G. Ansaldo

Collaborative Gym:
A Simulation Benchmark for Multi-Robotic Tasks

Collaborative Gym:

A Simulation Benchmark for Multi-Robotic Tasks

By

G. Ansaldo

Master Thesis

in partial fulfilment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Department Maritime and Transport Technology of Faculty Mechanical, Maritime and Materials Engineering of Delft University of Technology to be defended publicly on Thursday 19th of January, 2023 at 3:30 PM

Student	5350859	
MSc Track:	Multi-Machine Engineering	
Report	2022.MME.8743	
Supervisors:	B. Heydari, PhD	
	F. Schulte, PhD	
Thesis	F. Schulte, PhD	TU Delft Committee Chair,
	Jason K. Moore, PhD	TU Delft Committee Member
	Christoph Schmidt, PhD Student	TU Delft Committee Member
	B. Heydari, PhD	NEU Supervisor, Northeastern
Date:	January 13, 2023	

Contents

List of Figures

List of Tables

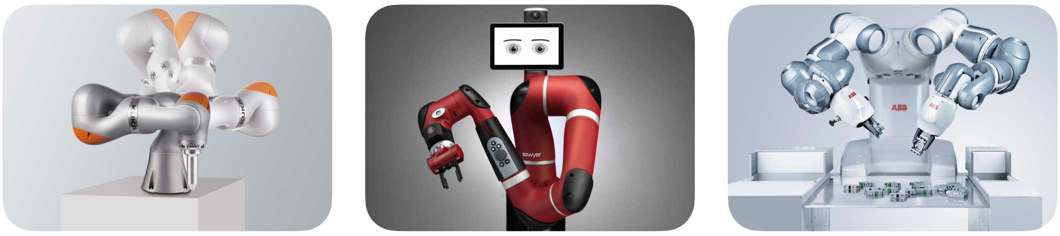
Abstract

The design of multi-robot systems has gained increasing attention in recent years. The field of cooperative Multi-Agent Robot Systems (MARS) has shown the potential to provide reliable and cost-effective solutions to a wide range of automated applications. Communication and coordination between autonomous agents require robust and intelligent control systems in order to achieve high-quality performance. This paper presents Collaborative Gym, an open-source, physics-based simulation framework for multi-robot interaction. This simulation environment differs from existing robotic simulation environments in that it is designed to model the interaction between multiple robots. Despite the presence of a large number of single robotic environments, multi-robotic simulation environments for reinforcement learning are rare. Collaborative Gym contains four simulated tasks in which different commercial robots work in collaboration: poking, lifting, balancing, and passing. For each of the four tasks, baseline policies are presented for various combinations of commercial robots which have been trained using reinforcement learning. The study demonstrated that Collaborative Gym is a promising open-source framework for the development of multi-robotic collaborative robotic tasks.

Introduction

Modern robotics involves robot-robot cooperation in unstructured environments. For instance, in automated logistics and manufacturing scenarios, such as warehouses, distribution centers, and automotive companies, repetitive and sequential operations can be performed more efficiently by transferring objects between robots equipped with object handover abilities (Costanzo, De Maria, and Natale 2021). Recent years have seen a surge in research in the field of cooperative Multi-Agent Robot Systems (MARS). Having efficient coordination among autonomous agents in order to complete tasks is of primary concern in order to achieve high quality performance (Ismail, Sariff, and Hurtado 2018). Among the most challenging aspects of achieving proper coordination is the design of control architectures that command the robots. It is true that robust and intelligent control systems are required to efficiently and effectively communicate and coordinate among agents in order to accomplish a variety of tasks. Therefore, developing the control architecture has been identified as one of the most relevant aspects. Recent developments in Artificial Intelligence (AI) and Reinforcement Learning (RL) may make it possible to develop robust and efficient intelligent control systems capable of enabling robust and efficient coordination and collaboration between robots in unstructured environments.
RL is a machine learning technique that allows agents to learn from their actions and experiences in an interactive environment by trial and error. The RL process requires a tremendous amount of ”trial and error” episodes, or interactions with an environment before a good policy can be learned. For this reason, the use of simulations is essential to achieve results in a cost-effective and timely manner. Many simulation environments are currently available for RL research. Some of the most common and popular environments are OpenAI Gym, Meta-World, and DeepMind Control Suite. Despite mostly being focused on single-agent environments, these benchmarks are excellent for training RL policies in different scenarios. However, multi-robot environments are yet to be fully explored and could be of great assistance for the development of multi-agent robotic systems, as well as the testing of new multi-agent reinforcement learning techniques.

This project primarily focuses on robotic arms, specifically cobots. In contrast to traditional industrial robot applications, collaborative robots, or cobots, come into direct contact with humans. A cobot's safety can be assured by lightweight construction materials, rounded edges, the inherent limits of speed and force, or via sensors and software that ensure safe behavior.



(a) (b) (c)

Figure 1: Example of Collaborative Robots (a) KUKA LBR iiwa (b) Sawyer Robot (c) ABB Yumi

This study contributes to the development of the current RL environment library by introducing a simulation benchmark for MARS. In this paper, Collaborative Gym[1], an open-source, physics-based simulation framework for multi-robot interaction, is presented. Specifically, this project attempts to design various multi-robotic tasks that require collaboration and coordination among robots. Collaborative Gym differs from existing robotic simulation environments in that it focuses on modeling the interaction between multiple robots.

This project is divided into four parts as shown in Figure 2. Part I, the Conceptual & Technical Research Design, provides a clear picture of how the research is designed and carried out. A problem formulation is presented, followed by the objective of the research project. Consequently, research questions alongside methods of research are presented. Part II provides a clear overview of the literature related to the project. The state of the art on MARS, Multi-Agent Reinforcement Learning (MARL), and RL simulation environments is presented. Part III presents Collaborative Gym. Finally, Part IV is the Results & Discussion stage, in which Collaborative Gym tasks are trained and their results are discussed.

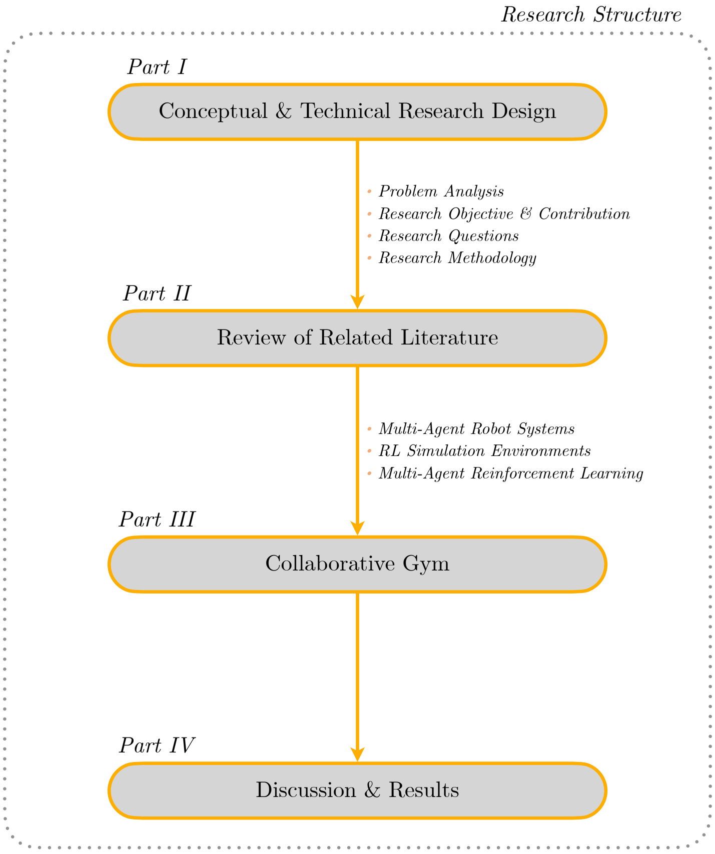

Research Structure
Part I
Conceptual & Technical Research Design
Problem Analysis
Research Objective & Contribution
Research Questions
Research Methodology
Part II
Review of Related Literature
Multi-Agent Robot Systems
RL Simulation Environments
Multi-Agent Reinforcement Learning
Part III
Collaborative Gym
Part IV
Discussion & Results

Figure 2: Research Structure and Project Division

[1] https://github.com/gabriansa/collaborative-gym

Part I

Conceptual & Technical Research Design

1 Problem Analysis & Motivation

Achieving high quality performance requires efficient coordination and collaboration among autonomous agents. Recent advances in artificial intelligence and, specifically, in reinforcement learning offer new opportunities to enhance robot collaboration and coordination. Currently, the majority of robotic simulation environments are developed using OpenAI Gym. The OpenAI Gym architecture forms the basis of many of the standard environments for evaluating continuous control reinforcement learning algorithms. The OpenAI Gym framework contains a collection of benchmark problems, a common interface, and a set of comparison tools for learning control policies for simulated agents. Some of the benchmark environments include Atari games and physics-based locomotion agents. Gym is a standard API for reinforcement learning that has been used to develop a variety of reinforcement learning environments. Figure 3below provides an overview of some of the relevant RL environments in robotics and multi-agent systems.

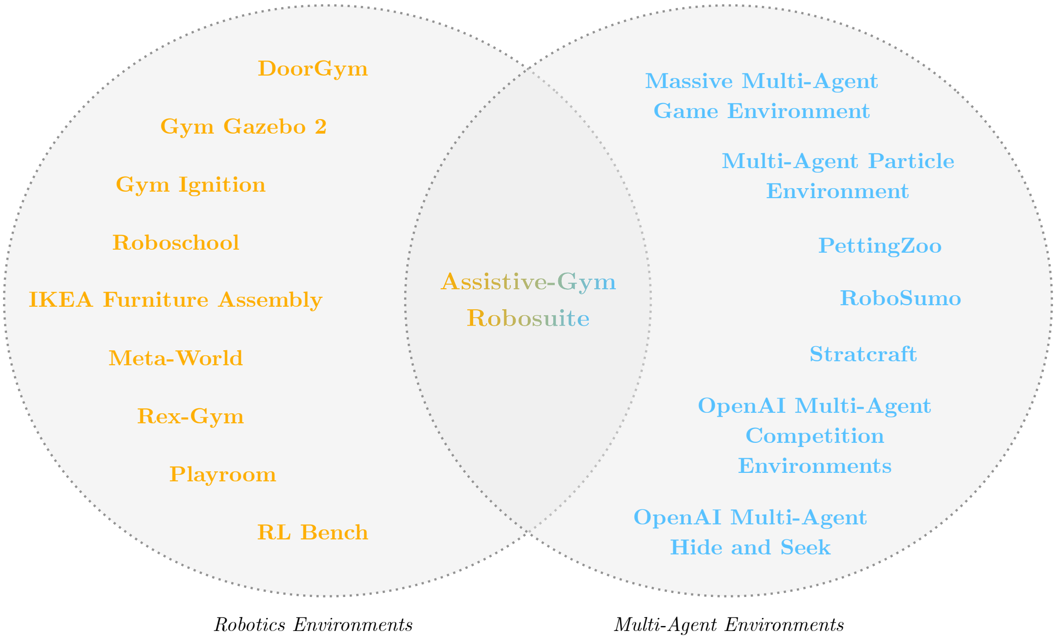


Figure 3: Overview of Relevant RL Environments

Aside from a large number of single robotic environments, multi-robotic simulation environments for RL are scarce. The Assistive Gym and the Robosuite are two environments that provide a taste of robotic multi-agent tasks. On the one hand, Assistive Gym has the ability of training policies for robots collaborating with an active human. On the other hand, Robosuite provides some tasks that require collaborative efforts between two robots. Nevertheless, no RL environment is currently available that is solely dedicated to studying multi-agent robotic tasks.

