Spherical Robot Control with Deep Reinforcement Learning

1st Matthew McGraw
University of Colorado, Boulder
Boulder, Colorado
matthew.s.mcgraw@colorado.edu

***Abstract*—Spherical robots are nonholonimic systems that can be quite complex and difficult to control using classical and modern control techniques. This paper discusses and implements Proximal Policy Optimization (PPO) on a spherical robot in different environments, show that capabilities that Deep Reinforcement Learning agents have when learning to control difficult systems. The robot used is a Sphero BB-8 and is simulated in Webots through different environments of varying difficulty.**

***Index Terms*—spherical robot; Proximal Policy Optimization; Deep Reinforcement Learning; Webots**

I. Introduction

The Sphero BB-8 robot is a spherical robot with 3 separate motors, one that controls the body's yaw, one that controls the body's pitch, and one that controls the yaw of it's head. For the purposes of this project, both the body pitch and body yaw motors are the main source of controlling how BB-8 moves. The body itself is weighted with motors and additional weights, which tend to make the robot wobble sporadically and difficult to control. The purpose of this project is to work towards creating a model that can control the BB-8 robot in numerous environments.

Fig. 1. BB-8 Robot Model [2]

In order to progress through the creation of a models that work for the BB-8, three separate problems have been designed to train the controller for the robot:

1) Train the agent to move the BB-8 robot as fast as possible in any direction.
2) Train the agent to have the BB-8 robot reach a designated goal while avoiding an obstacle.
3) Train the agent to have the BB-8 robot perform a small stunt such as moving through a hoop.

Simulating and training the agents on for each environment was done in Webots using Python, with the Deepbots [1] wrapper framework to simplify implementing a reinforcement learning agent that can control and train the robot. The Deepbots documentation and example code also provided an Actor-Critic Proximal Policy Optimization (PPO) [1] algorithm that was used and later modified to suit the needs of this project.

The Webots environment uses a physics engine to simulate the robots motion in real time, meaning that it can take quite a long time to train a reinforcement learning agent. Luckily, Webots also provides an option to run the simulation as fast as possible, for this project specifically, we were able to increase training time up to 75x normal speed thanks to this feature. The time-step can also be increased to allow the agent to gain more time in the environment, increasing the time-step too much can cause the physics engine to be unreliable, but a limit of 64ms was determined to be reasonable without compromising the integrity of the physics engine.

II. Background and Related Work

A. Deepbots

Deepbots is a wrapper framework for the simulation environment Webots, developed mainly for the purpose of implementing deep reinforcement learning algorithms to train robots. [1] discusses the limitations of Webots and the overhead it takes to implement Deep RL algorithms in the simulation environment, as well as limited attempts to implement packages for other simulation environments like Gazebo.

[1] goes on to provide examples of environments, including a pit escape environment in which the Sphero BB-8 is tasked with escaping a pit using their discrete PPO algorithm. The Deepbots wrapper framework is shown to be a useful and user-friendly tool that allows developers and researchers to implement Deep RL algorithms in Webots with relative ease.

B. Spherical Robot Motion Planning Algorithms

[3] discusses the motion of a spherical mobile robot similar to that of the Sphero BB-8. Similar to the BB-8, the robot model they use is nonholonomic due to it's nonintegrable rolling constraints, making it much more difficult to implement motion plannings for the system. [3] goes on to discuss their geometric path-planning algorithm that uses the path traced

by the spherical robot. Their simulations in MATLAB show that the algorithm they've developed is an efficient method of motion planning for spherical robots, but it is difficult to say if it would translate well to a simulated environment like Webots, or a real-world robot.

C. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is used throughout this project and was created by the developers at OpenAI. [4] discusses the basis on which PPO is formed and how it builds and improves upon existing reinforcement learning methods. They mention the limitations of current RL algorithms such as DQN for failing on simple tasks, vanilla policy-gradient methods for having poor robustness, and trust region policy optimization for being overly complex and difficult to implement [4] . The PPO algorithm uses 'clipped probability ratios' to create a conservative estimate of how to policy is performing, optimizing the policy by alternating between data from the policy and optimizing over the sampled data for a small number of epochs [4] . Moving on to show an implementation of PPO versus several pretuned algorithms in continuous physics-based environments, PPO outperforms a majority of the pre-existing algorithms. [4] shows that PPO is more reliable and easier to implement than other RL algorithms.

A real-world implementation of PPO can be seen in [5], where a UAV with a depth camera is first trained in Webots to create a motion plan while avoiding obstacles. [5] creates environments with obstacles in randomized positions for the drone to navigate through and the actor-critic agent to learn from. After training, the UAV agent is tested against previous motion planning algorithms on six separate environments of increasing difficulty, showing that it has a higher or equal success rate and better safety cost compared to previous motion planning algorithms. Taking it a step further, [5] then uses the trained agent on a real world UAV with a depth camera, showing that the trained RL agent works in real, physical scenarios when transferred from a simulated environment.

[6] builds upon the actor-critic PPO (AC-PPO), creating a new algorithm called Queue Batch Sampling Actor-Critic PPO (QAC-PPO) implemented on a bipedal robot in Webots. The QAC-PPO algorithm uses higher-value samples that are stored in memory to train on, with the idea that it will improve the training performance of the agent. [6] used the Nao robot in Webots with both the QAC-PPO and original AC-PPO and compared the training results to see if the QAC-PPO outperforms AC-PPO. The results in [6] show that QAC-PPO earned higher rewards 20,000 episodes earlier than AC-PPO, and was able to travel around 20% further than the AC-PPO agent. Given that the Nao robot has 12 degrees of freedom in it's legs alone [6], it's easy to see that either QAC-PPO or the original AC-PPO algorithms are extremely powerful when it comes to training agents on robotic systems.

D. Inside the Sphero BB-8

In the video "How the BB-8 Sphero Toy Works", a physical version of the BB-8 is opened op to show it's internal components and how it works [7]. Inside the main sphere of the body, there are two main motors with wheels that can drive and rotate the robot, with a counter-weight that keeps the wheels close to the ground. There is no shown stabilizing mechanism in the robot, which is discussed on how this might the system difficult to control [7]. Knowing that there is limited systems in the robot, it will be interesting to see how the PPO agent trains and learns to control it's motion through various different environments.

III. Task 1 - Move Quickly in Any Direction

A. Problem Formulation

For this initial task, the environment consists of a 100 meter X 100 meter flat terrain with walls on each side to keep the BB-8 robot inside, shown in Figure 2. The starting position for the BB-8 is the center of the environment, so it can learn to move in any direction as fast as possible. The observation space consists of the x and y positions, as well as the axis-angle values for the x, y, and z axes and angle of rotation. Each of these parameters are pulled from the supervisor in Webots at each time step.

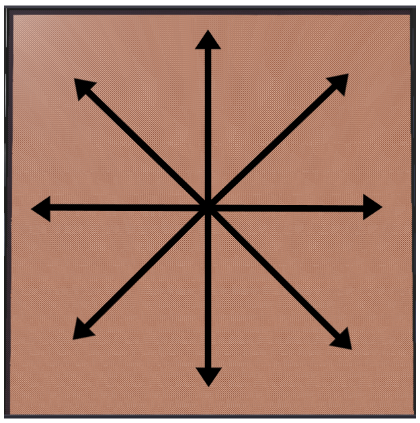
Fig. 2. Task 1 Environment, 100m x 100m Arena

The action space for this environment consists of five discrete actions:

- All Motors Off
- Increase Pitch Motor Speed
- Decrease Pitch Motor Speed
- Increase Yaw Motor Speed
- Decrease Yaw Motor Speed

Both the pitch and yaw motors have a maximum speed of 8.72 (positive & negative directions) in Webots, and the attenuation of the motor speeds is set to one-tenth of the maximum. When an increase/decrease in the motor speed is selected as the action, either the pitch or yaw motor speeds are increased/decreased by an amount equal to the attenuation.

