Unlocking Robotic Manipulator Potential:
Exploring Model-Free Reinforcement Learning for Adaptive Task Performance

Aaditya Prabhu

Abstract:
Robotic manipulators hold significant promise across various industrial applications, yet their adaptability to dynamic and evolving environments remains a challenge. Leveraging artificial intelligence, particularly reinforcement learning (RL), presents a compelling avenue to enhance the capabilities of these manipulators. In this study, we investigate the efficacy of model-free RL algorithms in training robotic manipulators for pushing tasks, crucial for applications like high-mix-low-volume manufacturing and household assistance. We explore five prominent model-free RL algorithms: Proximal Primal Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3), Advantage actor critic (A2C), and Soft Actor Critic (SAC). These algorithms offer distinct approaches to training agents by iteratively interacting with their environments to maximize cumulative rewards. We study their mechanisms, focusing on policy optimization and value function estimation. Furthermore, we provide a detailed setup elucidating the components of RL, including agents, environments, observations, actions, and reward functions. We discuss the nuances between fully observed and partially observed environments, as well as discrete and continuous action spaces, crucial for training robotic manipulators. Utilizing Python libraries such as stable-baselines3 and panda-gym, we conduct experiments with a Franka Panda robot simulated in the pybullet physics environment.Ultimately, this study contributes to advancing the field of robotic manipulation through the integration of cutting-edge RL techniques.

Table of Contents

Introduction

Robotic manipulators have a variety of industrial applications in today's world, however, these manipulators are still limited in the tasks they perform.Yet, Robotic manipulators have a lot of potential to help the society by performing a lot of tasks such as high-mix-low-volume manufacturing and people's homes. Even though robots have been used in factories for a long time, it is still challenging to use in environments that require frequent changes such as the one mentioned above. To fix this, attempts have been made to introduce artificial intelligence (AI) to these robots as they help robots adapt to their surroundings. Reinforcement learning (RL), a part of AI, is a type of machine learning whereby an agent interacts with its environment in an attempt to maximize its cumulative reward over a finite or infinite time horizon[1].

RL can be applied to robots whereby the robots are designated a higher award depending on their success on completing various tasks. For example, a robot that is tasked with stacking blocks could be assigned a positive reward for each block that is stacked, and a reward of 0 for each block that falls.The advantages of RL are that it reinforces desirable behavior by encouraging the agent to both explore new actions and exploit actions that are known to increase reward.. Continual RL is a type of RL whereby the parameters of the learned policy are updated during the online functioning of the robot, thus helping the robot adapt to changes for an extended period of time. Traditionally, robots use model-based methods for control. For example, model-predictive control is a popular control strategy that optimizes the control actions of a robot over its entire trajectory, yet replans at every timestep. However, many tasks and systems cannot be modeled using physics. For example, tasks such as block pushing, although it seems simple,requires frequent making and breaking of contact. This introduces discontinuities in the system model, thus making model-based learning challenging. Contrary to model-based learning, model-free learning tends to be easier to implement and tune even though it lacks the advantages of sample efficiency. The model free RL that will be observed in the paper will be Proximal Primal Optimization(PPO),Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG(TD3), Advantage actor critic (A2C), Soft Actor Critic (SAC). This research paper will look at various types of model free RL algorithms to train a robotic manipulator to perform pushing tasks.

Setup

RL algorithm

RL has 2 main components: agent and environment. The environment is the world in which the perpetrator lives and interacts. Every interaction with the agent monitors the state of the environment and then decides to take action. The representative also came up with the gift, a number that shows interest in the current state of the world. The main aim of an agency is to maximize profits.

State "s" is a complete description of the state of the world. The “O” observation is a partial description of the situation that may not be included in the data. These two forces are represented by real vectors, matrices, or higher tensors. If the agent can observe the entire state of the environment, the environment is fully visible. However, when the agent can only see part of the observation, the environment is partially seen.

The process of working effectively in an environment is called the action space. A discrete decision environment is one in which there are only a few actions available to the agent, while a continuous action environment consists of actions with real value that can return value in real numbers. Therefore,they are used to control the workers of robots in the physical world. For example, the function of a robot arm is the force of each joint, which is a continuous value.