Problem Statement

Recent advances in robots and RL methods have made it possible to resolve the control challenges faced by cooperative and collaborative Multi-Agent Robotic Systems. Reinforcement Learning simulation environments are capable of efficiently training RL policies for a wide range of robotic tasks, however, they are often limited to training RL policies for single-agent tasks rather than for multi-agent scenarios.

2 Research Objective & Contribution

This research project has a practice-oriented nature. Specifically, this study focuses on the design of a reinforcement learning simulation environment for collaborative robotic tasks. A secondary objective is to solve the various tasks within the designed environment using multi-agent reinforcement learning techniques. Further, a concise goal statement is presented that encapsulates the objective of this study.

Goal Statement

This project aims to contribute to the current library of reinforcement learning environments by developing a Multi-Agent Robotic System environment where robotic arms can coordinate and collaborate to reach a common goal in a variety of tasks.

Collaborative Gym offers several applications for the research community. Firstly, Collaborative Gym can be used as a benchmark for comparing reinforcement learning algorithms for multi-robotic systems. Furthermore, as Collaborative Gym is also based on a similar framework as Assistive Gym, it provides the basis for researchers to create environments and control systems for their own collaborative tasks. Explicitly, through this work the following contributions are made:

- A simulation framework, Collaborative Gym, is developed for multi-robot interaction.
- The various multi-robotic designed tasks can be used as a benchmark to compare control algorithms for multi-agent interaction.

3 Research Questions

For the purpose of achieving the research objective, the following research questions have been formulated:

Research Questions

1. How can the design of a multi-agent robotic collaboration environment be realized?

    (a) What are the characteristics of the multi-agent robotic environment that is to be designed?

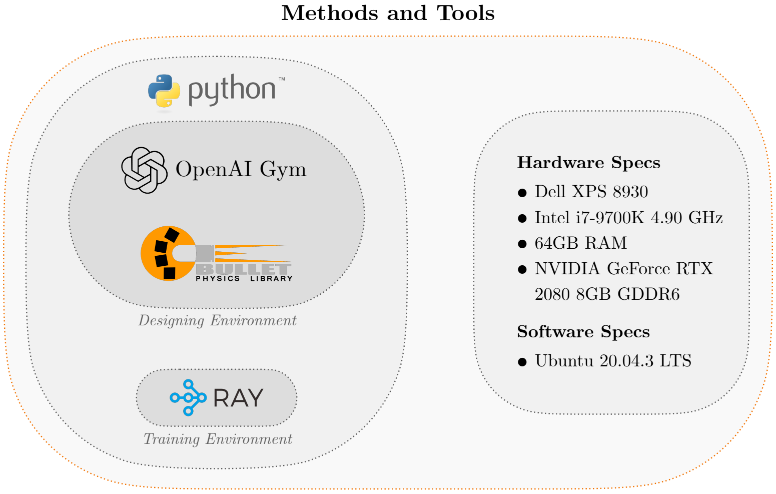
    (b) What are the limitations and assumptions related to the design of the multi-agent robotic environment?

    (c) What are the characteristics of the Observation Space, Action Space, and Reward of the environments to be designed?

    (d) What Multi-Agent Reinforcement Learning methods can be used to solve the various tasks within the designed environment?

4 Research Methodology

A compatible research strategy and a set of tools must be selected to answer the research questions. The research is focused on designing a Multi-Agent Robotic simulation environment. For this purpose, the computer programming language Python will be used alongside multiple relevant packages and frameworks. Research questions are answered with the help of literature, experimentation, and programming. Figure 4 below shows an overview of the research methods and tools that are used to carry out this project.



Methods and Tools

Hardware Specs
- Dell XPS 8930
- Intel i7-9700K 4.90 GHz
- 64GB RAM
- NVIDIA GeForce RTX 2080 8GB GDDR6

Software Specs
- Ubuntu 20.04.3 LTS

Figure 4: Methods and Tools

To train the multiple robots performing collaborative tasks, deep reinforcement learning techniques are utilized. The reason for this is that multi-agent learning deals with problem domains involving multiple agents, which means the search space involved can be extremely large. Small changes in learned behaviors can often result in unpredictable changes in the macro-level properties (”emergent”) of the multi-agent group (Panait and Luke 2005). Indeed, for large and unstructured search spaces, deep reinforcement learning is an optimal technique. The programming language Python is used to develop Collaborative Gym. In particular, the OpenAI Gym framework is adopted. Gym integrates directly with Collaborative Gym, enabling the use of control policy learning algorithms, including deep reinforcement learning. Specifically, for training Collaborative Gym tasks the Python library RAY is utilized (Liang et al. 2018). RAY is an open-source project which allows to flexibly run any compute-intensive Python workload — from distributed training or hyperparameter tuning to deep reinforcement learning and production model serving. Moreover, the open source physics engine PyBullet is at the core of Collaborative Gym (Coumans and Bai 2016). In addition to its ability to run multiple real-time simulations on both CPUs and GPUs, PyBullet is able to simulate cloth and soft bodies, and programmatically create robot models of varying shapes, sizes, weights, and joint limits. With regards to hardware specifications, a Dell XPS 8930 is utilized for running and training Collaborative Gym tasks.

Part II

Review of Related Literature

1 RL Simulation Environments

1.1 OpenAI Gym Framework

In reinforcement learning, an agent learns through interaction with its environment. As a result, testbed environments are necessary in order to test and compare the results of different reinforcement learning algorithms. The OpenAI Gym framework provides a collection of reinforcement learning environments for developing and testing reinforcement learning algorithms. Open AI Gym is an open-source toolkit which provides a range of environments for games, control problems, constructing algorithms, performing control tasks, robotics, text games, and more (Brockman et al. 2016). Among its benchmark environments are Atari games and physics-based locomotion agents. The Open AI framework forms the foundation for Collaborative Gym.

1.2 OpenAI Physics Engines

PyBullet, DART, and MuJoCo are three physics engines commonly used in OpenAI Gym for simulating robotic environments (Coumans and Bai 2016; Lee et al. 2018; Todorov, Erez, and Tassa 2012).

1.2.1 PyBullet

PyBullet is a simulation environment based on Bullet physics-based simulation, which simulates collision detection along with the dynamics of rigid and soft bodies. In addition to machine learning applications, this physics engine has been used for training and validating real robots utilizing physics simulations (Tan et al. 2018; Zeng et al. 2020; Sadeghi et al. 2017; Bousmalis et al. 2018). PyBullet supports loading articulated bodies from URDF, SDF, and other files. This library provides forward dynamics simulation, inverse dynamics computation, forward and inverse kinematics, as well as collision detection and ray intersection queries. Along with physics simulations, PyBullet also supports rendering, including CPU renderers and OpenGL visualizations as well as virtual reality headset support. As an open-source project, PyBullet has attracted a large community of contributors who continue to develop the simulation environment and provide support for beginners (*PyBullet Community* 2013).

1.2.2 DART

The Dynamic Animation and Robotics Toolkit (DART) is an open source library that is collaborative and cross-platform. As part of the library, data structures and algorithms are provided for kinematic and dynamic applications in robotics and computer animation. Featuring a multibody dynamic simulator as well as multiple kinematic tools for motion planning and control, DART has applications in robotics and computer animation. A variety of locomotive environments have been implemented using DART in conjunction with the OpenAI framework (*GymDart* 2018; *DartEnv* 2016).

1.2.3 MuJoCo

Multi-joint dynamics with contact (MuJoCo) is a simulation environment and physics engine dedicated to robotics, biomechanics, animation, and machine learning. MuJoCo is known for its deep learning applications that enable virtual animals and humanoid models to walk and perform other complex movements (Joe Booth and Jackson Booth 2019). In MuJoCo, models are described in a native, XML-based format which can be easily edited by users. It is not possible to install this simulation environment without a license, as opposed to the other simulation environments described in this work.

2 Multi-Agent Robot Systems

There are situations in which Multi-Agent Robot Systems (MARSs) or Multi-Robot Systems (MRSs) may be used to accomplish tasks that would otherwise be difficult for an individual robot to perform, such as when there are uncertainties, incomplete information, asynchronous computations, and distributed control (E. Yang and Gu 2005). There has been considerable interest in MRSs and MARSs over the last decades (Cao, Kahng, and Fukunaga 1997; Matarić 1997; Michaud and Matarić 1998; Balch and Arkin 1998; Asada, Uchibe, and Hosoda 1999; Wiering, Sałustowicz, and Schmidhuber 1999; Van Der Zwaan, Moreira, and Lima 2000; Touzet 2000; Fernandez and Parker 2001; J. Liu and Wu 2018; Matarić 2001; Bowling and Veloso 2003; Elhajj et al. 2003; Iocchi et al. 2003; Matarić, Sukhatme, and Østergaard 2003; Touzet 2004). Several examples of cooperative multi-agent robot applications include soccer robots (Candea et al. 2001; Brandão et al. 2022), unmanned guided vehicles (UGV's) and unmanned aerial vehicles (UAV's) (Rosa et al. 2015). The research in cooperative multi-agent robot systems has focused on three main elements, according to Ismail, Sariff, and Hurtado 2018: (1) the types of agents, homogeneous and heterogeneous, (2) the control architectures, reactive, deliberative, and hybrid, and (3) the type of communication, explicit and implicit. Nevertheless, developing the control architecture has been identified as one of the most relevant aspects for achieving efficient coordination among multi-agent robots. Developing MRSs can be challenging due to the fact that it is not possible to predict all possible situations that robots might encounter as well as to specify their behavior in advance. It is crucial that robots in MRSs learn from their operating environment and adapt to their counterparts. MRSs are therefore faced with the challenge of addressing learning as a key issue. It has become increasingly popular in recent years to extend individual reinforcement learning (RL) to multiagent systems, especially multi-robot systems (MRSs) (Fan et al. 2020; L. Zhang et al. 2020; Hu et al. 2020; Y. Yang, Juntao, and Lingling 2020; G. Ding et al. 2020). Through the use of multi-agent reinforcement learning, participating robots are able to learn the mapping between their states and the actions they take in response to rewards or payoffs obtained by interacting with their environment (E. Yang and Gu 2004). In many ways, MRSs can benefit from RL, where robots are expected to coordinate their behavior in order to achieve their objectives.

3 Multi-Agent Reinforcement Learning Overview

3.1 Multi-Agent Reinforcement Learning Environments

In recent years, a number of studies have investigated Multi-Agent Reinforcement Learning (MARL) (Vinyals, Babuschkin, et al. 2019; Iqbal and Sha 2019; Chu et al. 2019; Cui, Y. Liu, and Nallanathan 2019). As a result, it is necessary to create simulation environments that can be utilized as benchmarks in order to test and evaluate MARL algorithms and techniques. Table 1 below gives a concise overview of the most relevant multi-agent environments that have been developed.

Environment Name	Description	Related Literature
Robosuite	Simulation framework powered by the MuJoCo physics engine for robot learning. It also offers a suite of benchmark environments for reproducible research.	Zhu et al. 2020
Assistive Gym	Physics-based simulation framework for physical human-robot interaction and robotic assistance.	Erickson et al. 2020
Massive Multi-Agent Game Environment	It considers MMORPGs (Massive Multiplayer Online Role Playing Games) the best proxy for the real world among human games: they are complete macrocosms featuring thousands of agents per persistent world, diverse skilling systems, global economies, complex emergent social structures, and ad-hoc high stakes single and team based conflict.	Suarez et al. 2019
PettingZoo	Python library for conducting research in multi-agent reinforcement learning, akin to a multi-agent version of Gym.	Terry et al. 2021
RoboSumo	Sumo-wrestling between two ants using continuous control.	Al-Shedivat et al. 2017
Stratcraft	StarCraft is a 1998 military science fiction real-time strategy game.	Usunier et al. 2016 Vinyals, Ewalds, et al. 2017
OpenAI Multi-Agent Hide and Seek	In our environment, agents play a team-based hide-and-seek game. Hiders (blue) are tasked with avoiding line-of-sight from the seekers (red), and seekers are tasked with keeping vision of the hiders. There are objects scattered throughout the environment that hiders and seekers can grab and lock in place, as well as randomly generated immovable rooms and walls that agents must learn to navigate.	Baker et al. 2019
OpenAI Multi-Agent Competition Environments	A collection of various continuous control, multi-agent tasks.	Bansal et al. 2017

Table 1: Overview of the most relevant Multi-Agent Reinforcement Learning Environments

Comparatively to existing multi-agent simulation environments, Collaborative Gym solely focuses on multi-robot systems, thus enabling multiple robots to collaborate on a variety of tasks to achieve a common goal.

