RL Project - Applied case studies of Machine Learning

Rava Miro, Lucchina Luca, Nazifi Endrit, Sommaruga Tommaso
SUPSI

Abstract
This report explores the application of reinforcement learning (RL) techniques to robotic control tasks, specifically training a Franka Emika Panda robotic arm to autonomously grasp and lift a cube. The study leverages the Proximal Policy Optimization (PPO) algorithm in a MuJoCo-powered simulation environment to achieve robust task execution.

Results demonstrate that iterative refinement of the reward function was critical in addressing unintended behaviors, such as balancing the cube on its vertices instead of lifting it. Additional metrics, such as cube height and grasp counts, enhanced observability, enabling more efficient debugging.

Contents

1 Problem Definition **2**
1.1 Task Overview 2
1.2 Objectives of the Project 2

2 Provided Setup **2**
2.1 Codebase and Structure 2
2.2 Simulation Environment 3
2.3 Robot Description 3

3 Reward Function Design **3**
3.1 Conceptual Approach 4
3.2 State Management 4
3.3 Intermediate Rewards 4
3.4 Additional Components (e.g., Rotation Penalty) 4
3.5 Merging 5

4 Experiments and Methodology **5**
4.1 Reward Function Iterations . . 5
4.2 Hyperparameter Tuning 6

5 Results and Observations **7**
5.1 Training Performance and Behavior 7
5.2 Challenges: The "Balancing Maneuver" 7
5.3 Enhanced Observation and Data Logging 8
5.4 Insights From Iterative Refinement 8

6 Challenges Encountered **9**
6.1 Attempts to Install `mujoco-py` . 9
6.1.1 Native Installation on Ubuntu 9
6.1.2 Containerized Installation Using Docker 9
6.1.3 Virtual Machine Setup on Windows 10
6.1.4 Switching to the Modern MuJoCo Library . . 10
6.2 Broader Implications and Observations 10
6.3 Recommendations for Future Iterations 10
6.4 Personal Reflection Of Our Group 11

7 Insights and Future Work **11**
7.1 Key Insights 11
7.2 Recommendations for Future Efforts 11
7.3 Final Thoughts 12

8 Conclusions and Acknowledgments **12**

1 Problem Definition

1.1 Task Overview

The primary task in this project involves training a Franka Emika Panda robotic arm to autonomously grasp and lift a cube placed on a flat surface. This requires the development of a learning system that can enable the robot to:

1. **Approach the Cube:** Navigate its gripper to a position suitable for grasping.
2. **Securely Grasp the Cube:** Control the gripper's motion to successfully pick up the object without dropping it.
3. **Lift the Cube:** Raise the cube off the surface to a designated height.

The robot's ability to perform these actions relies on reinforcement learning techniques. Specifically, the project uses the Proximal Policy Optimization (PPO) algorithm, a widely used RL method for continuous control tasks. The learning process occurs in a simulated environment powered by MuJoCo, where physics interactions such as friction and collision are realistically modeled.

1.2 Objectives of the Project

The final goal of this project is to enable robust task execution through reinforcement learning. This is achieved through the following objectives:

1. **Designing an Effective Reward Function:** A critical part of the RL system is the reward function, which guides the agent by providing feedback at different stages of the task. Instead of sparse rewards provided only upon successful task completion, this project explores *reward shaping*, where intermediate rewards are assigned to encourage step-by-step progress.

   For example:

   - A small reward for reducing the distance between the gripper and the cube.
   - A reward for successful grasping and lifting.
   - Penalties for dropping the cube or twisting the arm.

2. **Optimizing Training Parameters:** The learning process is sensitive to hyperparameter choices such as learning rate, discount factor, and batch size. The project involves also experimenting with these parameters to understand their impact on the robot's performance and stability during training.

---

2 Provided Setup

Codebase structure, simulation environment, and robot features essential for reinforcement learning in robotic grasping tasks.

