in stage I, and we assumed that stage II would be more difficult. In both cases we utilize the same network architecture in order to isolate the learning algorithms performance.

5.1 Training Environment

We train our policies in IsaacGym because of its speed and scalability, resulting from the capability to sample multiple environments in parallel. IsaacGym enables massive parallelism by simulating thousands of environments simultaneously on a single GPU, reducing training time. Its seamless integration with PyTorch and support for high-fidelity physics make it an ideal platform for developing and testing policies in dynamic and complex environments. This efficiency and realism allow us to iterate quickly and deploy robust, high-performing policies. Additionally, IsaacGym offered preexisting support for several quadrupedal-based robot environments, which allowed our research to focus purely on the learning algorithms rather than simulator design. As shown in Figure 6 we are able to train and evaluate realistic policies in IsaacGym.

5.2 Network Architecture

While utilizing PPO, both actor and critic network architectures are the same with only the output layer being different, with the former having 12 output neurons while the latter had 1 to approximate the Q function.

Architecture. Three fully connected layers with $|s|$, 512, 512, and 12 neurons. Non-linear activations are included between each layer with a ReLU function. Sampled actions are normalized with a tanH function to ensure that actions remain between -1 to 1 and then scaled appropriately.

5.3 Results

To compare these two methods, we looked at the number of times (out of 50) that the quadruped collided with an obstacle initialized at radii from 3m to 7m away at 1m increments. This range of distance provided suitable grounds of comparison between the two methods because the obstacle was not initialized too close to or far away from the quadruped. As Figure 5 shows, the curriculum learning method performed better than the method without curriculum learning across each initial distance, and markedly so at a radius of 3m away. While the quadruped failed to clear a majority of obstacles at longer radii, the results tell us that segmenting training into two sub-task sections can be at least somewhat effective in obstacle avoidance.

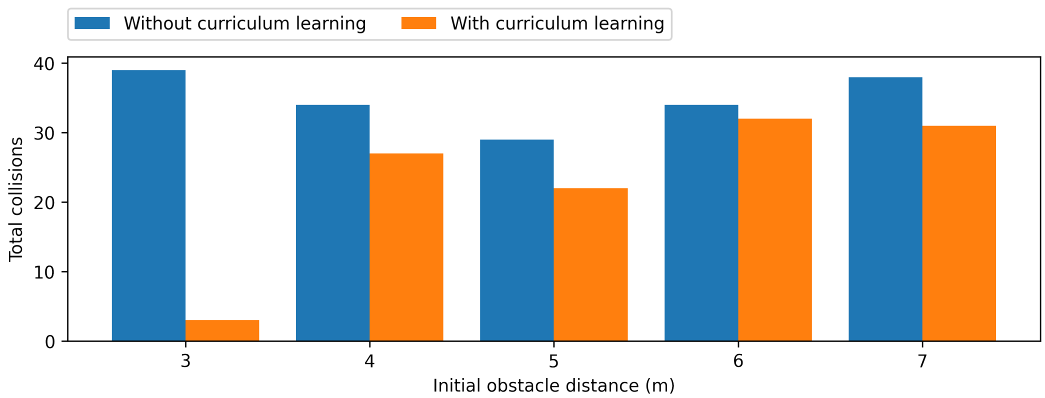
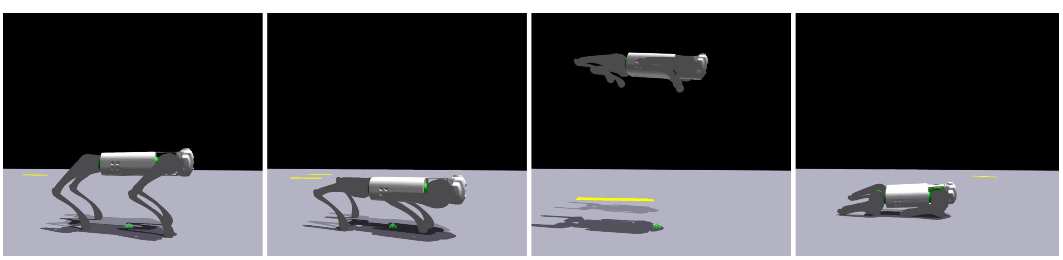


Figure 5: Number of Collisions at a Given Distance (50 Agents)

Why does the performance at longer radii leave much to be desired? Firstly, consider Figure 6, which demonstrates the quadruped jumping motion. We observed that the quadruped clearly learns how to jump – by settling into an athletic stance, propagating upwards from that stance, and landing decently well on its feet. It even learns how to jump over an obstacle it perceives, as demonstrated by its avoidance of all but three obstacles at a radius of 3 quadruped lengths. However, it is not clear that the quadruped has learned how to properly time its jump. It tends to jump as soon as it has perceived an obstacle, but at longer radii, this can result in a jump and a landing before the obstacle even arrives at the quadruped's position, resulting in a collision. Therefore, in future training, it would be important to encourage patience: the quadruped should gauge the speed of the obstacle such that it jumps only when the obstacle is sufficiently close to be cleared in a single jump.