3.1 MARL Techniques

In terms of task type, it is possible to classify Multi-Agent Reinforcement Learning into three categories, namely fully cooperative, fully competitive, and hybrid (K. Zhang, Z. Yang, and Başar 2021; Buşoniu, Babuška, and Schutter 2010; Du and S. Ding 2021). For instance, an agent's reward function is the same in a fully cooperative random game. Therefore, the objective of agents is to maximize their mutual returns. It is however possible to have a complete competition scenario when the objectives of two agents are opposite.

- *Fully cooperative learning*: A fully cooperative random game has the same reward function and learning objective for each agent. In the same manner as single agents environments, in a multi-agent setting agents are likely to pursue greedy strategies in order to maximize their returns. However, the agents cannot make their decisions independently, since they share the same objective, they must consider the other agents and cooperate as a group.

- *Fully competitive learning*: In contrast with complete cooperation, a complete competition environment has many similar characteristics, with the exception that the reward function is the opposite for each agent. It is also possible for competition to arise when there are more than two agents involved. However, most of the literature concerning RL in fully competitive games pertains only to two-agent games.

- *Hybrid learning*: In contrast to fully cooperative and competitive agents, hybrid agents tend to be selfish since their reward function is not restricted. A large number of algorithms in this category are devoted exclusively to static problems and are based on the concept of game theory equilibrium.

As part of a Multi-Agent System (MAS), multiple agents coexist in the same environment and all learn simultaneously. It is not uncommon for one agent's behavior to be affected by the behavior of other agents in the same environment. As a result, Multi-Agent Reinforcement Learning (MARL) methods are typically utilized to the solution of such problems. In general, there are three types of MARL methods: distributed independent learning, centralized learning, and distributed collaborative learning.

- *Distributed independent learning methods*: MARL research can be effectively approached through distributed independent learning methods (Wang, Xie, and Atanasov 2022; Wang, Xie, and Atanasov 2021). Distributed independent learning involves each agent taking other agents into account as part of their environment and learning strategies independently (Weiß 1995). Due to the forced decomposition of the decision process into multiple Markov decision processes, this method significantly reduces the representation of the state-action space (Lauer and Riedmiller 2000). However, there are two disadvantages to this type of learning:

    1. There is an absence of a coordination mechanism between agents
    2. A relatively poor and suboptimal strategy is obtained

  As a way to mitigate the absence of coordination mechanisms, the limited information of other environmental agents can be used as input to the learning process (Yu, Dong, et al. 2020).

- *Centralized learning methods*: As opposed to the distributed independent learning method, the centralized learning method requires each agent to communicate with a central controller and to choose actions in accordance with the controller's instructions (Khan et al. 2018). To control the learning process synchronously, the central controller must also be able to perceive the global environment (Sharma et al. 2021).

There are several advantages and disadvantages to this method (Yu, Dong, et al. 2020):

Advantages:

1. As a Markov decision-making process, the global decision-making process is regarded as a static and closed environment where all agents learn in the same manner
2. It is possible to develop a globally optimal strategy with sufficient learning time

Disadvantages:

1. A slow convergence rate
2. Dimensional disaster
3. Scalability and robustness issues

In relation to the three disadvantages, the centralized learning method usually faces the issue of slow convergence since the agent must explore all joint state-action spaces when the problem scale is large. With regards to dimensional disaster, the joint state-action space increases exponentially as the scale of the problem increases, which leads to an exponential increase in the number of agents. As a final point, centralized learning methods typically face scalability and robustness issues (Lyu et al. 2021). Since the agent is limited in its observation ability, it is able to perceive the environment only locally, and it cannot obtain information about the surrounding environment at a global level. Considering the limited communication capabilities of agents, centralized learning depends on effective communication between the central controller and all agents.

- *Distributed collaborative learning methods*: Collaborative distributed learning methods combine the benefits of distributed and centralized methods of learning. As part of the method, a collaborative relationship is maintained between the agents and collaborative learning is introduced between the agents based on distributed independent learning (Pawar and Leshem 2021; Kar, Moura, and Poor 2013). As a result, this method improves the performance of the Multi-Agent Learning System (MALS) at the same time as alleviating the dimensional disaster, robustness, and scalability issues associated with centralized learning techniques.

Collaborative Gym provides a variety of tasks that utilize distributed collaborative learning methods, where agents learn independently while sharing information with other agents and/or sharing reward functions. Nevertheless, Collaborative Gym allows users to design cooperative, competitive, and hybrid tasks that can be trained using either distributed, centralized, or distributed collaborative learning methods.

Part III

Collaborative Gym

1 Collaborative Gym Comprehensive Overview

The Collaborative Gym simulation framework provides high-level interfaces for creating and customizing simulation environments for multirobot systems. In addition to integrating directly into the open source PyBullet physics engine, Collaborative Gym environments are built upon the OpenAI Gym interface which allows the use of existing control policy learning algorithms including deep reinforcement learning. A comprehensive analysis of Collaborative Gym is further provided in terms of its control mechanisms, action and observation spaces, and policy learning.

1.1 Actions & Observations

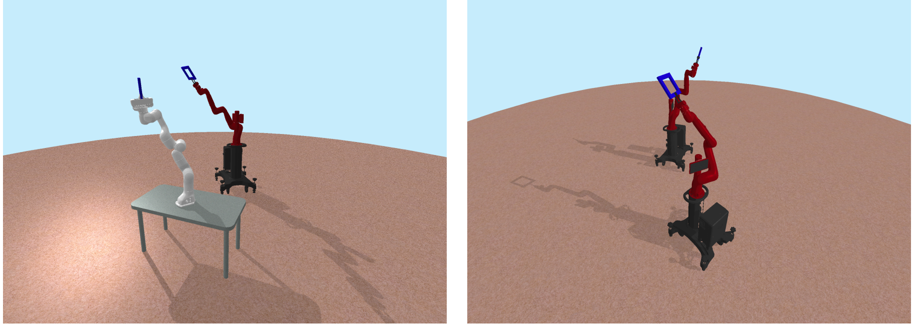
A position control system is provided by Collaborative Gym for controlling the various robots. Actions for each robot's $n - DoF$ arm are represented as changes in joint positions, $\Delta J \in R^n$ depending on the robotic arm. Additionally, a binary variable that selects between gripping and ungripping simplifies the gripping action. Physical limits of an arm affect the ability of a robot to perform actions at any given location. Indeed, each joint of a robot is able to rotate till a certain limit. Actions for the robots' arm are defined as $a = (\Delta J_0, \Delta J_1, \Delta J_2, ..., \Delta J_n, g) \in A$, where $J_0, J_1, J_2, ..., J_n$ are the robot's joint angles and $\Delta J_0, \Delta J_1, \Delta J_2, ..., \Delta J_n$ are delta joint angles that are added to the robot's current joint angles in order to move the robot's arm. Additionally, g is a binary decision variable that allows a robot to choose between grasping and ungrabbing objects in the environment. An object is grasped by the robot when it is close enough to it, and the binary variable g assumes value of 1. For the various robots in the environment, gripping has been simplified to facilitate a simpler and faster learning process. In other words, robot gripping is a complex task that falls within a separate research field, therefore, it was simplified for use in the Collaborative Gym project. Additionally, the process by which gripping occurs plays little role in this study. It is important to note that not all Collaborative Gym tasks require gripping, and therefore the g action may sometimes be neglected depending on the nature of the task.

Robots record observations from the state of the system at each time step, perform actions according to the control policy, and receive rewards at the end of the time step. Observations given to the robots are comparable to those obtained in a real-world scenario involving multiple robots. As an example, these include the position and orientation of the robot's end effector in 3D, the position of the robotic arm's joints in 7D, and the position of the task relevant objects in 3D. Positions are defined in terms of a global coordinate system. It is important to note that observations are task-dependent and may vary according to the study's objectives.

1.2 Collaborative Tasks

A set of four tasks are readily available on Collaborative Gym. These multi-robotic tasks, ranging from simple to more complex, which require collaboration and coordination among robots are further described in details.

- ***1. Poking Task***: In this task one robot holds a stick object while the other holds a "donut" shaped object. The goal is for the two robots to coordinate and insert the stick inside the donut hole. Figure 5 below shows a visual representation of this task.



(a) Panda and Sawyer Robots

(b) Sawyer Robots

Figure 5: Poking Task Environment

The action space for this task is defined as changes in joint positions ΔJ. Since grasping does not occur in this task, the action binary variable g is not utilized. The observation space for each robot includes the position of the robotic arm's joints in 7D and the position and orientation of the stick and donut objects. Table 2 and Table 3 below gives a detailed overview of the action and observation spaces for each robot.

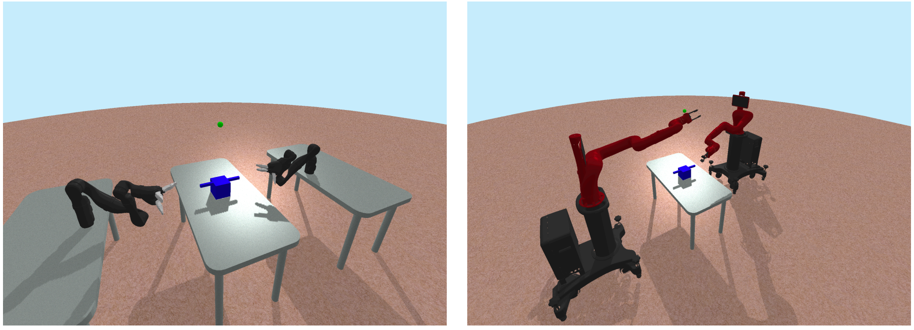
Robot 1		**Robot 2**	
Action	*Description*	*Action*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*

Table 2: Action Space for the Poking Task

Robot 1		**Robot 2**	
Observation	*Description*	*Observation*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Stick Object Position and Orientation	*pos x*	Stick Object Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Donut Object Position and Orientation	*pos x*	Donut Object Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*

Table 3: Observation Space for the Poking Task

- ***2. Lifting Task***: This task consists of two robots that need to coordinate to lift a heavy pot object. Two robots are required to lift the object since it is too heavy to be lifted by one robot. Indeed, the goal is for the two robots to coordinate and lift the pot object to a specific target position. Figure 6 below shows a visual representation of this task.



(a) Jacos Robots

(b) Sawyer Robots

Figure 6: Lifting Task Environment

The action space for this task is defined as changes in joint positions ΔJ. Since grasping occurs in this task, the action binary variable g is required. The observation space for each robot includes the positions of both robotic arm's joints in 7D, the position and orientation of the robot's end effectors in 3D, the status of each gripper (gripping or ungripping), the position and orientation of the pot object and its handles, and the target position. Table 4 and Table 5 below gives a detailed overview of the action and observation spaces for each robot.