The reward function for this environment is based entirely on the distance the BB-8 has traveled from the starting position

(0,0). Using this method as the reward function incentivizes the agent to move further and further away from the starting point. Throughout training the agent will observe the rewards received, observe the current state and action, and update the transition probabilities based on those parameters.

B. Solution Approach

The agent trained in this environment for a maximum of 10,000 episodes, with a maximum of 10,000 steps per episode. The episode terminates once the BB-8 reaches a distance of 49 meters away from the origin. Whenever the environment is reset, the BB-8 robot is moved back to the starting position (0,0) and original rotation (0,0,1,0).

At each step, the agent observes its translation and rotation values via the `get_translation()` and `get_rotation()` functions. The values returned are then normalized from their maxima to within a range of -1 to 1. Normalizing the parameters within these bounds is important, as the agent may have a more difficult time determining which parameters are more important if some have a much higher value than others (such as a position value of 30 compared to a rotation value of 0.5).

Originally the reward function was based solely on the starting position (0,0), with the idea that it would incentivize the agent to move away from the origin as quickly as possible. After a few attempts with thousands of episodes, it was observed that rather than moving further away from the origin, the agent preferred to drive in circles an continue to gain rewards rather than terminating. Seeing as how terminating the environment would result in a much lower reward, this made sense that the agent would prefer to keep the environment active. One possible way that this issue could be resolved would be to remove the termination of the environment and allow the agent to continue to gain rewards once its furthest away from the origin.

Rather than removing the terminating state from the environment, the reward function was altered to subtract the maximum distance to the terminating state, 49 meters from the origin (50 meters would be clipping the wall). By altering the reward function in this manner, if the agent stays closer to the origin, it continues to collect a negative reward. Only by moving away from the origin as quickly as possible and terminating can it achieve an optimal reward. A time-step of 48ms was used rather than the original 16ms to provide more training time for the agent.

After training was completed and the appropriate behavior was observed, the reward threshold of -125 was determined. In order for the agent to be considered fully trained in this environment, the average reward was taken over the last 100 episodes during training, and if the reward was greater than the threshold of -125, the environment was considered solved.

Aside from the addition of logging scores and loss values to tensorboard, the original PPO algorithm from Deepbots [1] was used, where both the actor and critic had a single hidden layer of 10 neurons, both using a ReLu activation function.

C. Results

After training for nearly 2300 episodes, the agent was able to learn an appropriate policy for moving as fast as possible in any direction, achieving a score of -125 or greater, shown in Figure 3.

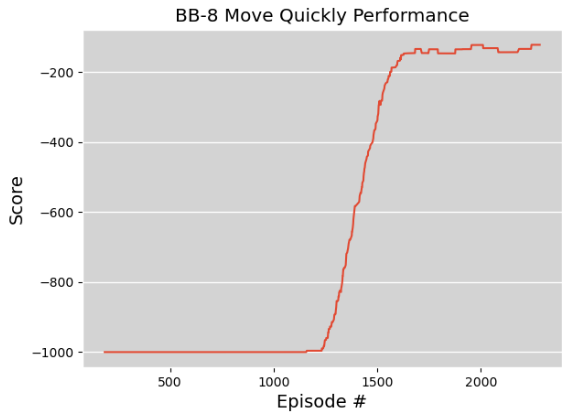

Fig. 3. Moving Quickly Training Performance

The fastest way that the bot could reach the terminal state without a controller would be to simply set the pitch motor speed to it's maximum of 8.72, by doing this the fastest time the motor could reach the terminal state was found to be 23.273 seconds. Using the controller that was created using the agent, the time to reach the terminal state was 23.280 seconds, which is only 0.03% slower than the maximum.

