

Table of Contents

Table of Contents	1
Executive Summary:	2
Background	3
Significance of robotics	3
Challenges in robotics	3
Applications of Reinforcement Learning	4
Project Goals and Previous Work	4
Deep Reinforcement Learning Background	6
Project Contributions	6
Implementation of Reinforcement Learning	6
Existing reinforcement techniques and its limitations	6
Proximal Policy Optimization	7
Implementation in Simulation	7
Policy Gradient	7
Efficient Online Reinforcement Learning with Prior Data (RLPD)	8
Single Policy Formulation	8
Challenges of Multi Stage Policy	9
Why Multi Stage Policy	9
Design and Methodology	10
Design	10
Multi Stage Reward Policy	10
Stage 1: Tracking Motion	10
Stage 2: Kicking Motion	11
Stage 3: Goal!	12
Methodology	12
Reward gradient	12
Status flag	12
Placeholder	13
Results and Performance	13

Executive Summary:

Over the years, the robotics field has made notable advancements in quadruped robots, often relying on classical model-based approaches for controller design. Yet, as robotics applications expand to include more complicated tasks like self-driving cars and intricate object manipulation, these methods demand detailed modeling, posing challenges for online deployment due to computational complexity. Moreover, in scenarios involving dynamics and unpredictable elements, such as deformable objects like soccer balls, modeling the dynamics becomes impractical. As a result, reinforcement learning (RL) techniques have been getting more and more attention but advancements in robotics are still limited. Our research, we aim to highlight the potential of RL techniques in complex robotics applications. In particular, we showcase our work of training a quadruped to be able to kick a soccer ball into the desired goal

Imagine having a robot that can kick a soccer ball into the top corner of a soccer goal by learning from experience. This project's objective is to develop a single reinforcement learning (RL) framework that performs multi-stage tasks to enable a quadruped robot to move, kick and score. The project aims to showcase the versatility of reinforcement learning application applied to the physical robot.

This project addresses multiple challenges of training quadrupedal robots to perform precise shooting skills in real-world scenarios using deep reinforcement learning (RL). This project's framework composed of three main components (a) a tracking stage where the robot learns how to walk to the ball in a short period of time (b) shooting stage, as the robot is near the vicinity of the ball it performs the kicking motion to propel the ball forward to the target position (c) Goal!, the final stage and objective of the project, which aims to maintain the stability of the robot while waiting to compute the accuracy of the robot's kicking motion to the target. This proposed framework is implemented on a robot developed by Unitree Robotics, Go1. A soccer playing robot offers a proof of concept for robots navigating the complexity of real-world environments.

Background

Significance of robotics

Robotics has become pervasive across various industries, offering solutions to address a wide array of common challenges encountered in our daily lives. This interdisciplinary field encompasses a multitude of techniques and applications that can be leveraged to realize diverse design objectives.



Figure 1

Figure 2

Figure 3

Figure 1 illustrates the advantages of employing robotics in manufacturing. This technology is especially useful for tasks requiring repetitive actions, as robots can deliver precise measurements consistently, scale production at a much faster rate surpassing that of human capabilities. Additionally, robotics applications have extended into the realm of fulfillment operations as shown in Figure 2. Here companies are increasingly relying on robotics to streamline processes such as product reorganization for loading into trucks, palletizing, depalletizing and many more. Meanwhile, Figure 3 highlights how companies are harnessing various engineering technologies, including artificial intelligence, computer vision and controls to develop autonomous vehicles capable of driving without any human intervention. However with that said, there are many challenges inherent in the widespread adoption of robotics across industries.