2.1 Codebase and Structure

The project was built upon a pre-existing codebase designed to facilitate reinforcement learning for robotic grasping tasks. The key components of the codebase include:

- `config/`: Contains configuration files that allow customization of hyperparameters such as learning rate, discount factor, and batch size.
- `controller/`: Implements controllers that translate high-level policy actions

into joint torques for the robotic arm.

- `environments/`: Provides the main logic for task simulation and interaction, including physics modeling and reward function definition.
- `models/`: Manages simulation objects and robot modeling.
- `rl/`: Includes implementations of reinforcement learning algorithms such as Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).
- `utils/`: Offers utility functions for data processing, logging, and additional tasks.

This modular structure facilitated efficient experimentation and modifications during the project.

2.2 Simulation Environment

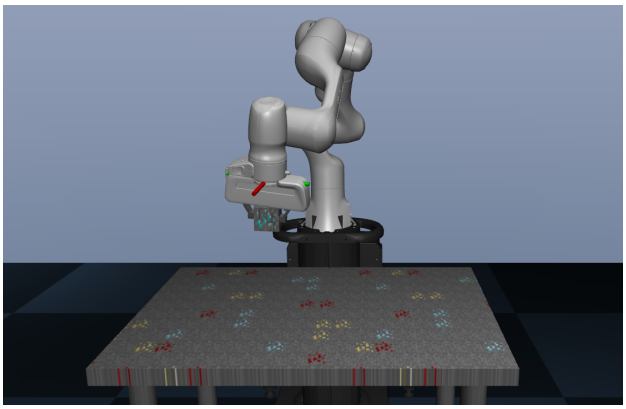
The simulation environment was powered by the MuJoCo physics engine, known for its high-fidelity modeling of rigid-body dynamics and continuous control scenarios. The environment features a Franka Emika Panda robotic arm placed near a flat table, with a cube serving as the primary object of interaction.

Key elements of the simulation include:

- Realistic physical properties such as friction, gravity, and collision detection.
- Adjustable initial conditions, including cube position and orientation.
- Real-time visualization for observing the robot's interactions and behaviors.

2.3 Robot Description

The Franka Emika Panda robot is a seven-degree-of-freedom (7-DOF) robotic arm equipped with a two-fingered parallel gripper. Its advanced design makes it ideal for precision tasks such as grasping and manipulation.



[fig 2.3] The provided setup (the robot arm is lifting the cube here).

Features of the robot include:

- High dexterity due to its multiple joints, enabling complex movements.
- A gripper with adjustable force and position control, crucial for securely grasping objects.
- Compatibility with the MuJoCo physics engine, allowing seamless integration into the simulation environment.

3 Reward Function Design

This section describes the approach taken in reward function design

The reward function is a critical component of the reinforcement learning setup, guiding the agent toward the desired behavior. In this task, the goal is to design a reward system that incentivizes the robot to manipulate and interact with an object effectively. The following subsections outline the conceptual approach, state management, and various components that contribute to the reward design

3.1 Conceptual Approach

The primary objective of the reward function is to encourage the robot to reduce the distance to the target object (a cube), grasp it, and manipulate it within a defined workspace. The reward is a dynamic function of the gripper's position relative to the cube and considers various factors like distance, alignment, and action intensity. The reward is adjusted based on the following principles:

- **Distance-based reward:** When the robot moves closer to the target, it receives a positive reward, while larger distances are penalized.
- **Action-based penalty:** If the gripper applies an unnecessary force or incorrect action (e.g., gripping when not needed), it receives a penalty.
- **Grasping and manipulation incentives:** If the gripper successfully grasps the cube and manipulates it, the reward increases significantly.