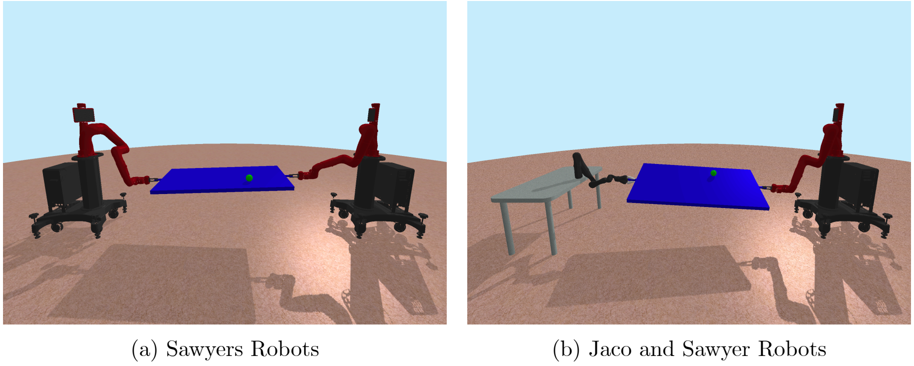
Robot 1		**Robot 2**	
Action	*Description*	*Action*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Grasping	*g*	Grasping	*g*

Table 4: Action Space for the Lifting Task

Robot 1		**Robot 2**	
Observation	*Description*	*Observation*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 1)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Joint Angles (Robot 2)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Gripper Status (Robot 1)	*binary* {*0,1*}	Gripper Status (Robot 1)	*binary* {*0,1*}
Gripper Status (Robot 2)	*binary* {*0,1*}	Gripper Status (Robot 2)	*binary* {*0,1*}
End Effector Position and Orientation (Robot 2)	*pos x*	End Effector Position and Orientation (Robot 2)	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Handle 1 Position and Orientation	*pos x*	Handle 1 Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Handle 2 Position and Orientation	*pos x*	Handle 2 Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Pot Position and Orientation	*pos x*	Pot Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Target Position	*pos x*	Target Position	*pos x*
	pos y		*pos y*
	pos z		*pos z*

Table 5: Observation Space for the Lifting Task

- ***3. Balancing Task***: In this task, two robots must coordinate in order to balance a moving sphere on a flat surface. The goal is for the two robots to coordinate and keep the sphere at the center of the board as long as possible. Figure 7 below shows a visual representation of this task.



(a) Sawyers Robots

(b) Jaco and Sawyer Robots

Figure 7: Balancing Task Environment

The action space for this task is defined as changes in joint positions ΔJ. Considering that grasping does not occur in this task, the action binary variable g is not taken into account. The observation space for each robot includes the position of both robotic arm's joints in 7D, the position and orientation of the center of the balancing board in 3D, the position orientation of the sphere object, and the linear and angular velocities of the sphere object. Table 6 and Table 7 below gives a detailed overview of the action and observation spaces for each robot.

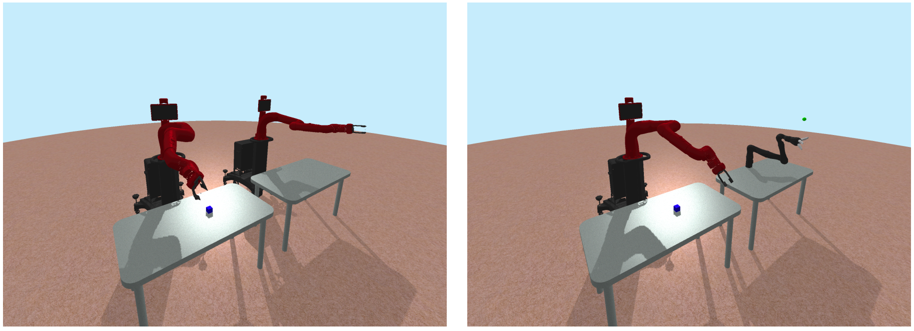
Robot 1		**Robot 2**	
Action	*Description*	*Action*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*

Table 6: Action Space for the Balancing Task

Robot 1		**Robot 2**	
Observation	*Description*	*Observation*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 1)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Joint Angles (Robot 2)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Sphere Position and Orientation	*pos x*	Sphere Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Sphere Linear and Angular Velocity	*velocity x*	Sphere Linear and Angular Velocity	*velocity x*
	velocity y		*velocity y*
	velocity x		*velocity x*
	velocity wx		*velocity wx*
	velocity wy		*velocity wy*
	velocity wz		*velocity wz*
Balancing Board Center Position and Orientation	*pos x*	Balancing Board Center Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*

Table 7: Observation Space for the Balancing Task

- ***4. Passing Task***: This task consists of two robots that need to coordinate to pass a cube object. Specifically, one robot picks up a cube then hands it over to the second robot which consequently needs to move it to a specific target. Indeed, the goal is for the two robots to coordinate and pass the cube object. Figure 8 below shows a visual representation of this task.



(a) Sawyers Robots

(b) Jaco and Sawyer Robots

Figure 8: Passing Task Environment

The action space for this task is defined as changes in joint positions ΔJ. As grasping occurs in this task, the action binary variable g must be used. The observation space for each robot includes the positions of the robotic arm's joints in 7D, the position and orientation of both robot's end effectors in 3D, the status of each gripper (gripping or ungripping), the position and orientation of the cube object, and the target position. Table 8 and Table 9 below gives a detailed overview of the action and observation spaces for each robot.

Robot 1		**Robot 2**	
Action	*Description*	*Action*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Grasping	g	Grasping	g

Table 8: Action Space for the Passing Task

Robot 1		**Robot 2**	
Observation	*Description*	*Observation*	*Description*
Joint Angles (Robot 1)	*J0*	Joint Angles (Robot 2)	*J0*
	J1		*J1*
	. . .		. . .
	Jn		*Jn*
Gripper Status (Robot 1)	*binary {0,1}*	Gripper Status (Robot 1)	*binary {0,1}*
Gripper Status (Robot 2)	*binary {0,1}*	Gripper Status (Robot 2)	*binary {0,1}*
End Effector Position and Orientation (Robot 1)	*pos x*	End Effector Position and Orientation (Robot 1)	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
End Effector Position and Orientation (Robot 2)	*pos x*	End Effector Position and Orientation (Robot 2)	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Cube Position and Orientation	*pos x*	Cube Position and Orientation	*pos x*
	pos y		*pos y*
	pos z		*pos z*
	orient x		*orient x*
	orient y		*orient y*
	orient z		*orient z*
	orient w		*orient w*
Target Position	*pos x*	Target Position	*pos x*
	pos y		*pos y*
	pos z		*pos z*

Table 9: Observation Space for the Passing Task

1.3 Policy Learning

A variety of deep reinforcement learning techniques can be used to train Collaborative Gym tasks. As this environment is built on the OpenAI Gym framework, any deep reinforcement learning technique can be utilized. It is noteworthy that Collaborative Gym is already connected with RLlib, a freely available open source library that offers support for production-level, highly distributed reinforcement learning workloads with simple and unified APIs that can be used for a variety of industry applications. Due to recent research, PPO-based multi-agent algorithms have shown surprisingly high performance in multiple popular multi-agent testbeds (Yu, Velu, et al. 2021). Collaborative Gym utilizes proximal policy optimization (PPO) as the base algorithm for training robots. However, different RL techniques can be utilized, depending on the type of task. In most Collaborative Gym tasks, a fully-connected neural network is employed with two hidden layers and 256 nodes. Instead of using the original activation function *tanh* for PPO, the *relu* function was utilized. This was done since *tanh* often performs better with smaller networks while *relu* gives higher performance with bigger networks. Hyperparameters for PPO have been left to default setting as provided by RLlib (RAY 2022).

On a general note is possible to train the multi-agent RL task in any of the following ways, depending on the nature of the task:

- Cooperative with shared or separate policies and/or value functions.
- Adversarial scenarios using self-play and league-based training.
- Independent learning of neutral/co-existing agents.

For the designed tasks described above, cooperative learning with shared policies and value is utilized.

2 Collaborative Gym GitHub Overview

Collaborative Gym is an open-source simulation framework publicly available on GitHub. For developers, GitHub is one of the most popular resources for sharing code and collaborating on projects. Due to its free and easy-to-use nature, it has become a leading platform for the development of open-source software. In essence, GitHub, Inc. provides an Internet hosting service for software development and version control using Git. In addition to distributed version control, it provides access control, bug tracking, software feature requests, task management, continuous integration, and wikis for each project.
This project is being published on GitHub with the purpose of enabling users to easily install and use Collaborative Gym while providing detailed tutorials on how to use the existing designed environments and also allowing for customization. A comprehensive tutorial is available that guides users through the process of creating custom environments. Collaborative Gym offers this benefit as users can customize the application to suit their individual needs.Appendix A shows the wiki page available on the Collaborative Gym GitHub which includes a high-level overview of the capabilities, step-by-step installation tutorials, integration with RL libraries, and a detailed guide for designing custom environments.

Part IV

Discussion & Results

1 Policy Training Results

As part of the following sections, baseline control policies are presented and analyzed for various robots performing the four collaborative tasks. Table 10 shows the combination of robots used for each task. PPO is used to train robot controllers. It was necessary for the algorithm to run for fifty million timesteps in order to generate the policies. As a practical matter, this number proved to be sufficient for the robots to devise a useful policy. Each episode of each task was 200 timesteps long, which means that the robots could observe 200 observations and take 200 actions in order to complete the task. Each episode ended with the robots and environment being reset to their original positions. It is through a series of training episodes that the robot is able to learn which actions are appropriate for completing the task and which are not until an effective policy is established. An NVIDIA GeForce RTX 2080 with 8GB GDDR6 GPU and an Intel i7-9700K processor was used to train the policies. Depending on the task, training times ranged between 10 hours and 30 hours. Co-optimization is used to accomplish collaboration, where all robots are trained simultaneously with independent control policies. The following subsections give a detailed description of the training results.

Task Name		**Robot 1**	**Robot 2**
Poke Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Panda	Sawyer
Lift Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Jaco	Jaco
Balance Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Jaco	Sawyer
Pass Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Jaco	Sawyer

Table 10: Combination of Robot Types for each Task

1.1 Poking Task Results

To calculate the reward value used to train policies for the poking task, a reward function $R(r_n)$ for each robot r_n with $n = 1, 2$ has been formulated and is defined as follows:

$$\begin{aligned} R(r_n) = &-dist\_stick\_to\_donut - 0.8 * orient\_stick\_donut \\ &- 0.01 * moving\_penalty(r_n) + stick\_is\_inside \end{aligned}$$

where:

- *dist_stick_to_donut*: is the euclidean distance between the center of the donut and the middle of the stick.
- *orient_stick_donut*: is the absolute value of the cosine of the angle between the stick and donut. This makes sure that the stick and the donut are perpendicular to each other.
- *moving_penalty*(r_n): is the norm of the vector of joint actions for each robot r_n. This makes sure that each robot doesn't make unnecessary movements.

- *stick_is_inside*: is a sparse positive reward of value +1 given to each robot whenever both the *dist_stick_to_donut* and *orient_stick_donut* are smaller than some given thresholds.

Figure 9 and Figure 10 below show the learning curves over the fifty million timesteps used to obtain the policy.