Challenges in robotics

Although there have been significant strides in the field of robotics, often through the utilization of traditional model-based techniques for designing controllers, as robotics application expands to tackle more complex systems, the need to have accurate dynamic models becomes significantly more difficult. In addition to this, classical controls are highly sensitive to the environment they are in. For instance the dynamics for a robot to walk on a flat normal ground is very different from walking on an icy road. As a result, in this project we are utilizing an alternative approach known as deep reinforcement learning techniques to highlight its benefits and limitations in the development of the robotics field. Deep reinforcement learning provides a bridge between classical controls because it doesn't require the need to design an accurate model of the environment; instead it gives rewards to agents that are behaving as desired. With

this in mind, we will be giving thorough analysis of common vocabularies and definitions related to deep reinforcement learning.

An algorithm called an artificial neural network, inspired by neuronal connections in the human brain, consists of cascading an input layer, intermediate layers called hidden layers, and an output layer. Having a large number of hidden layers creates a deep neural network. These deep networks can be used for inference tasks such as predicting objects in an image or optimal trajectories for a robot's path.

Some examples of classical robot path planning algorithms include graph-based methods such as A* and Rapidly-exploring Random Trees. Some sampling based planning methods include Monte Carlo Tree Search (MCTS) and Monte Carlo Localization (MCL).

In order to get the robot to take actions in accordance to inputs, the robot must utilize a method of feedback to nudge actions in a certain direction. This feedback involves constructing an error value that denotes the difference between the current action and the desired action. Prior to the advent of deep learning, roboticists utilized classical control theory to develop these feedback loops. Some examples include developing control loops in the frequency domain using laplace transforms and using PID (proportional-integral-derivative) controllers. Recently, a paradigm called **reinforcement learning** has been used to develop algorithms to dictate the robot's actions based on the environment's current state.

Applications of Reinforcement Learning

Reinforcement learning is a powerful framework that has been applied to a variety of domains from games such as Go and Atari to natural language processing recommendation systems. Famously, the deep RL policy AlphaGo defeated Go world champion Lee Sedol. In a robotics context, reinforcement learning has been utilized for the development of autonomy of systems such as self-driving vehicles, and for robotic manipulation tasks such as pick-and-place robots.

Project Goals and Previous Work

This project uses deep reinforcement learning (deep RL) to control the actions of a quadruped Go A1 robot to kick a soccer ball into a goal.

We use python frameworks such as PyTorch for algorithm development and IssacGym for simulation. Robot Operating System (ROS) is used to simplify the development and integration of complex robotics systems by providing a common interface and standard communication between nodes such as perception, robotic control, etc. In particular, we use the ROS Gazebo simulation environment to transfer the policy to the physical robot.

In this instance, the experiments will feature the four-legged robot developed by Unitree Robotics, the Go1, highlighting its numerous advantages in design and functionality. Firstly, a quadrupedal robot offers enhanced stability during kicking motions, as it swings one of its legs to propel the ball forward. This stands in contrast to bipedal robots, where the center of mass may pose challenges during kicking movements. Lastly, there exists numerous research on quadruped robots engaging in various tasks such as shooting, dribbling across different terrains, and demonstrating goalkeeping skills. This provides our team to be able to start with existing frameworks instead of building it from scratch.

Previous work on this problem includes *Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills using a Quadrupedal Robot* [1]. It uses a hierarchical framework with separate policies for planning and control of the soccer shooting motion. The planning policy generates desired **Bezier parameters** and the control policy utilizes these parameters to generate motor inputs for the quadruped. Bezier parameters control the shape of a continuous path. Another previous work, *Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning* [2], presents a similar RL technique to the policy we train. It enables quadruped robots to perform soccer goal-keeping tasks in the real world.

Our work uses a single policy based on existing algorithms such as Proximal Policy Optimization (PPO) [3]. We build upon this by adding a multi-stage reward function. The three stages for this reward function are 1) the robot walking to the ball 2) kicking the ball and 3) minimizing discrepancies between simulation and reality. In order to generate the gait of the legged robot, we use a method of locomotion gait called a **Central Pattern Generator** from *Learning Free Gait Transition for Quadruped Robots via Phase-Guided Controller* [4]. Central pattern generators generate rhythmic signals in the absence of rhythmic input. This enables the robot to smoothly walk to the ball, which is the first phase of the project. Because reinforcement learning requires a substantial amount of data, simulations provide an effective platform for robots to learn. Unlike learning directly, in the physical world, simulations allow robots to engage in various actions with multiple agents attempting to acquire the same skills. However, the physical world presents limitations such as hardware constraints, feasibility concerns and environmental factors.

