Dexterous Robotic Manipulation

CS 484 Final Project 2024

Blake Cantrell
Department of Computer Science
University of Alabama
Tuscaloosa, USA
btcantrell@crimson.ua.edu

I. Introduction

Modern technology has allowed the intersection of robotics and reinforcement learning to pave the way for advancements in autonomous manipulation tasks. Being able to teach robotic agents complex manipulation skills through trial and error of a simulation allows real-life scenarios without the need for a physical machine. In this project, we explore the use of reinforcement learning techniques to achieve dexterous robotic manipulation, with a focus on picking up an object. Robotic manipulation involves the control and coordination of robotic arms and hands to interact with objects in the environment. Usually, robotic manipulation relies on kinematic models and scripted motions in order to move around, leaving the ability to adapt to the environment out of the question. However, with the use of reinforcement learning, manipulation skills are learned autonomously from experience, giving adaptability to diverse scenarios. Algorithms such as Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) have become popular choices for training robotic agents in simulations [1]. These reinforcement learning (RL) algorithms use trial and error to learn it's policies through exploration and exploitation, where the agent receives feedback based on its actions. In this project, we used a simulated robotic environment powered by PyBullet, a physics engine, capable of accurately modeling interactions between robotic manipulators and objects. Our goal is to train a robotic agent to grab and pick up an object in the environment using an RL algorithm. Through experimentation and analysis, we want to gain insight into the effectiveness of different RL algorithms and reward functions for robotic manipulation tasks. By analyzing the success rates and performance of trained agents, key factors that contribute to successful grabbing and manipulation are hoping to be found. Our results show the impact of hyperparameter tuning on algorithm convergence and task performance, as well as the importance of balancing exploration and exploitation.

II. Method

A. Overall Approach

To create the simulation, we provided a virtual space using PyBullet, which gave everything needed in order to set up the environment. Within the environment we instantiate a Sawyer robotic arm model to act as our primary agent for manipulation. The robotic arm is a simplification of a real-life robot with waist, shoulder, elbow, wrist, and finger joints along with 5 prongs at the end to resemble a hand with fingers. Along with the robotic arm, a table model is rendered with an object to grab. In this case, a model of an aerosol can cap.

We integrated PyBullet with Stable Baselines3, a reinforcement learning library, that provides a collection of RL algorithms and utilities for training and evaluating agents within a simulated environment. This library allows quick use of RL algorithms such as PPO, Deep Q-Networks, and Actor-Critic methods for both research and practical uses [2]. Using this, we used a PPO algorithm to teach the arm how to find and pick up the object. Experimenting with different hyperparameters, I was able to find improved ways to run the algorithm, resulting in higher success rates. Along with the algorithm, the reward function is the last main part of the framework. The reward function implemented was developed after experimenting with different variations. The function I chose is a combination between rewarding based on correct axis and based on distance from the hand to the object.

B. Technical Description

A Proximal Policy Optimization (PPO) algorithm is a reinforcement learning algorithm that operates by iteratively optimizing a parameterized policy to maximize rewards obtained from interaction in the environment [3]. This implementation is a policy gradient method that employs a clipped objective function to ensure stable and efficient updates [2]. The key components of the PPO are the actor, critic, and clipped surrogate objective. The actor is the policy network that parameterizes the policy function, mapping observations from

the environment to actions [3]. In this implementation, a Multi-Layer Perceptron architecture is used to represent it. The critic is the value network that approximates the state-value function. This provides an estimate of the expected cumulative rewards from a given state, allowing use for in calculation for policy updates. This version of a PPO includes a clip surrogate objective, which has the job of constraining policy updates to a trust region [2]. This ensures that policy updates remain inside a threshold by clipping the probability ration between old and new policies [3]. Choosing a Proximal Policy Optimization (PPO) algorithm initially was a simple implementation that was expanded on during testing. Changing the hyperparameters through experimentation allowed the algorithm to improve its success rates, resulting in the current implementation. Increasing the train time led to better result, as well as changing the learning rate and the steps increased success rate as well.

C. *Reward Function Design*

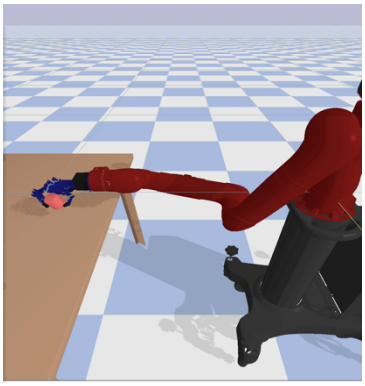
The reward function is utilized in our task to incentivize the agent to move toward the object and grasp it. To begin with, the reward function was a simple implementation that rewarded being in all three correct axes and punished not being in. This led to low success results, leading to it being changed and tested in different ways. The method I used was a result of a combination between two methods I attempted. The first method rewards based on position in regard to the target of the axis. Each axis has a point the hand needs to move to, and if that axis is in the correct area, it is rewarded based on how many axes are in the correct location. The second method that was combined along with this one is based on distance between the hand and the object. An algorithm is used to calculate the reward by taking the distance and creating an inverse proportional to it:

$$R = \max(1 - dist / dist\_max, min\_reward)$$

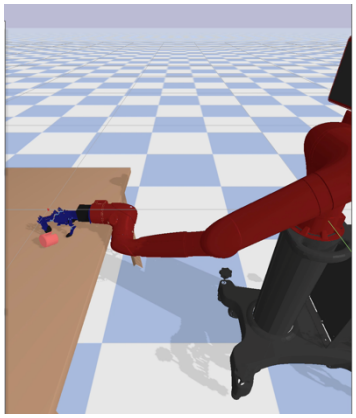
Combining these two methods produced better results than either did individually.

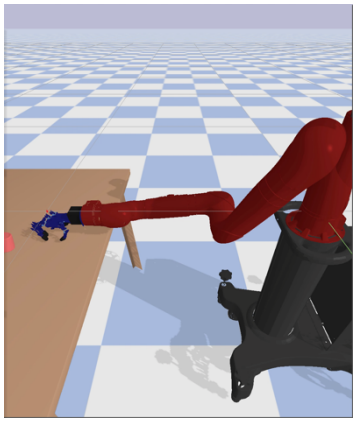
III. Results

Measuring the manipulation of an object is done through a success rate for each trained model. These rates are calculated based on the percentage of trials in which the object was grabbed and lifted to the desired location. Success rates found in our data range from 20% to 70% depending on the given method. Altering of the hyperparameters led to these different results and allowed testing to determine the ideal parameters. Testing with different time steps, smaller time steps produced low success for results with a high fluctuation. Increasing this benefited the algorithm greatly and allowed more time for the model to learn. Experimenting with the learning rate, the initial learning rate of 3e-4 turned out to be the best from testing. Decreasing the rate resulted in a high stability, but a low convergence. Increasing led to faster learning but would result in a higher range of rates. Lastly, the modifying of the training steps changes the number of steps in which the RL agent collects before it performs a policy update. When increased from 1024 to 2048, the success rates had an increase as well. However, steps higher than this dropped the success of the model back down and increased the computational complexity of each iteration a little too high. As a result, our method was able to consistently achieve results of success in the 40-50% range with an occasional result in the 30% or upper 50% range. Other results produced wide ranges of rates in the 20%-60% or maintained lower rates in the 20-40% range.



Models that performed a successful grasp picked up the object without dropping it and moved it up to the desired location.





Failed attempts usually resulted from the finger of the arm sliding the object away from its resting point or getting stuck in the hole of the cap.

IV. Discussion of Results

Using the PPO algorithm, we were able to see some interesting results. By choosing the PPO algorithm, our model was able to produce stable results that never ended in a complete failure. We were able to scale it to fit our environments needs and edit parameters to better our results. However, the hyperparameters are sensitive and at times can cause drastic issues if not tuned carefully. Also due to the exploration strategy, our PPO can at times be limited in exploration of the action space. Overall, our algorithm was not a success in getting high rates of completion and resulted in an average of under half of the models succeeding. On the other hand, our model was successful in showing the effects of changing hyperparameters and getting stable results, showing that further testing and tuning may lead to improvement of the system. The reward function used in our project produces promising results with room to improve. After testing many methods, using the three parameters for the axes along with the distance produces a reward function that combines position and distance-based rewards. Choosing the position-based reward made sense that if an object is in the space, it is needed then it should be incentivized to remain there. Using distance, on the other hand, we have a way to give somewhat of a direction that the arm should move in. By using the decrease of distance to increase rewards, we are able to close in faster on the resting object.

V. Conclusion

In conclusion, our project aimed to train a robotic agent to manipulate and object using the application of reinforcement learning algorithms. Using our PyBullet environment to simulate and train, we were able to come up with many results regarding our algorithm. Implementing a PPO algorithm with high train time, we came up with many different methods trying to achieve the most accurate result. Creating a reward system that takes into account both position and distance to grab our object, we were able to teach the simulation how to move to the object and pick it up on average 40-50% of the time. In summary, the transformative impact of RL in robotics shows that the future of autonomous manipulation and human-robot collaboration is bright.

References

[1] "Introduction to Reinforcement Learning – A Robotics Perspective» Lamarr Institute," Jul. 05, 2023. https://lamarr-institute.org/blog/reinforcement-learning-and-robotics/

[2] "Examples – Stable Baselines3 2.4.0a0 documentation," *stable-baselines3.readthedocs.io.* https://stablebaselines3.readthedocs.io/en/master/guide/examples.html

[3] R. S. Sutton and A. Barto, *Reinforcement learning : an introduction*. Cambridge, Ma ; Lodon: The Mit Press, 2018.