Policy (i.e., the agent's brain) are the rules the agent uses to decide what to do. It can be arbitrary or random. Parameterized strategies are often used to support learning. Parametric techniques are techniques whose results are calculated based on a function that can be modified by some optimization method to change the behavior.

The reward function R depends on the current state of the world, the action, and the next state. Non-discounted returns are limited to the benefits received. Short-term rewards are not necessarily the sum of all rewards an agent receives, but a discount based on future rewards. This function includes a discount factor.

The reinforcement learning problem is to find the policy that directs evaluation to the action (or actions) that leads to the best results. The cost function estimates the expected future rewards the agent can receive from a state or state action. There are 4 function values, one of which is the rulevalue that gives the expected return when you start from state "s" and follow the policy "π".

If you start in state "s", perform action "a", and then "act" according to the rule ad infinitum, the ordinal value according to the rule gives the expected payoff: you start in state s and always comply with the agreement. Working in the environment will give the best value, expected return. The best way to do this is to start from the "s' ' state, make everything "a", and then follow the same rules around forever.

The advantage function similar to the policy "π" explains how it is better to make a particular decision in the state than to choose an action based on the assumption that your decision penalty will last forever.

PPO algorithm

Proximal Primal Optimization(PPO) is an RL algorithm based on the Minor Maximization (MM) algorithm. As PPO is classified as a policy gradient method, it works by computing an estimator of the policy gradient and then inserting it into a stochastic gradient ascent algorithm which is usually in the form.

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

In this scenario, where ‘pi’ denotes a probabilistic policy and 'A^t' stands for an estimate of the advantage function at a specific time 't,' the term 'E^t' represents the average calculated from a limited set of samples.This includes algorithms that switch between models and optimization levels. Applications using different software work automatically by creating an objective function whose gradient functions follow the gradient approximation rule. The "g^" estimator is obtained by taking the derivative of the objective function.

The objective function will allow maximizing the dominance function necessary to extract the basis of variation. However, this prohibition does not apply in order to ensure that the new law does not differ from the old law.

PPO with truncated target is the best way to use the first line to solve the problem without penalizing the target when we change the rules too much.

Link:
https://colab.research.google.com/drive/1KaZuScT3akzTtHio2LLlekcM89SxhSo2?usp=sharing

DDPG algorithm

The Deep Deterministic Policy Gradient (DDPG) algorithm is an algorithm that learns both the Q function and the policy simultaneously. If you know the best performance values, you can make a good deal by building them into your strategy.
We can develop the standard deviation of Bellman error (MBSE) which tells us that the approximator satisfies the Bellman equation.

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

Here, when calculating (1-d), we use Python's rule of evaluating True as 1 and False as 0. Therefore, when d is True, that is, when s a, the end state - Q function must be shown. that the agency will no longer receive gifts after the current situation. Once DDPG calculates the maximum rank on the target, DDPG processes the target's network policy to calculate the focus rank. DDPG's operating policy is simple. Since the order is constant, we assume that the Q function is a variable of the order and we can solve this with the gradient rise function.

Link:
https://colab.research.google.com/drive/1PtkkBJh6jzmsMjvytxmDtR9J-hgFXDM_?usp=sharing

TD3 algorithm

Although the DDPG algorithm works well, the Q learning function can cause damage by overestimating the Q value. Twin Delay DDPG (TD3) is an offline algorithm that solves this problem by introducing 3 main ideas.

The first tip is to trim the two Q-learnings where TD3 learns 2 Q functions instead of 1 and then uses the smaller of the 2 Q values to create a target in the Bellman error loss. The second method is "delayed" policy update, where TD3 updates the policy less frequently than the Q function. The third strategy is objective function smoothing, where TD3 adds noise to the objective function and makes it difficult for the authority to misuse the Q function by changing Q and work.

TD3 achieves target law smoothing by using the action to create a Q-function learning target based on the target law, but with clipping noise at each point in the action. After all noise cuts are collected, the target motion is cut in the middle of the validity range. TD3 uses truncated two Q runs using a target in two Q functions to calculate which of the 2 functions gives the smaller value to the target. Then return to that goal and learn both. Finally, the rule is learned by making predictions.