Figure 4 shows the robot moving directly from the origin to the terminal state 49 meters away:

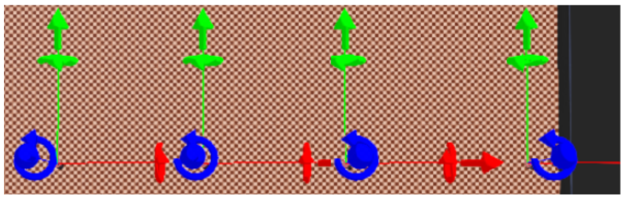

Fig. 4. BB-8 Moving to Terminal State

This problem was able to be solved in a relatively short amount of episodes, most likely because it doesn't require a large amount of control to reach to proper reward. The best policy for this problem would be to simply always move forward or always move backward. The next task will require a much higher degree of control, as the agent will need to determine where it is and using varying actions to reach the goal while avoiding an obstacle.

IV. Task 2 - Avoid Obstacle and Reach Goal

A. Mistake of Starting with a Challenging Environment

Originally, the environment in Figure 5 that was created for this problem was far too challenging for the agent to learn in a reasonable amount of time.

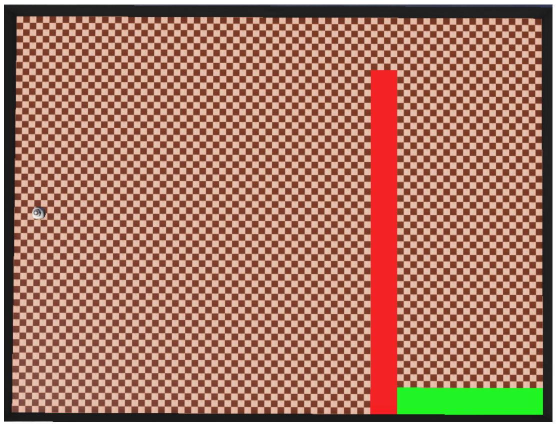

Fig. 5. Overly Challenging Goal-Oriented Environment

In the environment above, the agent is supposed to learn to avoid the large red line (obstacle) and reach the green line (goal) behind it. When attempting to train for this environment, the agent wasn't able to make it's way around the obstacle enough time's to reach the goal, and began to associate the y-axis of the obstacle with a negative reward, keeping it from exploring further.

The environment was far more challenging to solve than expected, with the assumption that since the agent was able to solve Task 1 relatively quickly, it should only be slightly more challenging to solve this environment. Due to the difficulty that this environment posed, a simpler environment was designed for the agent to solve, with the hopes of attempting to more challenging environment afterwards.

B. Problem Formulation

The simpler environment in Figure 6 is a much smaller 8m x 12m arena, where the robot is placed on one side of a small obstacle, and it must reach the goal on the other side. The state-space was discretized somewhat by rounding the position to the nearest integer and rounding the angle to the nearest tenth. The idea behind this is that if the state and observation space are much smaller, it should be easier for the agent to learn an appropriate policy.

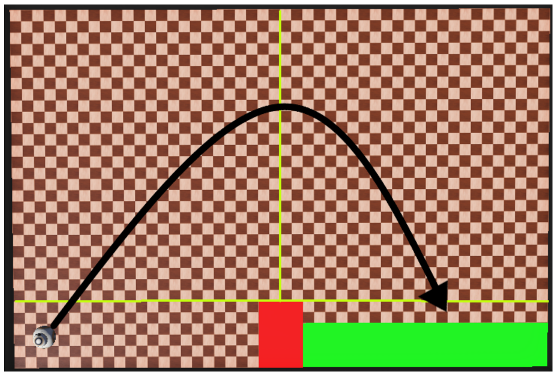

Fig. 6. Simpler Goal-Oriented Environment

In addition to this discretization of the state-space, the action space was decreased as well by removing the reverse pitch motion and motor attenuation. This now limited our action space to 4 actions:

- All Motors Off
- Pitch Motor Max Speed ('forward')
- Yaw Motor Max Speed ('clockwise')
- Yaw Motor Reverse-Max Speed ('counter-clockwise')

With these new set of actions, rather than having to learn the environment for 40 different pitch and yaw motor speeds, the agent now only has to learn the environment with 3.