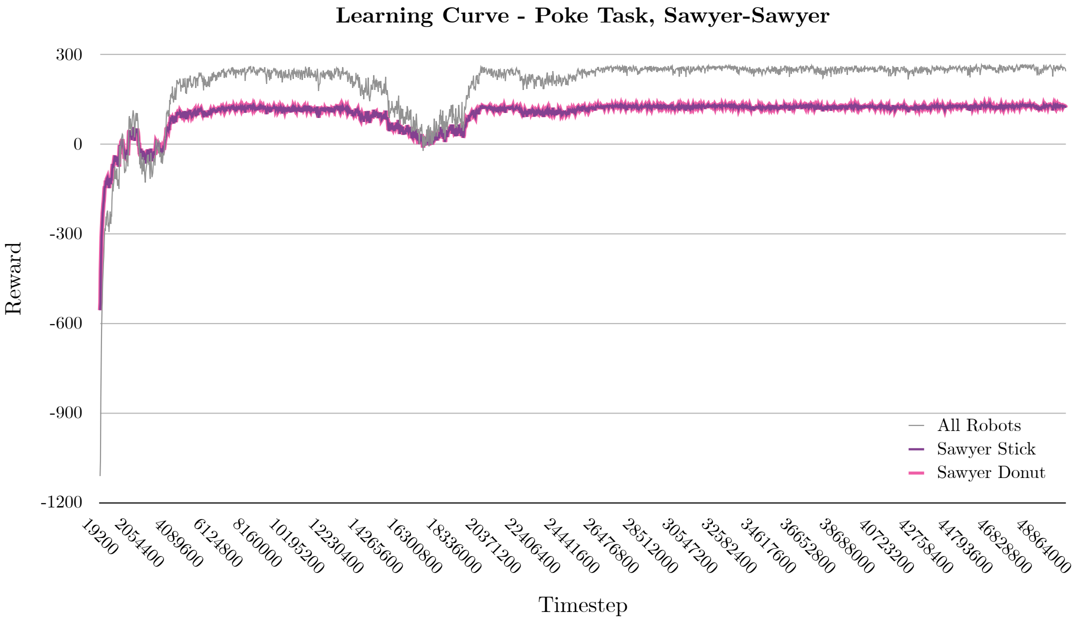


Figure 9: Learning Curve for the Poking Task with Sawyers using PPO

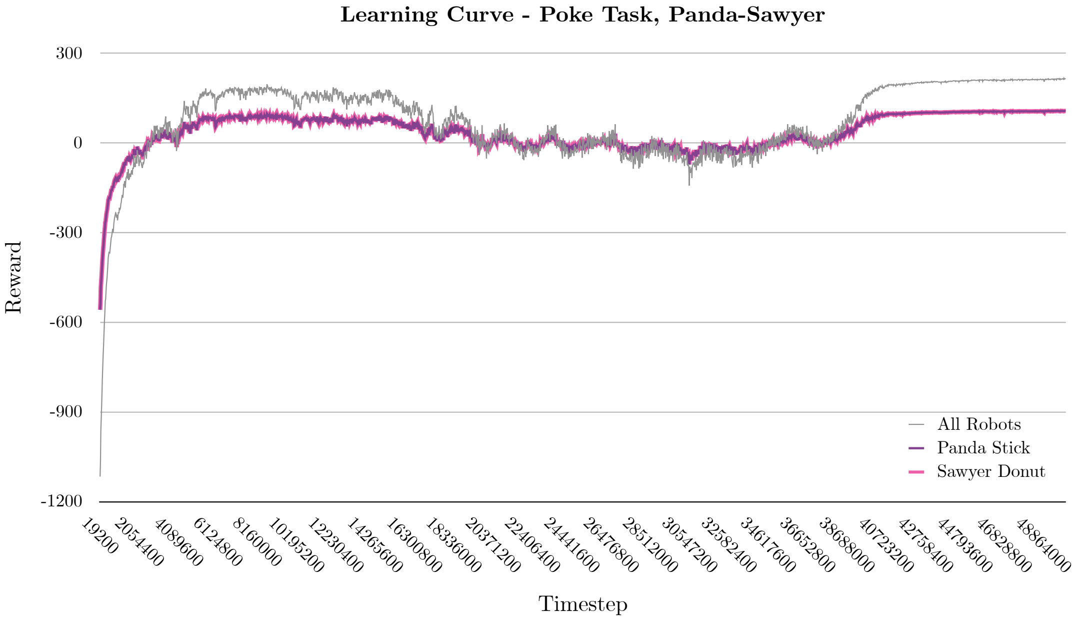


Figure 10: Learning Curve for the Poking Task with Panda and Sawyer using PPO

It can be noted from the two graphs above that an effective policy is learned for both combination of robots within the first 7500000 time-steps. Indeed, the learning curve follows a logarithmic growth pattern, in that improvements come quickly at the beginning but the gains decrease over time. Since positive sparse reward is present in such task, the reward for all robots becomes positive.

1.2 Lift Task Results

To calculate the reward value used to train policies for the lifting task, a reward function $R(r_n)$ for each robot r_n with $n = 1, 2$ has been formulated and is defined as follows:

$$\begin{aligned} R(r_n) = &-dist\_to\_handle(r_n) - 2 * dist\_pot\_to\_target - 0.2 \\ &* pot\_tilt - moving\_penalty(r_n) + gripping\_incentive(r_n) \end{aligned}$$

where:

- $dist\_to\_handle(r_n)$: is the euclidean distance between the end effector of robot (r_n) and one of the handles of the pot object to be lifted.
- $dist\_pot\_to\_target$: is the euclidean distance between the center of the pot object and the target.
- pot_tilt: is the sum of the pot's orientation around the x and y axes. This makes sure that the pot is lifted maintaining a horizontal position.
- $moving\_penalty(r_n)$: is the norm of the vector of joint actions for each robot r_n. This makes sure that each robot doesn't make unnecessary movements.
- $gripping\_incentive(r_n)$: is a positive reward which is a function of the gripping action g, and is defined as follows: $+0.1 * g$. This reward is given to each robot whenever the $dist\_to\_handle(r_n)$ is smaller than a given threshold.

Figure 11 and Figure 12 below show the learning curves over the fifty million timesteps used to obtain the policy.

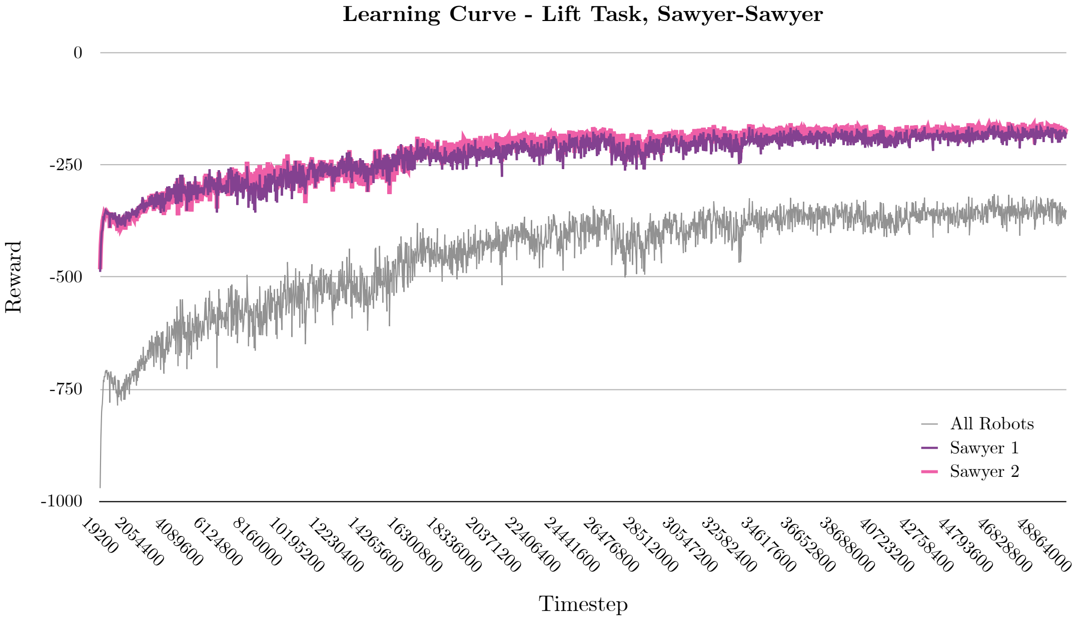


Figure 11: Learning Curve for the Lifting Task with Sawyers using PPO

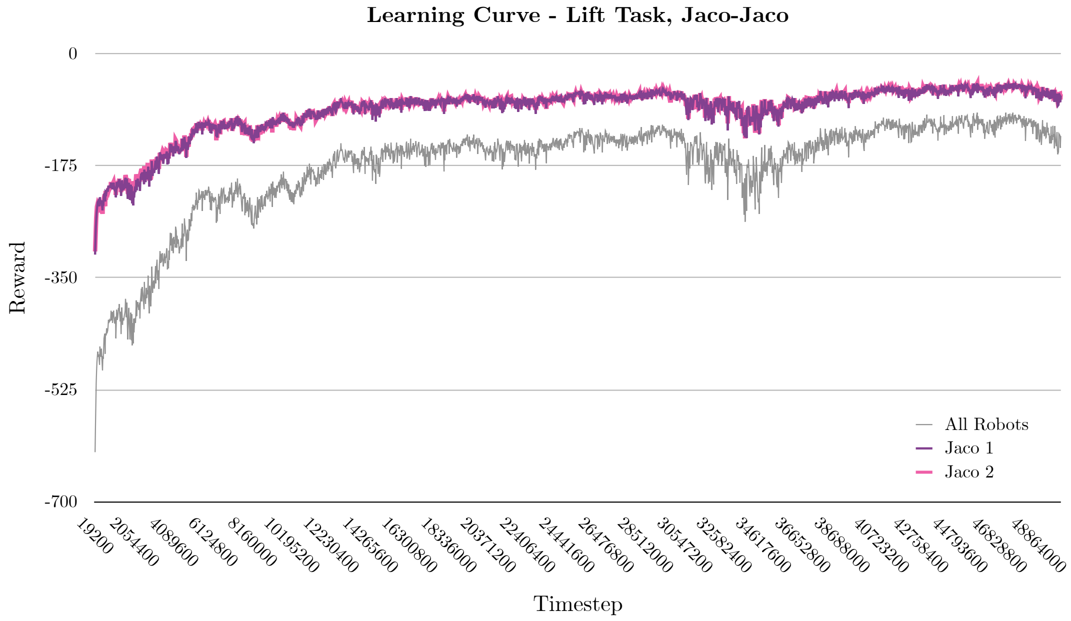


Figure 12: Learning Curve for the Lifting Task with Jacos using PPO

It can be noted from the two graphs above that an effective policy is learned for both combination of robots within the first 2000000 time-steps. Once again, the learning curve follows a logarithmic growth pattern. In this case, the robots learn primarily two actions: (1) reach (2) and pick and place. Based on the learning curve, it can be observed that the robots learn to reach the handles of the pot within the first 2 million time-steps, and then gradually learn to coordinate to grab and place the heavy pot at the desired location.

1.3 Balance Task Results

To calculate the reward value used to train policies for the lifting task, a reward function $R(r_n)$ for each robot r_n with $n = 1, 2$ has been formulated and is defined as follows:

$$R(r_n) = -dist\_sphere\_to\_balancing\_board\_center + penalty\_ball\_fell$$

where:

- $dist\_sphere\_to\_balancing\_board\_center$: is the euclidean distance between the center of the sphere and the center of the balancing board.
- $penalty\_ball\_fell$: is a sparse negative reward of value -1 given to each robot whenever both the ball falls to the ground

Figure 13 and Figure 14 below show the learning curves over the fifty million timesteps used to obtain the policy.