Conversely, certain environmental factors are inherently difficult to replicate in simulations. These include nuances like the friction between the ground and the ball, the deformation of the ball upon being kicked by the robot, sensor measurement noise and many more. Training the robot in the physical world enables us to refine both offline and online data, ultimately leading to a deeper comprehension of the real world environment. This is where Reinforcement Learning with Prior Data (RLPD) comes into play which is an important algorithm for fine-tuning the performance of the robot [5]. It combines the offline data we collected from the pretrained model with the online data generated by the updated model, which can be used to fill the gap between simulation environment and real environment. Moreover, since it is based on the Soft-Actor-Critic (SAC) algorithm, the exploration property makes it more suitable for fine-tuning work.

This project in the Hybrid Robotics Lab tackles the problem of using deep reinforcement learning to control a robot to shoot a soccer ball into the goal. We look into robotics sub-problems such as developing methods for training the robot control policy in simulation, transferring this policy from the simulated environment to the real world, and dealing with perception tasks for the physical robot to acquire information from its environment. Improvements on these tasks can provide insight into more generalized robotics problems and also show the potential of applying learning based methods to robotics.

Deep Reinforcement Learning Background

Deep learning is an inference method based on an algorithm called neural networks used to make predictions on data. A **neural network** is an algorithm inspired by neurons in the human brain that can perform a series of computations for prediction. It consists of an input layer, hidden layers, and an output layer. Non-linear functions are introduced after each layer and a technique called backpropagation is used to finetune the weights of each of the layers. “Deep” in deep learning comes from using a network with a large number of neural network layers. This technology has been rapidly developed since its popularization in 2012 with the release of AlexNet, a model used to make classifications on image data.

Deep learning has been effective when applied to a wide range of domains, from analyzing brain signals to making breakthroughs on physics problems. The most recent wave of deep learning advancements have focused on foundational generative models, which are large end-to-end networks that can produce a wide range of content such as text, images, and videos. Popular AI tools include ChatGPT, a chatbot that generates answers based on a generative model called GPT, and Sora, a text to video model. The high quality of these results have brought up concerns in the wider community about AI potentially negatively impacting jobs.

These algorithms can also be applied to robotic tasks for all three parts of the robotic stack, from making predictions of the world based on sensor data, to planning generating potential trajectories for the robot, to controlling the actions of the robot. In particular, a subset called **deep reinforcement learning**, is a paradigm in which an **agent** interacts with the world by taking **actions** and accumulating information through a **reward** value based on the quality of its action, which can be used to generate a robot’s actions. A neural network is used to predict the robot’s actions based on the current state of the environment and is referred to as the agent’s **policy**.

Project Contributions

Implementation of Reinforcement Learning

Existing reinforcement techniques and its limitations

In recent years, various methodologies have emerged for reinforcement learning using neural network function approximators, with common techniques including deep Q-learning, “vanilla” policy gradient methods, and natural gradient methods. While effective in general, these techniques have limitations in scalability, data efficiency and robustness, often requiring careful hyperparameter tuning.

Q-learning algorithms can fail on simple problems and lack comprehensive understanding. Vanilla policy gradient methods suffer from inadequate data efficiency and robustness. Trust region policy optimization (TRPO), though effective, is complex and incompatible with certain model architectures involving noise or parameter sharing. Leaving the proximal policy optimization to be favorable for our complex system that requires a huge training set to train the robot to do a certain set of actions.