In the first attempt of solving this environment, the agent can collide with the outer walls without getting a negative reward, one of the few negative reward comes from touching the red obstacle, giving a reward of `-100`. The reward function is set up to provide multiple ways of earning rewards, the main reward being `+100` if the goal is reached. In the more challenging environment mentioned before, it was extremely difficult for the agent to learn given sparse rewards, so smaller rewards are given to encourage the agent to approach the goal.

The first smaller reward is simply by moving a reasonable distance from step to step, so that the agent doesn't just rotate in place to avoid touching the obstacle. Just like in the previous task, the reward is given by the distance traveled, but instead of the distance from the starting point, it's the distance from the previous step. If the distance traveled is less than 0.0001 meters from the previous position, a negative reward of `-0.1` is given, otherwise the reward is `+distance/10`. The additional way the agent can get rewards is by passing through the checkpoints, shown by the vertical and horizontal lines coming from the obstacle in Figure 6, along with an additional checkpoint for the goal. Once the agent passes through these checkpoints, it receives a reward of `+10` for each checkpoint, making its way closer to the goal.

C. Solution Approach

The agent trained in this environment for a maximum of 100,000 episodes, with a maximum of 2,500 steps per episode. The only terminal states in the environment are the obstacle or the goal, so if the agent touches either, it obtains the corresponding reward and the environment resets.

Just like in the previous task, the `get_translation()` and `get_rotation()` functions are used to obtain the agents position and rotation in the environment to provide as an observation. Additionally, a list of passed checkpoints is passed as an observation as well for whenever the agent passes through a new checkpoint. If the agent has passed through a checkpoint, a value of 1.0 for that checkpoint will be passed, otherwise a value of 0.0 is passed. Once the agent passes through a checkpoint, that value will remain 1.0, and never switch back to 0.0. The purpose behind this is to provide information of the progress it's made to the goal, rather than just depending on the position and rotation data alone.

Since it is given that the cumulative reward after reaching the goal will be 140 plus the additional rewards for moving through the environment, a reward threshold of 142 was chosen. In order for this environment to be solved, the average reward of the latest 300 episodes was taken, and if the average is greater than the threshold of 142, the environment was considered solved.

D. Results

With the current implementation, the agent was able to solve the environment in 38,000 episodes with an undesired but predictable outcome. Since the Sphero BB-8 has no stabilizing mechanism, it can be extremely difficult to control, which is why the agent ended up using the wall for added stability. Figure 7 shows the steps that the agent takes to reach the goal, using the wall as a guide to the goal. *Unfortunately this behavior was difficult to replicate and I was unable to obtain the proper learning performance plot for this agent.