Link: https://colab.research.google.com/drive/1HbaVJoVPlgVryJLklkveL2XnMKTGwfpO

A2C algorithm

The Advantage Actor Critic (A2C) algorithm involves estimating the value of “Critic” and sets the gradient rule in the direction specified by the Critic(“Action”). Both functions are parameterized by neural networks. Subtracting the Q value from the V value gives the dominance value. We can find the loss of A2C using the relationship between Q and V in the Bellman equation.

$$\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)$$

Link:
https://colab.research.google.com/drive/1jvnWKyiO-Lwv-7WQf0hbHngWu4U4TRA_?usp=sharing

SAC algorithm

Soft Actor Critic (SAC) is an algorithm used to optimize stochastic policies in a non-policy way, creating a link between stochastic policy optimization and DDPG-style methods. Like TD3, SAC uses a truncated double Q number and a smoothing target. Additionally, a feature of SAC is entropy normalization. The rule is trained to maintain a balance between the desired reward and entropy, which is the average amount of "information"contained in the variable; This will help the algorithm speed up the learning process and prevent the policy learning process. Preliminary discussion to a bad place. Best price. allows us to define a value function that will include the reward entropy each time.

The rule has been modified to include an entropy reward for all time steps except the first. SAC learns one rule and 2 Q functions at the same time.SAC uses this approximate target to set the MSBE parameters for each Q function. SAC uses two trimmed Q numbers and uses the minimum Q of 2 approximators.
Where the goal is to fix. In all cases, the SAC policy must maximize expected future return plus expected future entropy. So it must be the most effective. Then there is the optimization of the law.

Link:
https://colab.research.google.com/drive/1IEKAARwvE0HktedbOKWGSrAht2mD7W58?usp=sharing

FRANKA PANDA ROBOT

The Franks Panda robot push task creates a robotic arm with 7 degrees of freedom, it also has a parallel jaw end effector. The robot is moduled using Python's pybullet physics simulation library.

Python libraries

The libraries used in this research paper are stable-baselines3 and panda-gym is a set of reliable implementations of RL learning algorithms. It has a unified structure for algorithms. Panda-gym is based upon a gymnasium which is a library for RL algorithms. This free open source package allows us to define robotic tasks.

RESULTS

Hyperparameters used:

MultiInputPolicy is a type of Policy that allows multiple inputs. env is a function which creates a copy of the environment. n_steps is the number of steps to run for each environment per update. gae_lambda is a weighted estimator of an advantage function. gamma is the discount factor. n_epochs is the number of epochs of the interaction. ent_coef if the current value of the entropy coefficient. learning_rate is used to govern the pace at which an algorithm updates. clip_range gives the range for clipping the policy. Sde_sample_freq samples a new noise matrix every n steps.

The following Hyperparameter were used for all of these algorithms. The hyperparameter for PPO & A2C algorithm were:

```
("MultiInputPolicy",
env,
verbose = 0,
n_steps = 1024,
gae_lambda = 0.95,
gamma = 0.9,
n_epochs = 10,
ent_coef = 0.0,
learning_rate = 1e-3,
clip_range = 0.2,
use_side = True,
sde_sample_freq = 4)
```

The hyperparameter for DDPG & TD3 algorithm were:

```
("MultiInputPolicy",
env,
verbose = 0,
learning_rate = 1e-3)
```

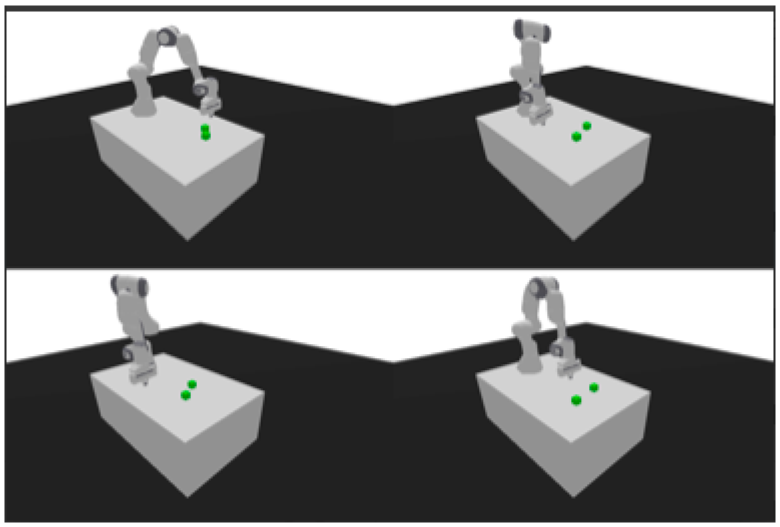
The hyperparameter for SAC algorithm were:

```
("MultiInputPolicy",
env,
verbose = 0,
gamma = 0.9,
ent_coef = 0.0,
learning_rate = 1e-3,
use_side = True,
sde_sample_freq = 4)
```