Initialized Position | Athletic Stance | Jumped over obstacle | Post-jump landing

Figure 6: The Quadruped Jumps Over a Flying Obstacle

In addition, as mentioned in Section 4, we do not desire the ideal jumping motion in this scenario, because we want to minimize the time the quadruped takes to enter an athletic stance. It could be that the curriculum learning process as currently constructed places too much emphasis on attaining the *ideal* jumping form, rather than simply learning *how* to jump and maximizing obstacle avoidance efficiency from there.

6 Conclusion

In conclusion, this work has demonstrated the feasibility and effectiveness of leveraging reinforcement learning and curriculum learning strategies to enable a quadrupedal robot to perform jumps over dynamic obstacles. With our formulated MDP using PPO as the baseline algorithm, we demonstrated that the structured, multi-stage approach of curriculum learning can feasibly be used to train agents in dynamic environments. The quadruped was guided through tasks with increasing difficulty, starting from basic jumping mechanics and eventually learning the more complex task of dodging flying obstacles.

Beyond curriculum learning, modified RSI was crucial to the project. RSI ensured that the quadruped encountered a broad distribution of initial states, including intermediate positions it would not naturally discover from a static start. This exposure accelerated learning and prevented the policy from converging prematurely to trivial but non-transferable behaviors. Domain randomization was also used, although to truly test its effects the policy should be implemented on a real quadruped to test generalization.

Finally, we utilize intelligent reward shaping to achieve the desired behavior in the second stage. The penalty for not maintaining an athletic stance was reduced to encourage the quadruped to prioritize timing and spatial awareness over good jumping form. Moreover, sparse penalties were introduced for collisions with the obstacle, reinforcing the importance of proactive avoidance strategies.

Overall, this project underscores the power of coupling advanced RL algorithms with principled training strategies. This being said, we note that there is room for significant improvement in the quadruped's performance. Learning the timing of an optimal action is an inherently challenging problem for a MDP given issues like balancing time-horizon discount with the patience needed to account for slowly unfolding dynamic environments. Ideas to improve this component of the project are given in section 6.1.

6.1 Future directions

With more time, there are several different avenues we could pursue to iterate upon this project. Firstly, the obstacle in our experimental setup was guaranteed to pass through the quadruped's position, so one interesting problem to solve would be teaching the quadruped to jump *only* when an obstacle will directly interfere with its position. In addition, our scenario mimics that of a human jump roping, so exploring the quadruped's ability to perform multiple successive jumps – requiring not just proper jump timing but also a landing maneuver that enables the quadruped to reset for another jump – could be another scenario worth considering.

As far as increased performance in the current environment setup, we have several ideas. First, the reward-shaping for stage II could be improved to better encourage optimal timing of initial ground clearance. Second, RSI in its original format could be utilized by manually creating expert reference trajectories. Imitation learning strategies like DAgger could be implemented in a similar fashion. Finally, a predictive network for obstacle trajectory could be incorporated into the state space, potentially giving the policy better insight than obstacle properties alone.

Returning to our motivation, solving problems like these using curriculum-based reinforcement learning strategies could easily be applicable in real-world scenarios, where obstacle avoidance is often essential for quadrupedal and humanoid robots alike. Breaking down multi-stage problems like clearing obstacles enables agents to learn a variety of skills.

References

[1] Vassil Atanassov, Jiatao Ding, Jens Kober, Ioannis Havoutis, and Cosimo Della Santina. Curriculum-based reinforcement learning for quadrupedal jumping: A reference-free design, 2024.

[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, New York, NY, USA, 2009. ACM.

[3] Dohyeong Kim, Hyeokjin Kwon, Junseok Kim, Gunmin Lee, and Songhwai Oh. Stage-wise reward shaping for acrobatic robots: A constrained multi-objective reinforcement learning approach, 2024.

[4] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey, 2020.

[5] 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*, 37(4):1–14, July 2018.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[7] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.