3.2 State Management

State management involves tracking and updating the key variables that influence the reward. The state is primarily defined by the relative positions of the cube and the robot's gripper, as well as the action being taken by the robot. These state variables are used to compute the reward in the following way:

- **Cube Position:** The position of the target cube is tracked using: `self.sim.data.body_xpos[self.cube_body_id]`.
- **Gripper Position:** The position of the robot's gripper is monitored using: `self.sim.data.site_xpos[self.eef_site_id]`.
- **Distance Calculation:** The Euclidean distance between the gripper and the cube is computed as:

$$\text{dist\_to\_cube} = \sqrt{(x_{\text{gripper}} - x_{\text{cube}})^2 + (y_{\text{gripper}} - y_{\text{cube}})^2 + (z_{\text{gripper}} - z_{\text{cube}})^2}$$

3.3 Intermediate Rewards

Intermediate rewards incentivize the robot for making progress towards the goal. These rewards are conditional based on the distance to the cube:

- **Large Distance (dist > 0.3):** A significant penalty is applied to discourage the robot from moving too far away.
- **Moderate Distance (0.03 < dist < 0.3):** A reward is given based on the inverse of the distance, encouraging the robot to approach the cube.
- **Small Distance (dist < 0.03):** At this stage, if the robot is in close proximity to the cube, the reward is further adjusted based on the grasping quality and action behavior.

3.4 Additional Components (e.g., Rotation Penalty)

Additional rewards are given based on specific actions or conditions:

- **Action-based Penalty:** If the action involves closing the gripper too soon, a penalty is applied, such as:
$$\text{penalty} = -1 \times \text{action[-1]}$$
- **Grasp Check:** If the robot successfully grasps the cube, further rewards are added. The grasp quality is checked with `self.check_grasp(self.sim.data.contact)`. If the gripper maintains a good grasp (either current or previous), the reward increases based on the quality of the grasp and the rotation of the cube.
- **Manipulation Reward:** A significant reward is given if the robot not only grasps the cube but manipulates it correctly, e.g., by lifting it without rotating it.

3.5 Merging

The reward function is designed to:

- Encourage the robot to approach the cube.
- Penalize unnecessary actions that take the robot farther from the goal.
- Reward proper grasping and manipulation, ensuring the robot is incentivized to complete the task effectively.

By Merging all these considerations, the reward function obtained became a robust mechanism for guiding the robot's learning process.

---

4 Experiments and Methodology

This section provides an overview of the experimental setup, methodology, and the hyperparameter tuning process.

At this stage, we conducted a series of experiments to evaluate the effectiveness of various reward functions and to fine-tune the learning process of the robot arm. Over 20 different reward functions were tested iteratively, with each batch of experiments targeting specific aspects of the robot's behavior.

4.1 Reward Function Iterations

The experiments were divided into different phases, each focusing on distinct aspects of the robot's task. Initially, we focused on testing reward functions that would encourage the robot arm to approach the target cube. The first five reward functions were designed to address this objective. These reward functions primarily provided positive feedback based on the robot's proximity to the cube, incentivizing it to move closer and align with the target object.

In the next set of experiments, another five reward functions were tested to encourage the robot arm to open the gripper and prepare for grasping the cube. These reward functions were specifically crafted to penalize the robot if it kept its gripper closed, and to reward it when the gripper was opened. The idea was to allow the robot to develop a clear understanding of when to release the gripper and when to grasp the cube.

The final set of experiments combined both the proximity-based rewards and the gripper behavior rewards, while introducing addi-

tional incentives for lifting the cube and penalties for rotating it. The latter modification was crucial because we observed that, without the penalty for rotation, the robot arm tended to rotate the cube on one of its vertices, lifting the center but not successfully raising the cube off the table. Thus, the penalty for rotation was implemented to discourage such undesirable behavior and encourage the robot to lift the entire cube off the surface.

The methodology for selecting the best reward functions involved trial and error. We closely monitored several key metrics during each experiment: the reward plot, the grasp counter (which we implemented to track the number of successful grasps), the cube's height in meters, the episode success plot, and the various loss plots. This data provided insights into how the robot was learning and where adjustments could be made in the reward functions.