Proximal Policy Optimization

Proximal policy optimization (PPO) addresses these limitations by aiming for improved data efficiency and reliable performance similar to TRPO. PPO's primary objective is to optimize policy functions effectively, particularly in environments with high-dimensional state and action spaces. It achieves this through techniques such as clipped surrogate objective functions and multiple epochs of updates, providing a more accessible and efficient alternative for reinforcement learning tasks especially for this project.

PPO provides an easy framework for an individual to use for their highly complex system such as a quadruped. While TRPO involves computing and enforcing a constraint on the size of policy updates to ensure stable training, PPO achieves a similar effect by instead clipping the surrogate objective function. Additionally, TRPO requires tuning additional hyperparameters related to the trust region constraint, such as the KL divergence threshold.

This paper considers PPO as the mainframe to design our offline policy due to the fact that it has a faster convergence rate and its simplicity.

Implementation in Simulation

Policy Gradient

The policy gradient methods in reinforcement learning work by directly optimizing the policy parameters to maximize the expected rewards. They typically use a gradient estimator to compute the gradient of the expected return with respect to the policy parameters which is used

to update the policy through stochastic gradient ascent. The most commonly used gradient estimator is the policy gradient estimator which has the form of:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

where:

1. $\pi_{\theta}(a|s)$ is the policy parameterized by θ , which specifies the probability of taking action a given s
2. $\hat{A}_t$ is an estimator of the advantage function at timestep t

The policy gradient estimator computes the gradient of the logarithm of the policy multiplied by the state action value function. Intuitively, this gradient estimator encourages actions that have higher values according to the state-action value function (Q-function) and discourages actions with lower values.

Once the gradient is estimated using this formula, it is plugged into a stochastic gradient ascent algorithm to update the policy parameters. The policy parameters are iteratively adjusted in the direction of the gradient to maximize the expected return.

Efficient Online Reinforcement Learning with Prior Data (RLPD)

While PPO demonstrates efficacy in simulation environments, its performance is really dependent on its access to a lot of data. This makes it difficult as we transition to the real world situation as obtaining the data will require a lot more effort and work increasing the complexity of the project even more. To tackle these challenges as we shift our experiments from offline to online data, we've explored a hybrid approach called reinforcement learning with prior data (RLPD) where it essentially combines both online and offline information. This combination allows us to make modifications to our existing off-policy reinforcement learning techniques, thereby striving for more consistent and reliable performance.

The RLPD approach offers the ability to simply apply existing off-policy methods that leverage offline data while learning online. RLPD encapsulates design choices that are beneficial for our project as it can incorporate prior data and online data together as their sampling strategy that is surprisingly effective. Moreover, to prevent overestimation which could result in training instabilities, RLPD refrains from extrapolating its values without constraints. Instead, it maintains adherence to predefined constraints throughout the process. Finally, RLPD uses sample efficient RL where it increases the efficiency of offline data without overfitting.

Single Policy Formulation

Traditionally, PPO is applied for training legged robots in single locomotion tasks, where the focus is on a specific set of actions, thereby requiring the design of a single reward function. *Reinforcement Learning for Quadruped Dribbling in the Wild* highlights a single policy by dribbling the ball with a single action task that is to control the ball's velocity and the position of the robot with the ball. *Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills Using a Quadrupedal Robot* propose a hierarchical framework that leverages multiple single policies including motion control policy that tracks robot motions, planning policy to decide on the desired kicking motion for the robot to follow and accurately shoot the ball and lastly an improvement policy that allows the robot to learn from the discrepancies between the real and simulated model. Real would mean that disturbances of the wind or friction between the ball and the floor exists. *Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning* also uses a hierarchical framework that enables quadruped robots to perform soccer goalkeeping tasks in the real world. In the paper, it introduces multiple control policies for different goalkeeping skills. This includes distinct behaviors for each policy which includes sidestepping, diving and jumping skills which are common goalkeeping motions that one can observe. Training individual skill-specific control policies avoids the challenge of training a single multi-task policy for all skills.