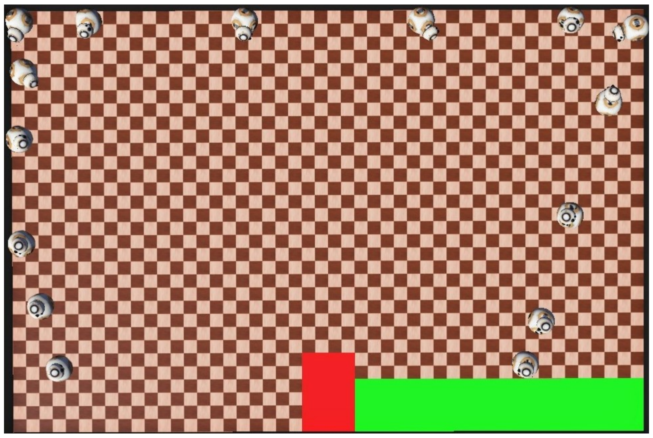

Fig. 7. BB-8 Agent Using Wall for Stability

Following the wall was a useful but undesired outcome used by the agent, so to eliminate this behavior a negative reward was added to the implementation for coming in contact with the walls, similar to the negative reward given for touching the obstacle. Since there are no sensors for obstacle detection on the Sphero BB-8, its distance from each wall is given as additional observation data. Additionally, the step rewards were normalized between the range of -1.0 to 1.0 with the hope that it will improve learning stability and not get stuck in local maxima. With this new implementation using an added penalty and new observation data, the PPO agent was able to solve the environment in just under 9,000 episodes, never making contact with the walls and reach the goal.

Figure 8 below shows the path of the agent in the solved environment, with the light blue arrows show its direction of translation, and the green/red/dark blue arrows represent the local x/y/z axes of the BB-8 robot respectively. The agent avoids contacting the wall the entire time, but it's motion is not as efficient as it could be, rather than turning towards and moving to the goal, it moves to the top wall turning before contact, then moves to the goal.

The learning performance of this new agent can be seen in Figure 9. The spikes of increased scores show how the agent was exploiting the information it trained on in the environment,

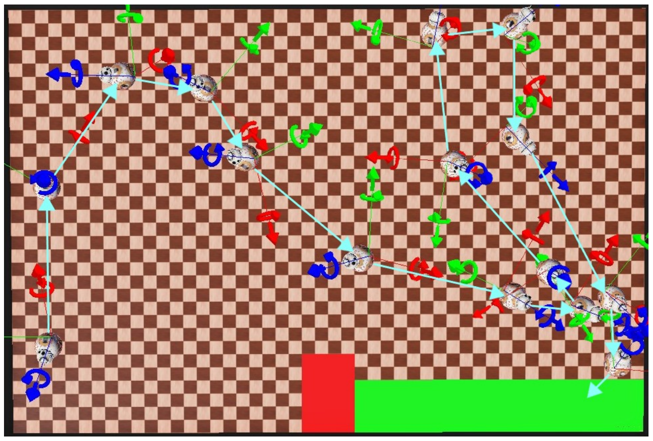

Fig. 8. BB-8 Agent Reaching Goal W/O Touching Wall

while the low dips most likely represent its exploration of the environment. After gathering enough information to form a reasonable policy, it was able to solve the environment around episode 8,700.

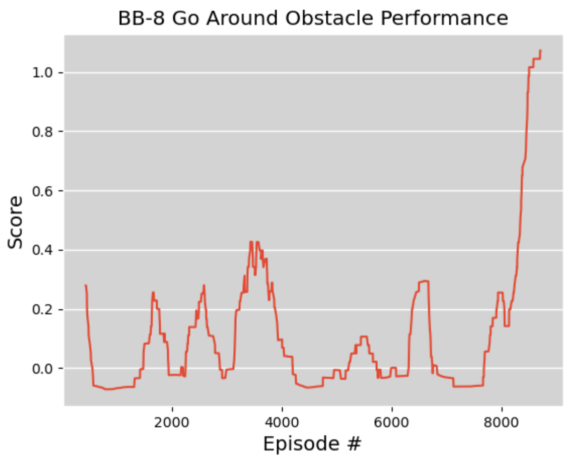

Fig. 9. BB-8 Go Around Obstacle Learning Performance

V. Task 3 - Perform a Stunt

A. Problem Formulation

The final task for this project is to train the BB-8 robot to perform a stunt, in this case, jumping through a hoop. The hoop was designed in Fusion 360 then imported as a mesh object into a new 10m x 3m flat environment. The hoop itself is 1.25m in diameter, whereas the BB-8 robot is approximately 0.65m in height, allowing enough room for the robot to pass through. The BB-8 robot itself does not have the ability to jump, so a "jump" action was added by using the `addForce()` method of the Webots `Supervisor()` class. As the hoop was designed to be 0.5m off the ground, a few different forces were tested to determine a reasonable jump height. A force vector of [0, 0, 3000] N allowed the robot to jump approximately 0.7m from the ground and was determine to be a sufficient amount of force for this problem.