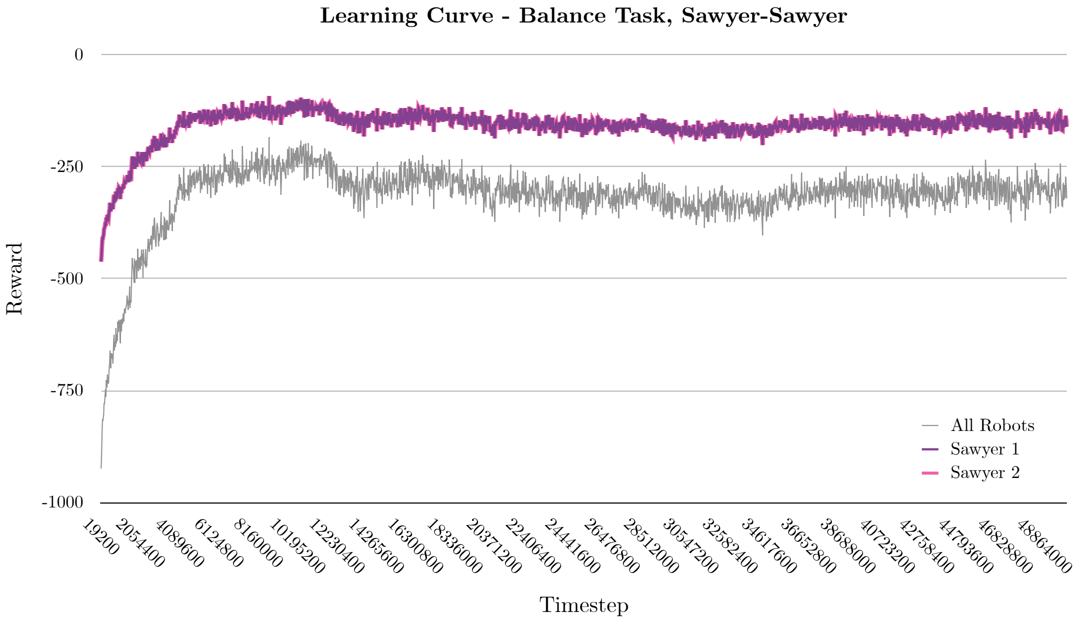


Figure 13: Learning Curve for the Balancing Task with Sawyers using PPO

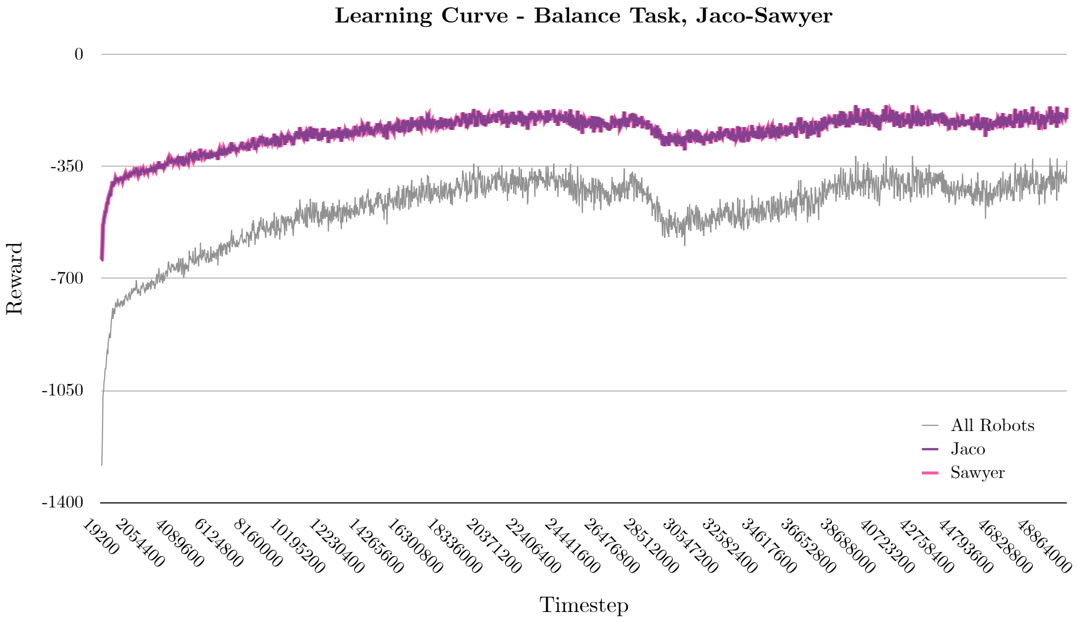


Figure 14: Learning Curve for the Balancing Task with Jaco and Sawyer using PPO

It can be noted from the two graphs above that an effective policy is learned for both combination of robots within the first 1800000 time-steps. Similarly to the poking task, the learning curve follows a logarithmic growth pattern, with gains occurring quickly at the beginning but decreasing over time.

1.4 Pass Task Results

To calculate the reward value used to train policies for the passing task, a reward function $R(r_n)$ for each robot r_n with $n = 1, 2$ has been formulated and is defined as follows:

$$R(r_1) = -dist\_to\_cube(r_1) - dist\_to\_cube(r_2) + moving\_penalty(r_1) \\ + gripping\_incentive(r_1) + task\_completion\_incentive$$

$$R(r_2) = -dist\_to\_cube(r_2) - dist\_cube\_to\_target + moving\_penalty(r_2) \\ + gripping\_incentive(r_2) + task\_completion\_incentive$$

where:

- $dist\_to\_cube(r_n)$: is the euclidean distance between the end effector of robot (r_n) and the cube object.
- $dist\_cube\_to\_target$: is the euclidean distance between the center of the cube object and the target.
- $moving\_penalty(r_n)$: is the norm of the vector of joint actions for each robot r_n. This makes sure that each robot doesn't make unnecessary movements.
- $gripping\_incentive(r_1)$: is a dynamic reward which assumes two different values depending on the state of the robot. This reward is a function of the gripping action g. Firstly, it is defined as follows whenever the $dist\_to\_cube(r_1)$ is smaller than a give threshold: $+0.1 * g$. Secondly, it is defined as $-0.1 * g$ whenever the $dist\_to\_cube(r_2)$ is smaller than a give threshold. This incentives the first robot to pick the cube and then drop it whenever it is close enough for the second robot to catch it.
- $gripping\_incentive(r_2)$: is a positive reward which is a function of the gripping action g, and is defined as follows: $+0.1 * g$. This reward is given to the second robot whenever the $dist\_to\_cube(r_2)$ is smaller than a given threshold.
- $task\_completion\_incentive$: is a positive sparse reward of value +1 given to each robot whenever the cube reaches the target position. It is found that adding a sparse reward improves learning.

Figure 15 and Figure 16 below show the learning curves over the fifty million timesteps used to obtain the policy.

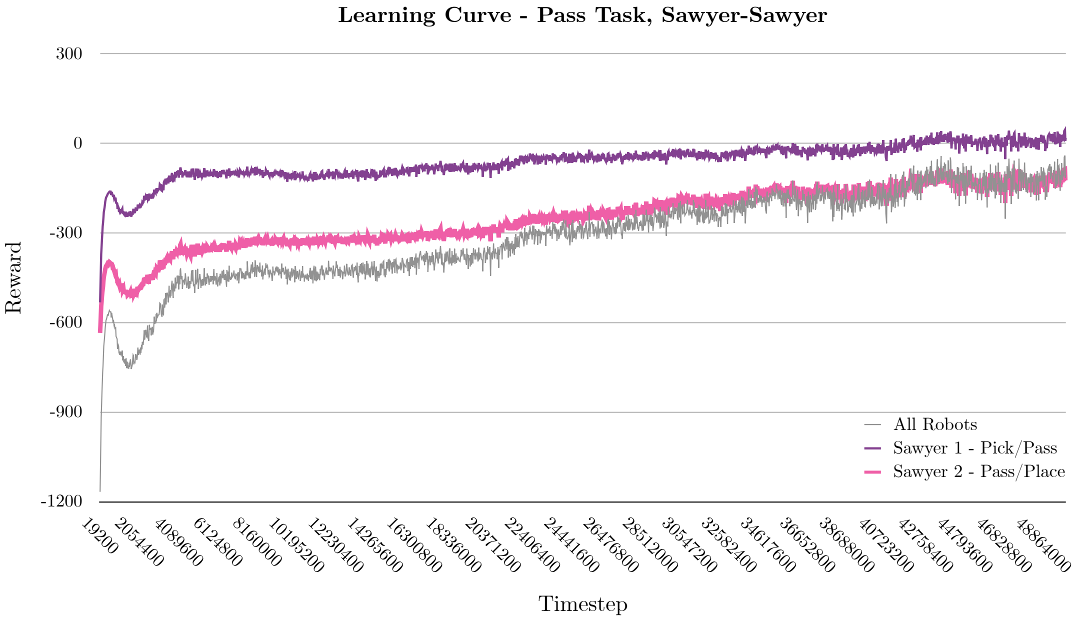


Figure 15: Learning Curve for the Passing Task with Sawyers using PPO

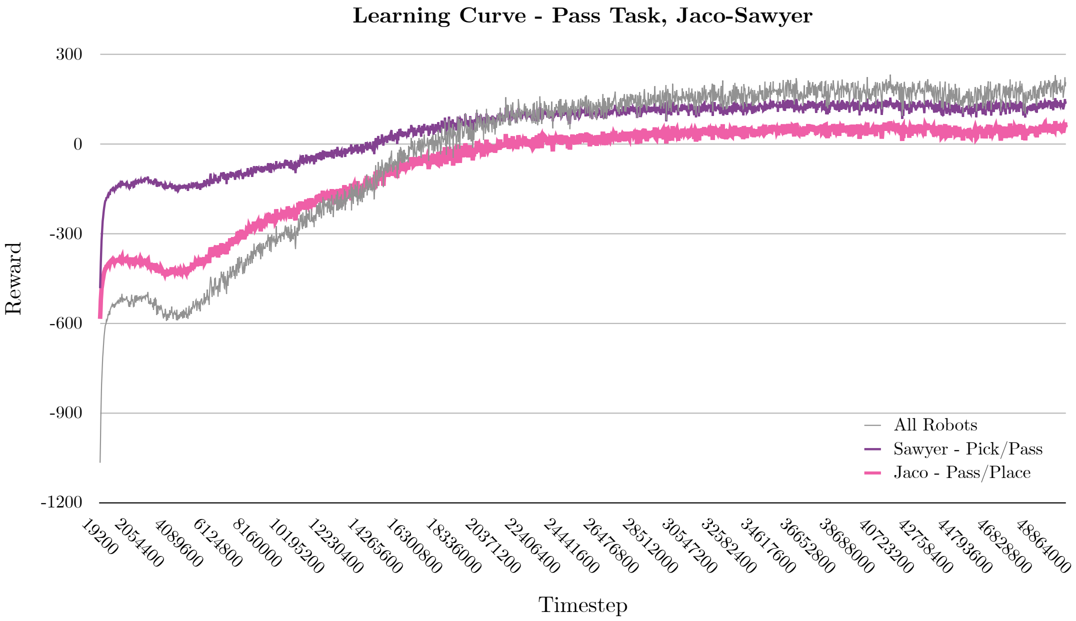


Figure 16: Learning Curve for the Passing Task with Jaco and Sawyer using PPO

As seen from the two learning curves above, an effective policy is learned for both combination of robots within the first 30000000 time-steps.

2 Policy Evaluation Results

Due to the fact that Collaborative Gym uses a variety of robots, it offers the opportunity to study and compare the collaboration between homogeneous and heterogeneous robots in solving various collaborative tasks. A comparison between a variety of tasks with different combinations of robots was conducted by holding all parameters and settings for PPO and the simulation environments constant. It was possible to evaluate the control policies over 200 simulations based on the trained control policies for the specific robots and collaborative task. Based on the 200 simulation rollouts, Table 11 displays the average reward achieved for each task. In addition, Collaborative Gym defines *task completion* and *task performance* for each collaborative task. While *task completion* is defined as the ability to complete the desired task by achieving the desired goal, *task performance* is defined for each collaborative task as follows:

- *Task Performance Poke Task*: The success of the poking task is determined by the ratio between the number of time-steps the stick stays inside the donut and the number of time-steps starting when the stick first enters the donut. As such, this metric does not focus on how long it takes the robots to complete the task, but rather on how well the task has been completed.
- *Task Performance Lift Task*: Similar to the poking task, the success of the lifting task is determined by the ratio of the number of time steps the pot stays within a certain distance to the target position and the number of time steps starting when the pot first reaches the target position. Once again, this metric does not take into consideration how long it takes the robots to complete the task, but instead focuses on the quality at which the task is successfully completed.
- *Task Performance Balance Task*: The success of the balancing task is determined by the ratio between the number of time-steps the ball stays on the balancing board and the total number of time-steps of an episode.
- *Task Performance Pass Task*: Similar to the lifting task, the success of the passing task is determined by the ratio of the number of time steps the cube object stays within a certain distance to the target position and the number of time steps starting when the cube first reaches the target position. Again, the metric does not account for the amount of time required for the robots to complete the task, but instead is concerned with the quality at which the task is successfully completed.