Challenges of Multi Stage Policy

Unlike single reward policies, multi stage reward policies involve multiple steps of reward functions that increases the complexity of this project. They involve multiple decision points where actions taken at one stage can have consequences across subsequent stages. Managing this complexity requires more sophisticated algorithms and decision making processes. In multi stage environments specifically, it is not very intuitive to determine which action or decisions that can lead to a specific outcome which is an issue known as “temporal credit assignment”. Additionally, multi stage reward policies often have delayed rewards, where consequences of an action might not be realized until several steps later. This makes it more difficult for the agents to learn as they must associate earlier tasks with differing rewards before they are penalized for their next action. On top of this, with multiple stages, the state space which comprises all the possible states the agent can be in can grow exponentially increasing the computational complexity of learning and decision making.

Why Multi Stage Policy

With all the challenges discussed regarding multi stage policy, this section answers the question to why we are choosing the multi stage policy instead of the single stage policy which was already done by *Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills Using a Quadrupedal Robot*. Multi-stage reward policies offer more flexible decision making compared to single stage policies. Realistically, the actions that we take in our daily lives consist of multi-stage actions rather than single policy actions. Having a one policy that encapsulates multiple stages of reward makes reward formulation much easier and generates more flexibility

to developers to produce numerous actions. Multi stage reward policies provide a framework for modeling and solving sequential decision-making problems, which are essential in many domains such as robotics, finance and game playing.

Design and Methodology

Design

Multi Stage Reward Policy

Building upon the insights provided in the previous section, the project underscores the importance of employing heterogeneous reward functions. These functions serve to specify distinct tasks or behaviors the robot needs to achieve to complete the full robot's operation. In this project, we have devised a three stage reward system tailored for the robot's performance, building upon the challenges discussed in *Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills Using a Quadrupedal Robot*. Rather than solely executing a predefined kicking motion along a bezier curve, our extended problem requires the robot to first navigate to the ball then execute a kick to accurately propel it to the desired target position while maintaining stability. The initial stage, termed the tracking stage, necessitates the quadruped to track the ball from a given initial condition and subsequently move towards it in preparation for the kicking motion. Stage 2 will allow the robot to perform the kick motion while stage 3 requires the robot to stay stationary while it processes the direction of the ball and learns whether that particular set of motion allows the ball to move towards the target position or not.



Stage 1: Tracking Motion

As stated briefly above, the first stage that the robot needs to perform is to be able to walk towards the ball. During the tracking stage, the rewards encompass various aspects crucial for successful navigation and tracking of the ball. These include:

1. **Reference motion reward:** incentivizes the robot to adhere closely to a predefined reference motion trajectory ensuring smooth and accurate movement towards the ball. Reference motion reward involves tracking reference motion generated by the CPG model[4], which defines the end position of each leg when a quadruped robot walks at the desired speed.
2. **Distance to ball reward:** encourages the robot to minimize the distance between itself and the ball, facilitating efficient navigation towards the target. It is used to measure the distance between the current robot position and ball position and return the reward that is inverse proportional to the distance.
3. **Tracking Speed Reward:** rewards the robot for maintaining an appropriate speed while tracking the ball, balancing between swift movement and stability. Tracking speed reward

is a measurement of how the robot current velocity vector (considering both direction and speed) aligns with the desired velocity, it will return the reward that quantifies the disparity between actual one and desired one.

4. **Tracking Yaw Reward:** Promotes alignment of the robot's orientation in yaw with the direction of the motion towards the ball enhancing tracking accuracy and efficiency. It is insufficient to only have a robot velocity vector to make the robot heading to the ball, velocity vector can only regulate the moving direction of the robot but cannot specify the direction of the heading. Thus, we also append the tracking yaw reward to make sure the robot can always head towards the ball, by returning the reward related to the angle difference between desired yaw and current yaw.