Similar to the previous task, the state and actions spaces were kept in a discrete form for the robot.Additional information from the body accelerometer and body gyro were used in the observation space, as well as the z position of the robot for it's height from the ground plane. The starting position of the robot is [-4.5, 0, 0] and it's rotation is [0, 0, 1, 0] so it is pointed towards the hoop initially.

The reward function for this problem is relatively simple compared to the previous task, where the reward at each step is a negative reward given by the distance from x = 4, and divided by 100 to reduce the weight of the reward. Using this as a step-by-step reward will driving the robot in the positive-x direction towards to hoop. An additional reward of +100 is given if the robot goes through the hoop and lands on the other side, which is also the only terminal state of the environment.

B. Solution Approach

The agent trained in this environment for a maximum of 100,000 episodes, with a maximum of 1,000 steps per episode. The only terminal state is when the robot passes through the loop and reaches the other side, in which the environment resets and the robot is placed back at it's original position. The maximum pitch speed for this environment was reduced to half of it's true maximum (8.72) , and the maximum yaw speed was reduced to a quarter of it's true maximum (8.72) in order to reduce instability in the system.

Ideally the robot will head directly for the hoop and jump through to minimize the amount of negative rewards it accrues. Since we know that the terminal state provides a reward of +100 and the maximum penalty at each step is approximately -0.09, with the desire to have the robot to complete the task in under 6 seconds, the environment will be declared solved if the most recent 100 episodes have an average reward greater than 90. Initially the average reward requirement was set to be an average of 99 or greater, but when the agent first solved the system, the maximum reward it was able to get was roughly 91, so the average was then reduced to 90 for future training.

Similar to the previous task, the neural network used for this problem consists of two hidden layers, but uses 64 neurons for the first hidden layer and 128 neurons for the second hidden layer. The input is the observations obtained from the environment, and the output is a softmax probability distribution over the available actions.

C. Results

As stated previously, the requirement for the environment being solved was higher than the maximum possible reward, but even so the agent was still able to find an optimal policy for the environment after training for 100,000 episodes. Figure 10 shows the training performance of this agent, where it can be seen that the agent found an optimal policy after around 14,250 episodes of training. Since it still had not met the initial required average of 99, it continued to train and solidify the policy it had learned up to episode 100,000.

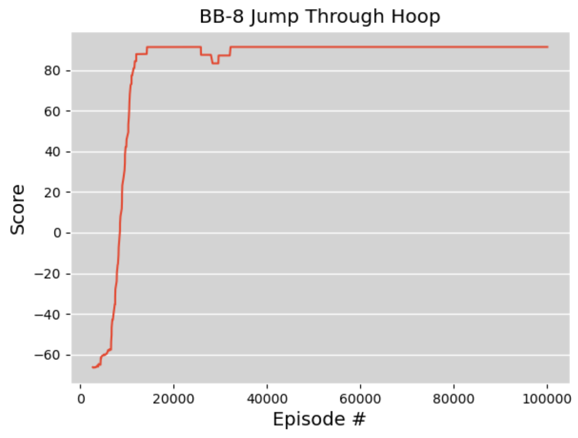

Fig. 10. BB-8 Jump Through Hoop Training Performance

Figure 11 shows snapshots of how the BB-8 robot moves through the environment. The robot moves directly to the hoop as intended, performing a small front flip on the way for some additional flare, before performing a second front flip through the hoop and into the other side.