4.2 Hyperparameter Tuning

In addition to experimenting with different reward functions, we also focused on fine-tuning the hyperparameters of the reinforcement learning model. Hyperparameter tuning was an essential part of the methodology, as it significantly influenced the stability and performance of the learning process. Each hyperparameter was iteratively adjusted and tested, requiring over 40 runs of the model, each of which could take up to 8 hours to complete. The following hyperparameters were the primary focus:

- **clip_param:** This parameter controls the clipping range for PPO's policy updates, ensuring that updates do not deviate too much and preventing instability in the model. A smaller value for `clip_param` resulted in more conservative updates, slowing down the learning process but improving stability. A larger value allowed the model to explore more aggressively but led to higher instability.
- **value_loss_coeff:** This coefficient determines the weight of the value function loss in PPO. It balances the importance of value estimation versus policy optimization. Higher values for this parameter placed more importance on improving value estimates, potentially at the cost of slower policy improvement. Lower values allowed faster policy updates but could lead to less accurate value estimates.
- **action_loss_coeff:** This parameter controls the weight for the action (policy) loss. It determines how much the policy should be penalized for suboptimal actions. A higher weight increased the focus on optimizing the policy, leading to faster but less stable improvements in the agent's behavior.
- **entropy_loss_coeff:** This coefficient encourages exploration by penalizing certainty in action selection. Higher values for `entropy_loss_coeff` promote more exploration of different actions. However, excessive exploration can reduce the model's ability to exploit learned behaviors effectively. Instead, lower values encourage the agent to exploit known actions, leading to faster convergence at the risk of prematurely converging to suboptimal solutions.
- **gae_lambda:** This parameter controls the bias-variance tradeoff in Generalized Advantage Estimation (GAE). A higher `gae_lambda` reduces bias but increases variance, leading to more volatile learning.

We also experimented with adjusting the discount factor and learning rate. Reducing the discount factor resulted in the robot focusing more on immediate rewards, leading to faster reactions but less long-term planning. Conversely, a higher discount factor encouraged the robot to consider future rewards more heavily, promoting long-term strategies but

potentially slowing down the learning process. Similarly, the learning rate determined the size of each policy update. A smaller learning rate led to more gradual changes, which helped stabilize learning but increased the total number of iterations required for convergence. A larger learning rate allowed faster learning but often led to instability and reduced repeatability of actions.

Through this extensive hyperparameter tuning process, we found that the default parameters provided a balanced tradeoff between exploration and exploitation. While adjusting these parameters did lead to some improvements in specific areas, it also introduced higher instability and made the model's behavior less repeatable. We observed that reducing the discount factor and learning rate led to more stable results but at the cost of slower convergence. Specifically, the robot was able to reach episode success after 500,000 iterations with the default settings, but reducing the learning rate and discount factor delayed this success until around 8 million iterations.

Overall, hyperparameter tuning was a time-consuming process that required numerous iterations to find a good balance between stability and learning speed, each iteration took sometimes many hour to conclude. The insights gained from these experiments helped refine the model and improve its overall performance in grasping and manipulating the cube.

5 Results and Observations

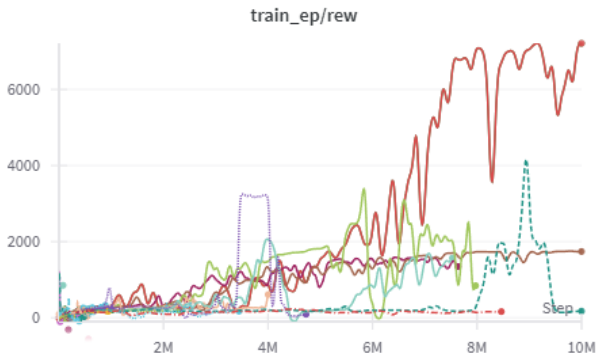
A complete overview on the results of the project

In this section, we summarize the results obtained from our experiments, highlight the key observations made during the training process, and discuss how modifications to the reward function and methodology influenced the robot's performance.