To align the observation space with the defined rewards, we construct observations that capture useful information about the state of the robot and the ball. These observations consist of the state of the robot including the position, orientation and joint position. It also contains information about the ball state including its initial position and its real time position after the robot kicks it. For stage1, since the robot hasn't approached the ball yet, the initial ball position and the real time ball position is trivially the same in this stage. Similar to the real time ball position, because the first stage doesn't take into account of the target position at all, the target position acts as a placeholder for that is consistent and time invariant value which will not affect the training performance of the robot at all time since it has no gradient to reward the robot behavior. Although they are not used in the current stage, it will matter significantly in other stages.

Stage 2: Kicking Motion

In the second stage of the task, the focus shifts to enabling the robot to kick the ball while maintaining its stability. To achieve this, several adjustments are made to the locomotion and manipulation aspects of the robot. The locomotion transition between stage 1 and stage 2 is that the desired velocity of the robot is adjusted from 0.5m/s to 0m/s to ensure it stops near the vicinity of the ball's position. Additionally, phase difference is used to regulate the legs' operation periodically and corporately so that each leg could do the reference motion alternately and make the whole body move forward. In stage1, we define the phase difference as $[0, \pi, \pi, 0]$, which means the FL and RR legs have the same phase, FR and RL have the same phase. The FR and RL pairs are half circle lagged then the FL and RR one. However, in stage 2, in order to make the robot stop entirely, rather than step in place, we set the phase of all legs to be zero and the swing height to be zeros as well.

The newly rewards for stage 2 are generated using:

1. **Ball Direction Reward:** Measures whether the robot successfully kicks the ball in the correct direction, determined by the unit vector pointing from the ball's initial position to the desired target position.
2. **Stand Still Reward:** Regulates the robot's joint positions to prevent excessive deviation from the normal posture, ensuring stability during the kicking motion.
3. **Stage 2 Reward:** provides a high reward to incentivize the robot to enter stage 2, signaling successful progression in the task.

Unlike stage 1, it is worth noting that there is no reference motion reward in this stage any more since it cannot guide the robot to learn the kicking behavior. Similarly, low-level rewards for kicking are not utilized; instead the robot learns through high-level rewards such as ball direction and the stand still reward. For the observation, the real-time ball position now is activated; it now contains the 3D(X,Y,Z) coordinates of the ball's center in real time. The target goal position is also activated, which contains the randomly assigned desired position that the robot should shoot the ball to.

Stage 3: Goal!

Finally in the third stage, after successfully kicking the ball, the robot needs to reset and wait until the ball stops or hits the target. The reward functions involve in this stage are:

1. **Ball Direction Reward:** Identical to before, because the ball heading direction really determines if the ball is going to hit the target or not, it is essential that one still keeps track of the ball direction even after the robot kicks it.
2. **Hit Target Reward:** The hit target reward quantifies the performance of the kick motion, which calculates the minimum history distance between the ball's real time position and the target. If the ball has ever passed the target, the minimum distance should be zero and the hit target reward function will return a high reward. If not, the function will return the reward inversely proportional to the minimum distance to reward the robot if it can even come close to the target.
3. **Stand Still Reward:** The stand still reward matters in this stage since the robot pose needs to be regulated even after shooting the ball.

It is worth noting that the hit target and ball direction reward should be weighted higher than other rewards as these are the factors that will determine whether the ball will accurately hit the target or not. The observation is identical as that in stage 2.

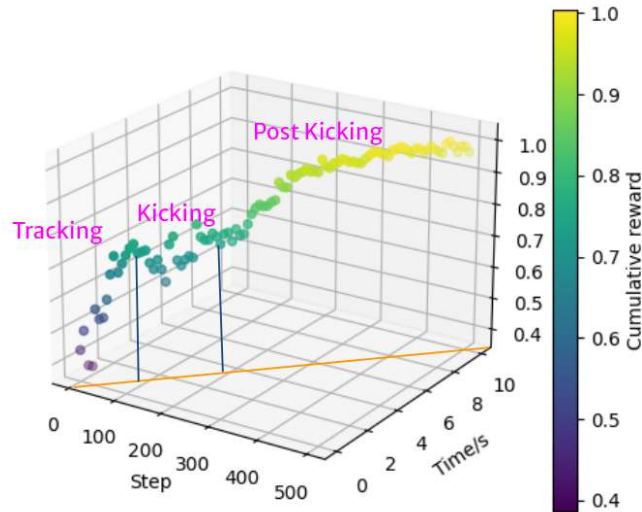
Methodology

There are some mechanics that we used to manipulate the connection between different stages.