Task Name		**Robot 1**	**Robot 2**	**Mean Reward**	**Task Completion**	**Task Performance**
Poke Task	*Variant 1*	Sawyer	Sawyer	254.23	98.5%	98.49%
	Variant 2	Panda	Sawyer	215.94	100%	99.98%
Lift Task	*Variant 1*	Sawyer	Sawyer	-356.88	98%	29.01%
	Variant 2	Jaco	Jaco	-144.53	98.5%	69.40%
Balance Task	*Variant 1*	Sawyer	Sawyer	-341.90	92%	72.00%
	Variant 2	Jaco	Sawyer	-404.53	73.50%	64.97%
Pass Task	*Variant 1*	Sawyer	Sawyer	-95.22	72.30%	59.93%
	Variant 2	Jaco	Sawyer	198.16	96.00%	95.46%

Table 11: Average Reward and Success Rate on 200 Trials

As can be seen in Table 11, *task completion* is close to 100% for most of the trained tasks. There is no doubt that *task completion* is lower for the second variant of the balancing task as well as the first variant of the passing task. It is possible that a lower completion rate is due to both the robots used as well as the difficulty of the task. Indeed, it could be argued that a specific combination of robots might perform better for a certain task than others. As a matter of fact, a relatively high *task completion* rate indicates that the robots were able to learn an effective policy to complete the designed tasks. It appears, however, that *task performance* varies significantly between tasks. As before, the difficulty of the task has a direct relationship with this. In fact, poking is considered to be a trivial task for robots, as a result of the high performance rate. However, when it comes to tasks such as lifting, balancing, and passing, success rates decrease because of task complexity. Interestingly, while most variants of tasks appear to have similar task performance rates, the lift task does not. According to the results, the combination of Jaco robots outperforms the Sawyers by almost 50%. A similar scenario is present for the passing task, whereas the Jaco-Sawyer combination outperforms the Sawyer-Sawyer one. As a result, it is evident that while all robots are capable of achieving the predetermined goal, the performance could be strongly dependent on the type of robot and the combination of robots available for the particular task.

Conclusion & Future Research

As part of this research, a Multi-Agent Robotic System (MARS) environment was developed where robotic arms are capable of coordinating and collaborating to meet a common goal across a wide range of tasks. In this paper, a framework for multi-robot interaction has been presented, called Collaborative Gym, which is an open-source, physics-based simulation framework. An overview of the action and observation spaces has been provided. Particularly, assumptions related to the grasping of objects have been addressed, which simplifies the learning complexity for the tasks that have been designed. In terms of the learning process, PPO was utilized as the base algorithm for training robots. Results indicate that robots can learn effective collaborative control policies. The purpose of Collaborative Gym is to encourage robots to interact cooperatively in a variety of different tasks. A significant difference between Collaborative Gym and existing robotic simulation environments is that it emphasizes the modeling of the interaction between multiple robots. In addition, Collaborative Gym has been shown to be a valuable tool for benchmarking and developing multiple collaborative environments. The results of the study show that Collaborative Gym is a promising open-source framework for the development of collaborative robots that are capable of solving complex tasks.

Ultimately, this study can be extended to explore the performance of heterogeneous and homogeneous robots in various collaborative tasks. In particular, it would be interesting to assess the learning rate and performance of different robots in different scenarios. It would also be beneficial to examine aspects of simulation-to-reality. As a matter of fact, implementing and transferring learned control policies to a real-world task could provide interesting results. A further consideration would be the design of different collaborative tasks pertaining to industrial applications such as order picking, manufacturing, and logistics. Moreover, it would be interesting to examine which alternative reinforcement learning algorithms are most suitable for multi robotic collaboration. Furthermore, a possible extension of this work could include aspects of human teaming by incorporating simulated humans into the simulation. Using computer vision, real motion data from humans could be used in Collaborative Gym to explore aspects of human-robot collaboration.

Bibliography

Asada, Minoru, Eiji Uchibe, and Koh Hosoda (1999). “Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development”. In: *Artificial Intelligence* 110.2, pp. 275–292.

Baker, Bowen et al. (2019). “Emergent tool use from multi-agent autocurricula”. In: *arXiv preprint arXiv:1909.07528*.

Balch, Tucker and Ronald C Arkin (1998). “Behavior-based formation control for multi-robot teams”. In: *IEEE transactions on robotics and automation* 14.6, pp. 926–939.

Bansal, Trapit et al. (2017). “Emergent complexity via multi-agent competition”. In: *arXiv preprint arXiv:1710.03748*.

Booth, Joe and Jackson Booth (2019). “Marathon environments: Multi-agent continuous control benchmarks in a modern video game engine”. In: *arXiv preprint arXiv:1902.09097*.

Bousmalis, Konstantinos et al. (2018). “Using simulation and domain adaptation to improve efficiency of deep robotic grasping”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 4243–4250.

Bowling, Michael and Manuela Veloso (2003). “Simultaneous adversarial multi-robot learning”. In: *IJCAI*. Vol. 3, pp. 699–704.

Brandão, Bruno et al. (2022). “Multi-Agent Reinforcement Learning for Strategic Decision Making and Control in Robotic Soccer through Self-Play”. In: *IEEE Access*.

Brockman, Greg et al. (2016). “Openai gym”. In: *arXiv preprint arXiv:1606.01540*.

Buşoniu, Lucian, Robert Babuška, and Bart De Schutter (2010). “Multi-agent reinforcement learning: An overview”. In: *Innovations in multi-agent systems and applications-1*, pp. 183–221.

Candea, Ciprian et al. (2001). “Coordination in multi-agent RoboCup teams”. In: *Robotics and Autonomous Systems* 36.2-3, pp. 67–86.

Cao, Y Uny, Andrew B Kahng, and Alex S Fukunaga (1997). “Cooperative mobile robotics: Antecedents and directions”. In: *Robot colonies*. Springer, pp. 7–27.

Chu, Tianshu et al. (2019). “Multi-agent deep reinforcement learning for large-scale traffic signal control”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.3, pp. 1086–1095.

Costanzo, Marco, Giuseppe De Maria, and Ciro Natale (2021). “Handover control for human-robot and robot-robot collaboration”. In: *Frontiers in Robotics and AI* 8, p. 132.

Coumans, Erwin and Yunfei Bai (2016). “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In.

Cui, Jingjing, Yuanwei Liu, and Arumugam Nallanathan (2019). “Multi-agent reinforcement learning-based resource allocation for UAV networks”. In: *IEEE Transactions on Wireless Communications* 19.2, pp. 729–743.

DartEnv (2016). https://github.com/DartEnv/dart-env.

Ding, Guohui et al. (2020). “Distributed reinforcement learning for cooperative multi-robot object manipulation”. In: *arXiv preprint arXiv:2003.09540*.

Du, Wei and Shifei Ding (2021). "A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications". In: *Artificial Intelligence Review* 54.5, pp. 3215–3238.

Elhajj, Imad H et al. (2003). "Design and analysis of internet-based tele-coordinated multi-robot systems". In: *Autonomous Robots* 15.3, pp. 237–254.

Erickson, Zackory et al. (2020). "Assistive gym: A physics simulation framework for assistive robotics". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 10169–10176.

Fan, Tingxiang et al. (2020). "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios". In: *The International Journal of Robotics Research* 39.7, pp. 856–892.

Fernandez, Fernando and Lynne E Parker (2001). "Learning in large cooperative multi-robot domains". In.

GymDart (2018). `https://github.com/dartsim/gym-dart`.

Hu, Junyan et al. (2020). "Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning". In: *IEEE Transactions on Vehicular Technology* 69.12, pp. 14413–14423.

Iocchi, Luca et al. (2003). "Distributed coordination in heterogeneous multi-robot systems". In: *Autonomous robots* 15.2, pp. 155–168.

Iqbal, Shariq and Fei Sha (2019). "Actor-attention-critic for multi-agent reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 2961–2970.

Ismail, Zool Hilmi, Nohaidda Sariff, and EG Hurtado (2018). "A survey and analysis of cooperative multi-agent robot systems: challenges and directions". In: *Applications of Mobile Robots*. IntechOpen, pp. 8–14.

Kar, Soummya, José M. F. Moura, and H. Vincent Poor (2013). "$\mathcal{QD}$-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through Consensus+Innovations". In: *IEEE Transactions on Signal Processing* 61.7, pp. 1848–1862. DOI: `10.1109/TSP.2013.2241057`.

Khan, Arbaaz et al. (2018). "Scalable centralized deep multi-agent reinforcement learning via policy gradients". In: *arXiv preprint arXiv:1805.08776*.

Lauer, Martin and Martin Riedmiller (2000). "An algorithm for distributed reinforcement learning in cooperative multi-agent systems". In: *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer.

Lee, Jeongseok et al. (2018). "Dart: Dynamic animation and robotics toolkit". In: *Journal of Open Source Software* 3.22, p. 500.

Liang, Eric et al. (2018). "RLlib: Abstractions for distributed reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 3053–3062.

Liu, Jiming and Jianbing Wu (2018). *Multiagent robotic systems*. CRC press.

Lyu, Xueguang et al. (2021). "Contrasting centralized and decentralized critics in multi-agent reinforcement learning". In: *arXiv preprint arXiv:2102.04402*.

Matarić, Maja J (1997). "Reinforcement learning in the multi-robot domain". In: *Robot colonies*. Springer, pp. 73–83.

Matarić, Maja J (2001). "Learning in behavior-based multi-robot systems: Policies, models, and other agents". In: *Cognitive Systems Research* 2.1, pp. 81–93.

Matarić, Maja J, Gaurav S Sukhatme, and Esben H Østergaard (2003). "Multi-robot task allocation in uncertain environments". In: *Autonomous Robots* 14.2, pp. 255–263.

Michaud, François and Maja J Matarić (1998). "Learning from history for behavior-based mobile robots in non-stationary conditions". In: *Machine Learning* 31.1, pp. 141–167.

Panait, Liviu and Sean Luke (2005). "Cooperative multi-agent learning: The state of the art". In: *Autonomous agents and multi-agent systems* 11.3, pp. 387–434.

Pawar, Pranav M and Amir Leshem (2021). "Distributed Deep Reinforcement Learning for Collaborative Spectrum Sharing". In: *arXiv preprint arXiv:2104.02059*.

PyBullet Community (2013). `https://github.com/bulletphysics/bullet3/issues`.

RAY (2022). *RAY RLlib Algorithms*. `https://docs.ray.io/en/latest/rllib/rllib-algorithms.html#ppo`.

Rosa, Lorenzo et al. (2015). "Multi-task cooperative control in a heterogeneous ground-air robot team". In: *IFAC-PapersOnLine* 48.5, pp. 53–58.

Sadeghi, Fereshteh et al. (2017). "Sim2real view invariant visual servoing by recurrent control". In: *arXiv preprint arXiv:1712.07642*.

Sharma, Piyush K et al. (2021). "Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*. Vol. 11746. SPIE, pp. 665–676.

Al-Shedivat, Maruan et al. (2017). "Continuous adaptation via meta-learning in nonstationary and competitive environments". In: *arXiv preprint arXiv:1710.03641*.

Suarez, Joseph et al. (2019). "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents". In: *arXiv preprint arXiv:1903.00784*.

Tan, Jie et al. (2018). "Sim-to-real: Learning agile locomotion for quadruped robots". In: *arXiv preprint arXiv:1804.10332*.

Terry, J et al. (2021). "Pettingzoo: Gym for multi-agent reinforcement learning". In: *Advances in Neural Information Processing Systems* 34, pp. 15032–15043.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 5026–5033.

Touzet, Claude F (2000). "Robot awareness in cooperative mobile robot learning". In: *Autonomous Robots* 8.1, pp. 87–97.

— (2004). "Distributed lazy Q-learning for cooperative mobile robots". In: *International Journal of Advanced Robotic Systems* 1.1, p. 1.