5.1 Training Performance and Behavior

The results showed that certain reward functions allowed the robot to successfully grasp the cube within approximately 300,000 steps and lift it by 500,000 steps. However, these runs often exhibited bursts of correct behavior followed by periods of instability during the later stages of training.

In contrast, other runs exhibited more balanced learning and exploration phases, where the robot consistently learned to pick up and lift the cube. Despite this success, these runs revealed a lack of direction, as the robot would lift the cube without having a clear goal. To address this, we implemented a target position for the cube, incentivizing the robot to transport the cube to a specified point in space.

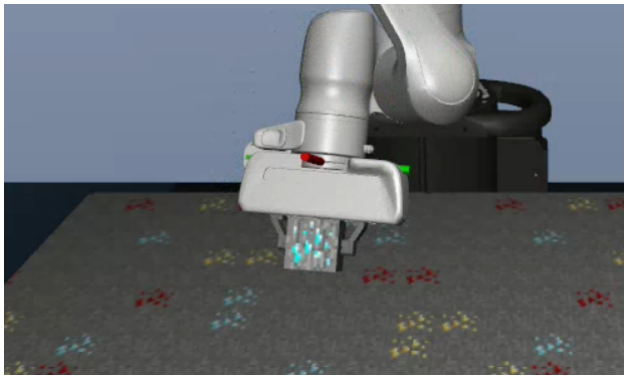


[fig 5.1] Values of the different reward functions tried during the training.

5.2 Challenges: The "Balancing Maneuver"

One of the main challenges we observed was what we termed the "balancing maneuver." During this maneuver, the robot arm would

attempt to balance the cube on one of its vertices instead of lifting it fully off the table. This behavior occurred because balancing the cube allowed the robot to lift its center, which satisfied the reward criteria, without actually achieving the task objective.



[fig 5.2] The Robot balancing the cube on one of it's angles.

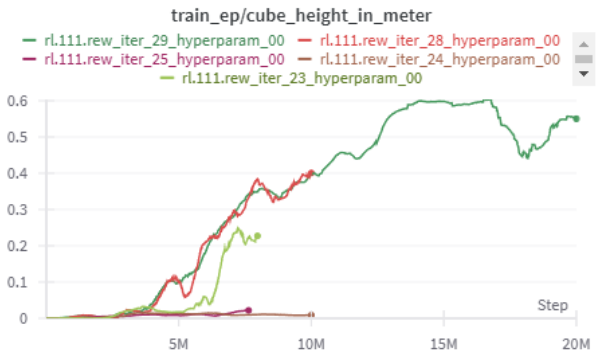
To address this, we introduced an angle factor into the reward function. This factor penalized the robot if the cube was twisted excessively from its initial orientation, even making the reward negative in such cases. This modification helped the robot correct its behavior, and after completing the exploration phase, it began exploiting the correct path to lift the cube and transport it to the target position.

5.3 Enhanced Observation and Data Logging

To better monitor key metrics during training, we modified the base functions in the MuJoCo-py code to log additional parameters. Specifically, we tracked:

- **Cube height in meters:** Allowed us to monitor how consistently the robot was lifting the cube off the table.
- **Cube distance to target in meters:** Helped assess how effectively the robot transported the cube toward the target position.
- **Grasp count:** Provided insight into how often the robot was successfully grasping the cube during training episodes.

These metrics were logged alongside the reward plot and the episode success plot in Weights and Biases (WandB). This combination of visualizations gave us a comprehensive view of the training process and allowed for quicker iteration and refinement of reward functions.