1. Reward gradient
2. Status flag
3. Placeholder in observation

Reward gradient

In policy gradient methods, the goal is to maximize the expected cumulative reward of the whole trajectory. Reward gradient is used to update the parameters of the policy that can maximize the cumulative reward. For a single, monotonic task, reward gradient can be easily designed by articulating the reward function. For our task, multi-stages task, the reward gradient design is more important in order to smoothly transfer from one stage to another stage. For example, in the third stage, the cumulative reward should be higher than that in the first stage so that the robot can successfully transit from stage 1, pass stage2, and finally get stage3. In order to achieve that, we provide the increasing stage reward gain from stage1 to 3.



Status flag

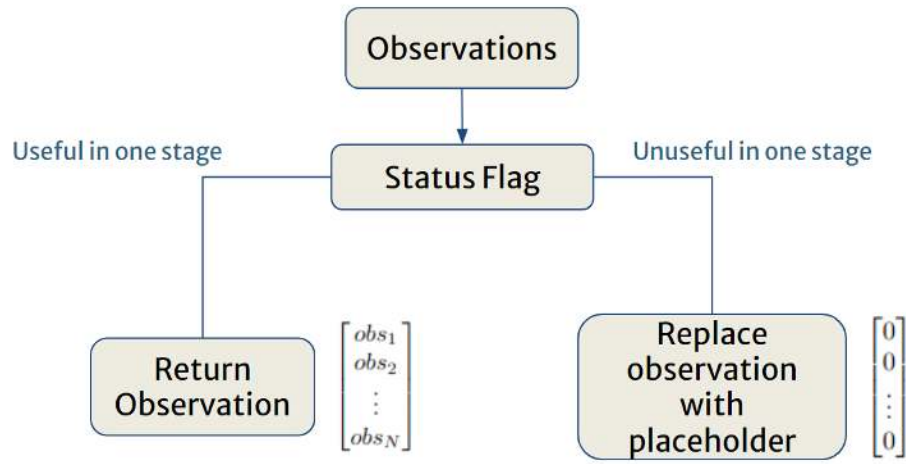
Status flag is the state machine that is used to separate different stages apart. There are two conditions that determine the status flag. One is the distance between the robot position and the initial ball position, when the distance is larger than the predefined threshold, the status flag is $[1, 0, 0]$ and the robot is in the first stage, it needs to go to the ball. Another condition is the distance between the initial position of the ball and the real position of the ball. This condition is used to judge whether the robot has kicked the ball. When the robot satisfies the first condition and not satisfied the second condition, it is in the second stage, it needs to kick the ball. When the second condition is satisfied, no matter whether the first condition is satisfied, which means the robot has kicked the ball, it is in the third stage, which indicates that it needs to wait and detect if the ball approaches the goal.

Conditions		Robot kicks the ball?	
		Yes	No
Robot near the ball?	Yes	Stage3	Stage2
	No	Stage3	Stage1

Placeholder

Placeholder in observation enables us to tackle different tasks with different observation terms. In this project, the stage1 doesn't need ball real time position and goal position for going to the ball so these two terms are useless, but in stage2 and stage3, they are pretty important. Thus, in order to keep the dimensionality of observation of different stages consistent, we view the ball's real time position and goal position as the placeholder in stage1's observation and mask

them with 0. In this method, the policy can smoothly transit between each stage without caring about the dimensional compatibility.