Description of how the code works

The first step of the code is to import libraries panda-gym and stable baselines. We create an environment by using the make_vec_env. We then create our model and choose our hyperparameter. We make the model learn for 10000 time steps. Then we save the trained agent, create a policy and then save that policy. After that we load the trained agent and the policy. Then we reset the environment and get the initial observation. Finally we demonstrate the trained policy using a for loop to stimulate the policy by predicting the states and action, and then saving it as a GIF. To attain the graphs I used the tensorboard function which created the function.

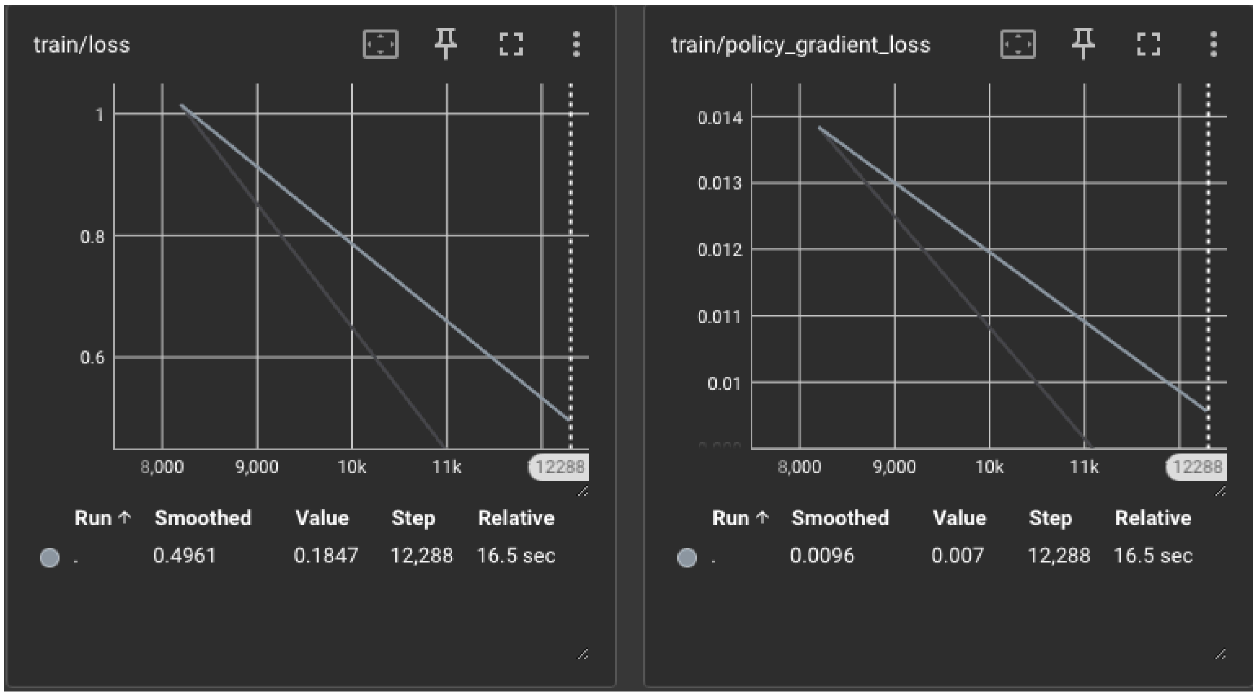
Robot Arm



Graphs

The graphs used to show the effectiveness of each algorithm were made by using the tensorboard algorithm. Below are the graphs for the general criteria of *train/loss.*