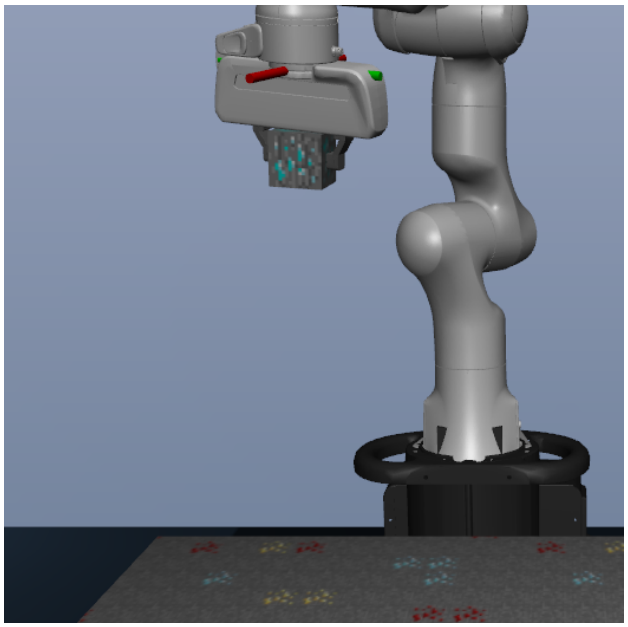


[fig 5.3] Performance metrics during training, showcasing the robot's progression.

5.4 Insights From Iterative Refinement

The iterative, trial-and-error approach proved to be more efficient than attempting to design a perfect reward function from the start. By observing the plotted metrics in real time, we were able to identify incorrect behaviors and adjust parameters or the reward function accordingly, without waiting for the complete training process to finish (which could take 5–8 hours per run).

This approach not only accelerated our workflow but also reinforced our understanding of how reinforcement learning models operate. Combining theoretical insights with practical experimentation allowed us to converge on an optimized solution where the robot successfully grasped the cube, lifted it off the table, and transported it to the target position approximately 55 cm above its starting location.



[fig 5.4] The Robot arm lifting the cube up to the target position.

After finalizing the reward function, we conducted additional iterations with various hyperparameter values. However, deviations from the default settings often led to instability during training. To ensure that our results were robust and not a fluke, we introduced random noise to the robot arm's starting position in each simulation iteration. Interestingly, this modification smoothed the reward plot and slightly accelerated convergence. We attribute this behavior to the added noise encouraging the model to generalize more effectively, thereby avoiding overfitting to local optima.

During the final training run, we extended the training steps to 20 million. Around the 15 million mark, the reward plateaued and became unstable, indicating that the robot was no longer consistently exploiting the learned optimal behavior. Instead, it reverted to an exploratory phase, testing various suboptimal strategies. This behavior suggests that the model may have encountered limitations in its ability to balance exploitation and exploration over extended training, possibly due to overfitting or saturation in the learning process.

6 Challenges Encountered

A non-exhaustive description of the challenges encountered during the project

The installation and configuration of the `mujoco-py` library presented significant challenges, which impacted the progress and engagement with the project. Below, we detail the extensive efforts undertaken to set up the required environment and the issues encountered at each stage.

6.1 Attempts to Install `mujoco-py`

The installation of `mujoco-py`, a library that is no longer actively maintained, proved to be exceptionally challenging. The following approaches were undertaken:

6.1.1 Native Installation on Ubuntu

A native installation was first attempted on an Ubuntu partition. This process was hindered by dependency issues and GPU driver compatibility errors. Despite using multiple versions of Ubuntu (three in total), each requiring a full system re-installation, the errors persisted. Different GPU drivers were tested across these installations, but no combination resolved the issues. The errors encountered were diverse, including unknown GPU-related errors, which lacked sufficient documentation or online solutions.

6.1.2 Containerized Installation Using Docker

As an alternative to a native installation, an attempt was made to use an Ubuntu container within Docker. While this method provided some isolation, it introduced new types of errors. Each error was addressed incrementally,

but unknown errors without searchable solutions posed a significant roadblock.

Subsequently, a preconfigured VNC-ready Ubuntu Docker container was used, which successfully launched `mujoco-py`. However, the rendered simulation failed to display the robot arm or rag doll, leaving the setup unusable despite extensive debugging efforts.