Results and Performance

To evaluate the shooting performance of the robot, several key metrics are considered. The metric includes:

1. **Final Distance from the Goal:** measures the distance between the final position of the ball and the goal. A smaller distance indicates more accurate shooting performance and vice versa.
2. **Direction of the Kick:** The direction in which the ball is kicked is analyzed to ensure it aligns with the desired target position. This ensures that the ball is propelled towards the goal with the intended trajectory.
3. **Time to Reach the Goal:** The time taken for the ball to reach the goal after being kicked is recorded. A shorter time indicates faster and potentially more effective shooting performance.
4. **Shooting Gait Stability:** The stability of the robot's gait after completing the kicking motion is essential. A stable gait indicates that the robot maintains balance and stability after executing the kick, hence affecting the overall performance.

By evaluating these metrics, the effectiveness and efficiency of the robot's shooting performance can be assessed comprehensively. This evaluation allows for refinement and optimization of the robot's behavior to improve its shooting accuracy and overall performance in achieving task objectives. The experimental kicking accuracy is approximately 60%.

Conclusion

In this project we design the multi-stage system with a single PPO algorithm to enable the quadruped robot to go to the soccer and kick the ball to the desired goal. Some methods, such as reward gradient, status flag and placeholder, are proposed to deal with the multi-stages task. The real experimental result shows that the whole task works well with this multi-stages policy.



Future Improvements

In future iterations, we envision integrating advanced camera detection technology to seamlessly track the ball and target, thereby enhancing precision in interactions. Our focus will extend to further refining kicking accuracy and quadruped walking motion, incorporating cutting-edge techniques to optimize performance. With the integration of Reinforcement Learning with Prior Data (RLPD), we anticipate fine-tuning real-world performance to unprecedented levels, leveraging existing data to continuously improve adaptability and effectiveness. This forward-looking approach underscores our commitment to pushing the boundaries of capability, ensuring our system remains at the forefront of innovation in dynamic environments.

References

- [1] Ji, Yandong, Zhongyu Li, Yinan Sun, Xue Bin Peng, Sergey Levine, Glen Berseth, and Koushil Sreenath. 'Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills Using a Quadrupedal Robot'. arXiv [Cs.RO], 2022. arXiv. <http://arxiv.org/abs/2208.01160>.
- [2] Huang, Xiaoyu, Zhongyu Li, Yanzhen Xiang, Yiming Ni, Yufeng Chi, Yunhao Li, Lizhi Yang, Xue Bin Peng, and Koushil Sreenath. 'Creating a Dynamic Quadrupedal Robotic Goalkeeper with Reinforcement Learning'. arXiv [Cs.RO], 2022. arXiv. <http://arxiv.org/abs/2210.04435>.
- [3] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 'Proximal Policy Optimization Algorithms'. arXiv [Cs.LG], 2017. arXiv. <http://arxiv.org/abs/1707.06347>.
- [4] Shao, Yecheng, Yongbin Jin, Xianwei Liu, Weiyan He, Hongtao Wang, and Wei Yang. 'Learning Free Gait Transition for Quadruped Robots Via Phase-Guided Controller'. *IEEE Robotics and Automation Letters* 7, no. 2 (April 2022): 1230–37. <https://doi.org/10.1109/lra.2021.3136645>.
- [5] Ball, Philip J., Laura Smith, Ilya Kostrikov, and Sergey Levine. 'Efficient Online Reinforcement Learning with Offline Data'. arXiv [Cs.LG], 2023. arXiv. <http://arxiv.org/abs/2302.02948>.
- [6] "What Are STEM Labs?" Education VEX. Accessed 20 Apr. 2024.
- [7] Ladd, Brittain. "Agility Robotics Is about to Revolutionize Fulfillment." Brittain Ladd, 20 Mar. 2023.
- [8] "Waymo Planning Expansion of Autonomous Taxis to L.A." NBCNews.com. NBCUniversal News Group, 23 Jan. 2024.