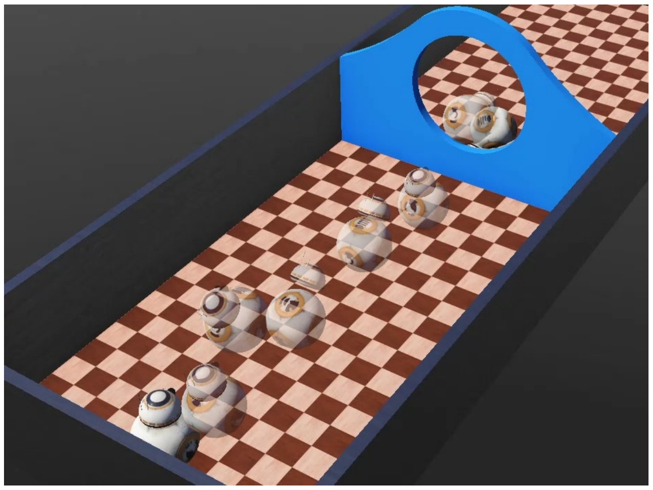

Fig. 11. BB-8 Jumping Through Hoop

VI. Conclusion

Talk about using QAC-PPO in future versions

In this paper, training of the Sphero BB-8 robot was proposed and implemented through various different environments using Proximal Policy Optimization. The purpose of this was to show the ability that deep reinforcement agents have in learning to control even some of the most difficult systems, such as a nonholonomic spherical robot with limited sensor input. It was shown that PPO was a useful reinforcement learning algorithm that enabled the agent to deal with multiple different environments, ranging from quite simple to very challenging. In the future, it would be interesting to train on even more challenging environments, such as obstacle courses where the robot would have to learn and perform multiple stunts. It would also be nice to implement the QAC-PPO algorithm shown in [6], and to observe if it has as significant improvement in training the agent for this system. It is also more than likely that the deep reinforcement learning controllers created during this project are not as robust as they could me. Creating more robust RL-based controllers that can account for different scenarios and starting positions in their environment is probably the next milestone for this project.

VII. Release

The author grants permission for this report to be posted publicly. The materials used in the creation of this project can be found in the GitHub repository BB-8_STUNTS, https://github.com/mcgrawm14/BB-8_STUNTS.

References

[1] M. Kirtas, K. Tsampazis, N. Passalis, and A. Tefas, ”Deepbots: A Webots-Based Deep Reinforcement Learning Framework for Robotics,” in *Artificial Intelligence Applications and Innovations*, Springer International Publishing, 2020, pp. 64–75, ISBN 978-3-030-49186-4.

[2] Cyberbotics. *Webots User Guide: Sphero’s BB-8*. Cyberbotics, 2023. https://cyberbotics.com/doc/guide/bb8

[3] V. A. Joshi and R. N. Banavar, ”Motion analysis of a spherical mobile robot,” *Robotica*, vol. 27, no. 3, pp. 343-353, 2009. doi: 10.1017/S0263574708004748.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[5] H. I. Ugurlu, X. H. Pham, and E. Kayacan, “Sim-to-Real Deep Reinforcement Learning for Safe End-to-End Planning of Aerial Robots,” *Robotics*, vol. 11, no. 5, article no. 109, 2022, pp. 1-12, doi: 10.3390/robotics11050109.

[6] C. Zhang, P. Zhong, Z. Liang, M. Liu, X. Wang, and J. Liu, ”Learning to walk with biped robot based on an improved proximal policy optimization algorithm,” in *International Conference on Intelligent Equipment and Special Robots (ICIESR 2021)*, vol. 12127, Q. Zhang and Z. You, Eds. International Society for Optics and Photonics, 2021, p. 121271B. doi: 10.1117/12.2625340.

[7] Adam Savage’s Tested, ”How the BB-8 Sphero Toy Works,” YouTube, 2015. [Online]. Available: https://www.youtube.com/watch?v=5FHtcR78GA0. [Accessed: May 6, 2023].