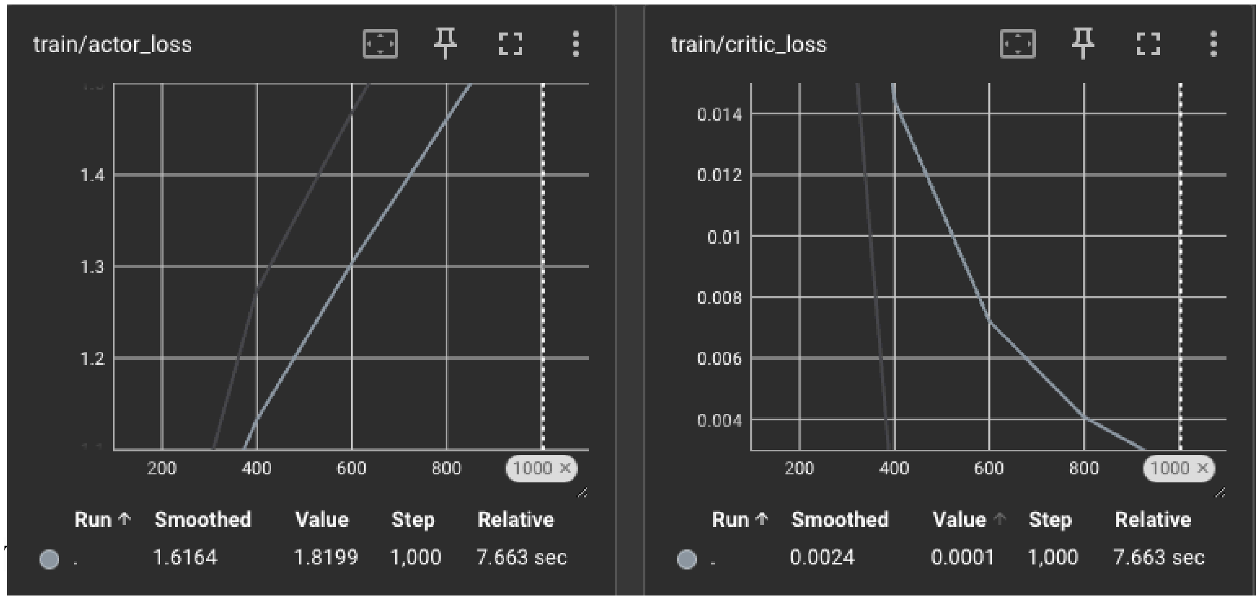
PPO algorithm:



Run ↑	Smoothed	Value	Step	Relative
● .	0.4961	0.1847	12,288	16.5 sec

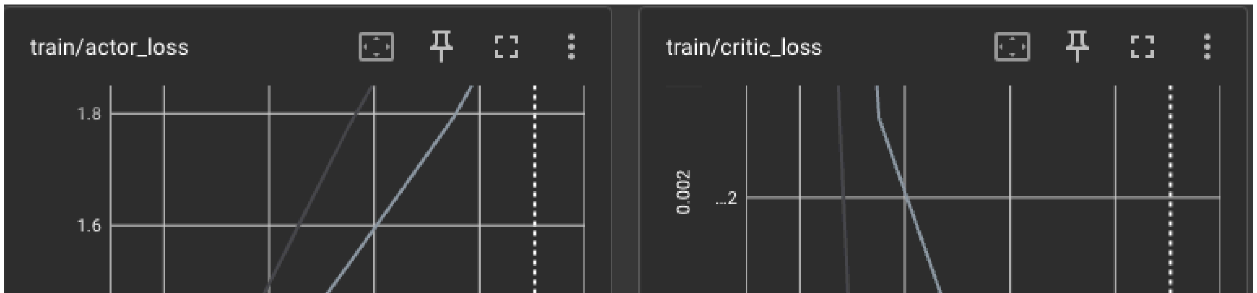
Run ↑	Smoothed	Value	Step	Relative
● .	0.0096	0.007	12,288	16.5 sec

DDPG Algorithm:

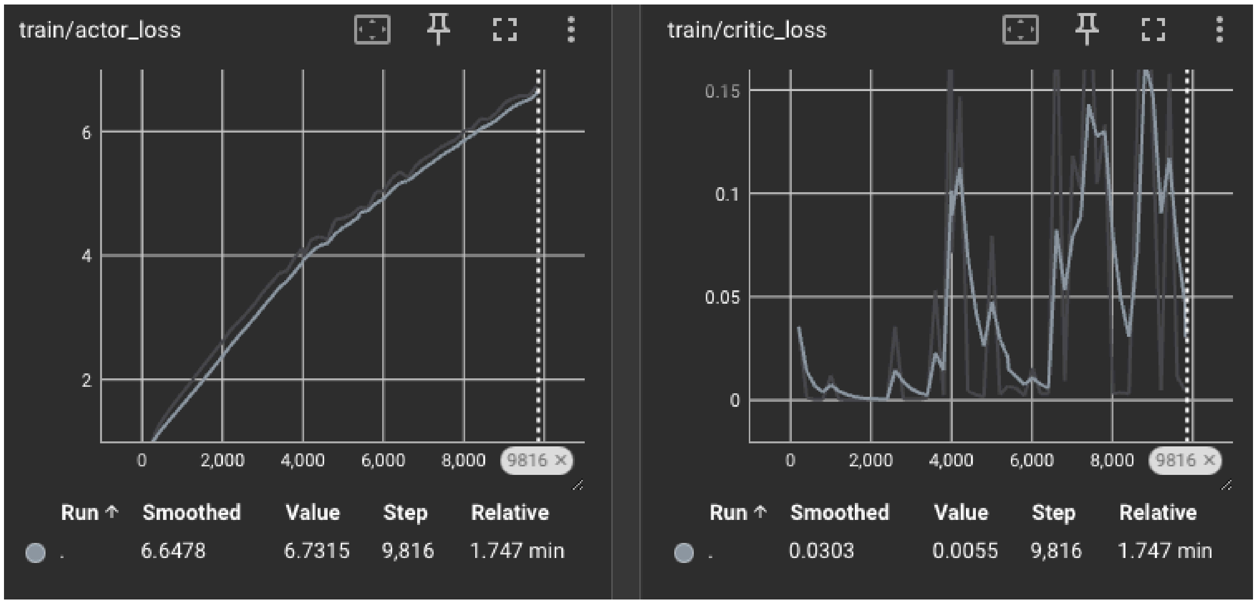


Run ↑	Smoothed	Value	Step	Relative
● .	1.6164	1.8199	1,000	7.663 sec

Run ↑	Smoothed	Value ↑	Step	Relative
● .	0.0024	0.0001	1,000	7.663 sec



SAC Algorithm:



Run ↑	Smoothed	Value	Step	Relative
● .	6.6478	6.7315	9,816	1.747 min

Run ↑	Smoothed	Value	Step	Relative
● .	0.0303	0.0055	9,816	1.747 min

Discussion

The results above do not converge with the optimum policy as that would require higher levels of computation.Furthermore the graph for A2C was not attainable as the tensorboard program was not applicable with A2C algorithm. .In the future the other tasks for the Franks Panda Robot Arm that we could train are Reach, where the robot must be placed its end-effector at a target position. Slide, where the robot has to slide an object to a target position, Pick and Place, where the has to pick up and place an object at a target position, Stack, where the robot has to stack 2 cubes at a target position, and Flip, where the robot must flip the cube to a target orientation.

Conclusion

In conclusion, this study explored the application of model-free reinforcement learning (RL) algorithms for training robotic manipulators to perform pushing tasks. Five prominent algorithms—PPO, DDPG, TD3, A2C, SAC—were investigated, shedding light on their strengths and limitations.

Through detailed experimentation and analysis, this research provided insights into the efficacy of these RL algorithms in enhancing the adaptability of robotic manipulators to dynamic environments. The comparative analysis of the algorithms, along with the discussion on their mechanisms and setups, contributes to advancing the field of robotic manipulation.

While the results indicate progress in training robotic manipulators for pushing tasks, further research is warranted to explore additional tasks such as reach, slide, pick and place, stack, and flip. Moreover, future endeavors could focus on optimizing hyperparameters and scaling algorithms to tackle more complex tasks. Overall, this study underscores the potential of RL in unlocking the full capabilities of robotic manipulators across various industrial and household applications.