6.1.3 Virtual Machine Setup on Windows

Another strategy involved setting up an Ubuntu virtual machine on Windows using VirtualBox. This mirrored the steps taken in the native installation but yielded the same unresolved errors, further emphasizing the limitations of the process.

6.1.4 Switching to the Modern MuJoCo Library

Given the difficulties with `mujoco-py`, the modern MuJoCo library, which is actively maintained, was tested as an alternative. Installation of the modern MuJoCo library was straightforward, requiring only a simple `pip install mujoco` command. The included demos ran successfully, demonstrating the library's functionality.

However, the project code provided was specifically designed for `mujoco-py` and was incompatible with the updated library. Initial attempts were made to refactor the code for compatibility with the new library, achieving partial success. Unfortunately, progress was halted by library-related errors that lacked descriptive documentation, creating another impasse.

6.2 Broader Implications and Observations

The challenges encountered during installation were not unique to this experience. Anecdotal evidence from peers indicated widespread difficulty in setting up `mujoco-py`. In class, only two students managed to install the library successfully. Conversations with students from prior years revealed similar struggles, with some unable to complete the installation altogether.

The time-consuming nature of these attempts, which often spanned multiple days and exceeded 40 hours in total, detracted from the core objectives of the project. For many students, including us, this created a significant barrier to engaging meaningfully with the project. While the project itself was conceptually interesting, the overwhelming focus on troubleshooting installation issues overshadowed the intended learning objectives.

To address this issue, a previous year's student graciously offered access to their home computer, where `mujoco-py` was already installed after considerable effort. This allowed work to continue but underscored the impracticality of the installation process.

6.3 Recommendations for Future Iterations

Several measures could improve the experience for future cohorts:

- **Transition to the Modern MuJoCo Library:** Updating the project codebase to use the actively maintained MuJoCo library would eliminate the reliance on the deprecated `mujoco-py`.
- **Preconfigured Development Environments:** Providing preconfigured Docker containers or virtual machines with `mujoco-py` or its modern equivalent installed could save significant time and frustration.
- **Clearer Guidance and Support:** Offering detailed installation instructions and troubleshooting steps tailored to the specific requirements of the project would alleviate many common issues.
- **Access to Ready-to-Use Lab Com-**

puters: Doing the project from start on lab computers with the environment preinstalled would allow students to focus on the project's objectives rather than environmental setup.

6.4 Personal Reflection Of Our Group

The installation process for `mujoco-py` was a significant source of frustration and detracted from the learning experience. While this aspect of the project highlighted real-world problem-solving challenges, the disproportionate amount of time spent on setup, compared to the project objectives, diminished overall engagement. Streamlining the setup process in future iterations would ensure that students can direct their efforts toward meaningful project outcomes.

7 Insights and Future Work

A brief summary of Insights and possible improvements for future work on RL

7.1 Key Insights

The project provided valuable insights into the challenges and opportunities associated with reinforcement learning (RL) for robotic control tasks. Through iterative experimentation and troubleshooting, the following key points were identified:

- **Reward Function Design:** Crafting an effective reward function is critical to guiding the agent toward desired behaviors. The iterative approach taken here, combining proximity-based rewards, penalties for undesired rotations, and explicit target goals, proved essential in achieving consistent performance.
- **Exploration vs. Exploitation:** Balancing exploration and exploitation during training required careful adjustment of hyper-parameters. While the default settings were generally effective, tuning learning rates and discount factors highlighted the trade-offs between stability and learning speed.
- **Environmental Factors:** The agent's behavior underscored the importance of environment-specific considerations. For example, penalizing cube rotations was necessary to address unintended solutions that exploited the reward system without fulfilling the task requirements.
- **Observability and Metrics:** Modifying the simulation to include additional metrics, such as cube height and grasp count, significantly improved the ability to diagnose and correct issues during training. These metrics provided actionable insights into the agent's learning progress.

Despite the challenges encountered, the project demonstrated the potential of RL techniques to solve complex robotic tasks. The hands-on experience reinforced a deeper understanding of RL principles and their practical implementation.

7.2 Recommendations for Future Efforts

Building upon the lessons learned during this project, several recommendations are proposed to streamline future efforts and improve outcomes:

- **Improved Setup and Tooling:** Transitioning to the modern `MuJoCo` library as said before, would simplify the

setup process and reduce time spent on troubleshooting outdated dependencies. Providing pre-configured environments or cloud-based solutions would further mitigate installation challenges.

- **Reward Function Optimization:** Future projects could explore automated methods for designing and tuning reward functions, such as using genetic algorithms or neural architecture search techniques, to reduce reliance on trial-and-error approaches.
- **Enhanced Metrics and Visualization:** Integrating comprehensive metrics into the training pipeline, along with real-time visualization tools, would enable more precise monitoring of agent behavior and accelerate debugging.
- **Transfer Learning:** Exploring transfer learning approaches could shorten training times by leveraging pre-trained models or reusing knowledge from related tasks.
- **Robustness Testing:** Expanding the testing framework to include diverse scenarios, such as varying cube positions or unexpected obstacles, would improve the robustness and generalization of the trained policy.
- **Documentation:** Developing detailed documentation for the setup, implementation, and troubleshooting processes would enhance accessibility for future teams.

7.3 Final Thoughts

This project highlighted the complexities inherent in applying RL to real-world robotic problems. While the challenges were significant, they underscored the importance of resilience, creativity, and critical thinking in tackling technical obstacles. By addressing the identified limitations and building on the successful aspects of this work, future efforts can achieve even greater success in advancing RL for robotics applications.

8 Conclusions and Acknowledgments

The journey through this project has been both challenging and rewarding. While the initial goals seemed straightforward—training a robot arm to grasp and lift a cube—the practical execution involved navigating numerous complexities, from designing an effective reward function to overcoming significant setup and tooling hurdles. Despite these challenges, the project provided a valuable opportunity to delve into the practicalities of reinforcement learning (RL) and apply theoretical concepts to a real-world problem.

One of the most significant lessons learned was the importance of crafting a well-balanced reward function. Early versions of our reward function, while functional, often led the agent to exploit loopholes, such as balancing the cube on its vertex rather than lifting it properly. Through iterative design and testing, we developed a reward system that guided the robot arm to consistently achieve the task while avoiding unintended behaviors. This process emphasized the iterative nature of machine learning, where trial and error are not just expected but necessary for success.

The project also underscored the critical role of observability during training. By introducing custom metrics such as cube height, grasp counts, and distance to the target, we gained deeper insights into the agent's learning process. These additions allowed us to diagnose issues and

make informed adjustments without waiting for entire training cycles to complete, greatly accelerating our progress.

On a practical note, the technical challenges—particularly the difficulties with the deprecated `mujoco-py` library—added a layer of frustration but also highlighted the importance of adaptable problem-solving skills. Ultimately, leveraging a preconfigured environment allowed us to focus on the RL task itself rather than the setup process, a lesson we hope will inform future iterations of this project.

Looking forward, there are several areas where the work could be extended or improved. For example, refining the reward function further or experimenting with automated tuning methods could lead to even more robust solutions. Incorporating transfer learning or pre-trained models might also shorten training times and expand the complexity of tasks the robot can handle.

Lastly, we would like to express our gratitude to those who supported us throughout this project. Special thanks go to our classmates and a previous-year student who kindly offered their resources and insights to help us overcome technical roadblocks. Additionally, we appreciate the teaching team for their guidance, even as we navigated some unexpected hurdles. While the process was not without its frustrations, the project ultimately reinforced our understanding of reinforcement learning (pun intended) and gave us a deeper appreciation for the challenges of robotic control tasks.

In conclusion, while there were moments of difficulty and setbacks, the experience has been overall educational. It allowed us to blend theory with practice and provided a foundation for tackling similar challenges in the future. We leave this project with a sense of accomplishment and a stronger grasp of RL concepts, ready to apply these skills to new and exciting problems.