Usunier, Nicolas et al. (2016). "Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks". In: *arXiv preprint arXiv:1609.02993*.

Van Der Zwaan, Sjoerd, JosE AA Moreira, and Pedro U Lima (2000). "Cooperative learning and planning for multiple robots". In: *Proceedings of the 2000 IEEE International Symposium on Intelligent Control. Held jointly with the 8th IEEE Mediterranean Conference on Control and Automation (Cat. No. 00CH37147)*. IEEE, pp. 351–356.

Vinyals, Oriol, Igor Babuschkin, et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782, pp. 350–354.

Vinyals, Oriol, Timo Ewalds, et al. (2017). "Starcraft ii: A new challenge for reinforcement learning". In: *arXiv preprint arXiv:1708.04782*.

Wang, Baoqian, Junfei Xie, and Nikolay Atanasov (2021). "Coding for Distributed Multi-Agent Reinforcement Learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 10625–10631.

— (2022). "DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors". In: *arXiv preprint arXiv:2202.09019*.

Weiß, Gerhard (1995). "Distributed reinforcement learning". In: *The Biology and technology of intelligent autonomous agents*. Springer, pp. 415–428.

Wiering, Marco, Rafał Sałustowicz, and Jürgen Schmidhuber (1999). "Reinforcement learning soccer teams with incomplete world models". In: *Autonomous Robots* 7.1, pp. 77–88.

Yang, Erfu and Dongbing Gu (2004). *Multiagent reinforcement learning for multi-robot systems: A survey.* Tech. rep. tech. rep.

— (2005). "A Survey on Multiagent Reinforcement Learning Towards Multi-Robot Systems." In: *CIG*. Citeseer.

Yang, Yang, Li Juntao, and Peng Lingling (2020). "Multi-robot path planning based on a deep reinforcement learning DQN algorithm". In: *CAAI Transactions on Intelligence Technology* 5.3, pp. 177–183.

Yu, Chao, Yinzhao Dong, et al. (2020). "Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit". In: *The Journal of Engineering* 2020.13, pp. 499–504.

Yu, Chao, Akash Velu, et al. (2021). "The surprising effectiveness of ppo in cooperative, multi-agent games". In: *arXiv preprint arXiv:2103.01955*.

Zeng, Andy et al. (2020). "Tossingbot: Learning to throw arbitrary objects with residual physics". In: *IEEE Transactions on Robotics* 36.4, pp. 1307–1319.

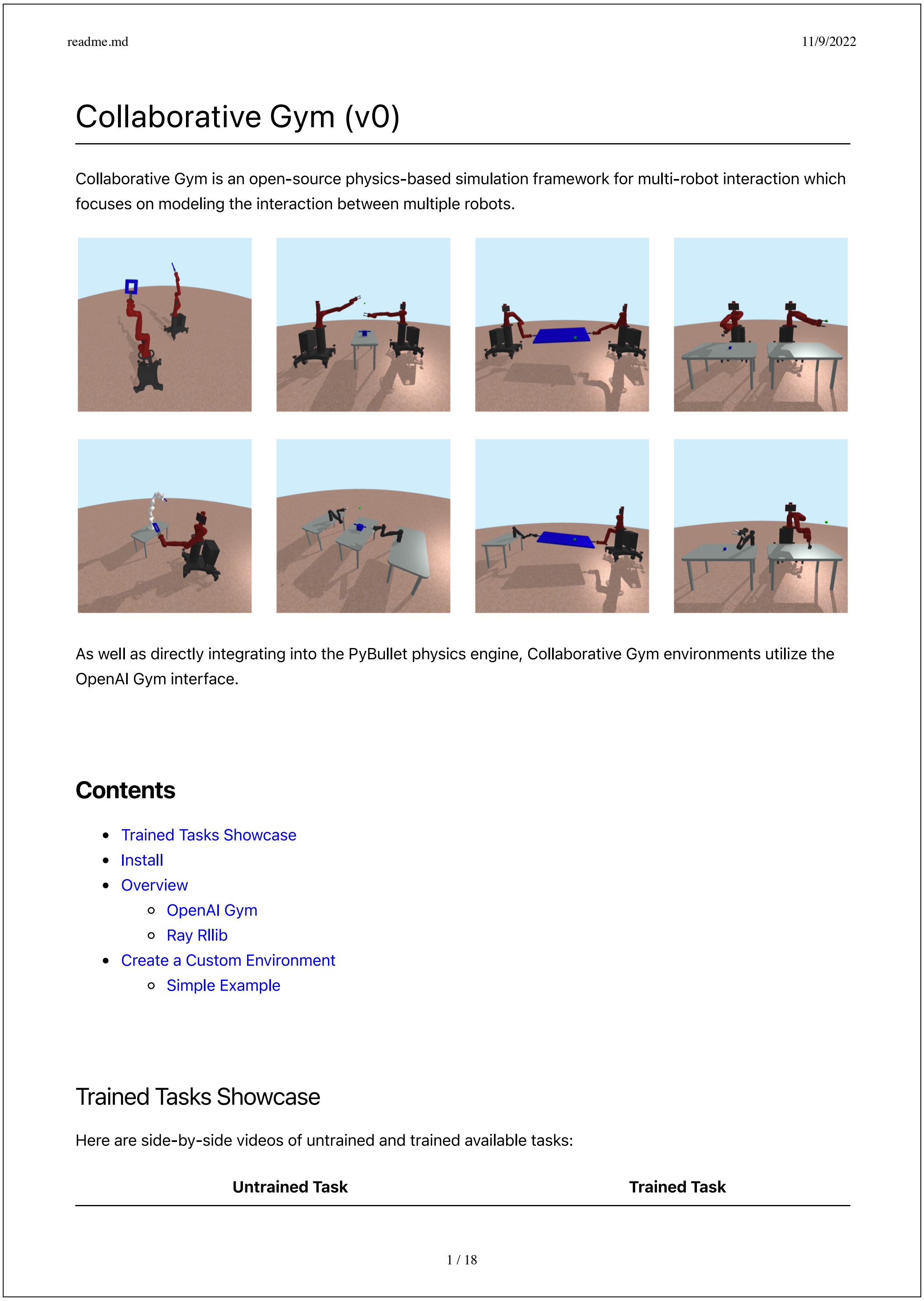
Zhang, Kaiqing, Zhuoran Yang, and Tamer Başar (2021). "Multi-agent reinforcement learning: A selective overview of theories and algorithms". In: *Handbook of Reinforcement Learning and Control*, pp. 321–384.

Zhang, Lin et al. (2020). "Decentralized control of multi-robot system in cooperative object transportation using deep reinforcement learning". In: *IEEE Access* 8, pp. 184109–184119.

Zhu, Yuke et al. (2020). "robosuite: A modular simulation framework and benchmark for robot learning". In: *arXiv preprint arXiv:2009.12293*.

Appendices

A GitHub Wiki



Collaborative Gym (v0)

Collaborative Gym is an open-source physics-based simulation framework for multi-robot interaction which focuses on modeling the interaction between multiple robots.

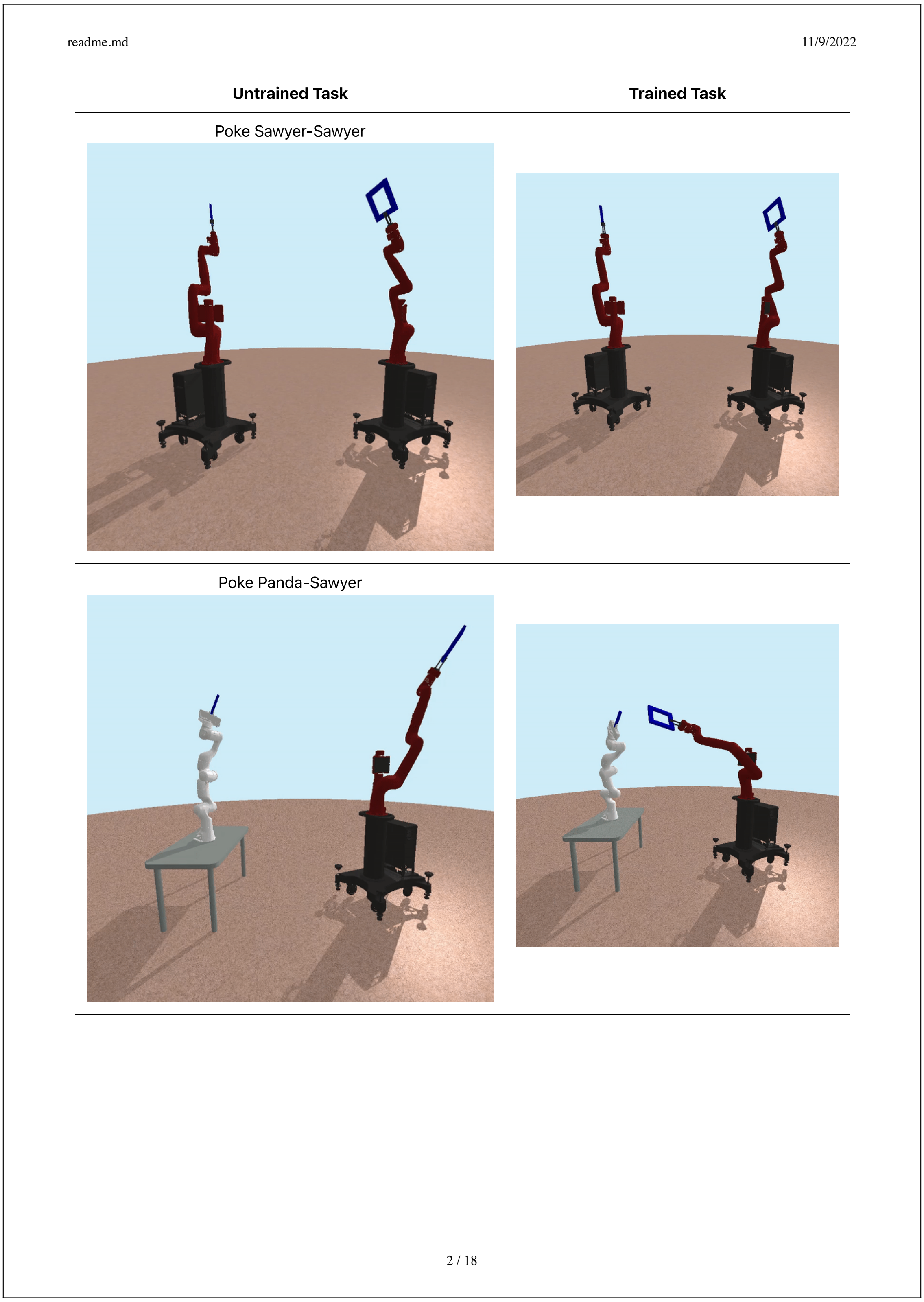
As well as directly integrating into the PyBullet physics engine, Collaborative Gym environments utilize the OpenAI Gym interface.

Contents

Trained Tasks Showcase

Here are side-by-side videos of untrained and trained available tasks:

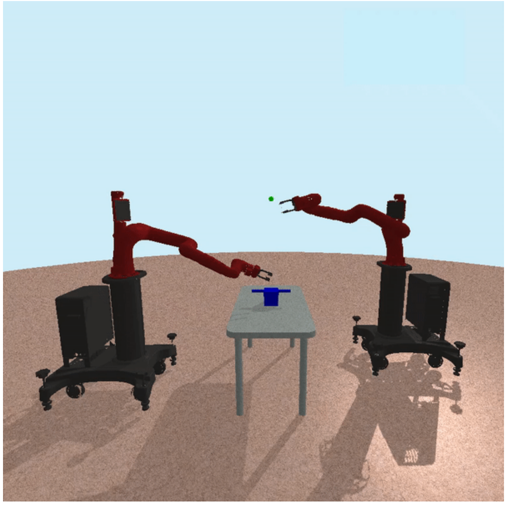
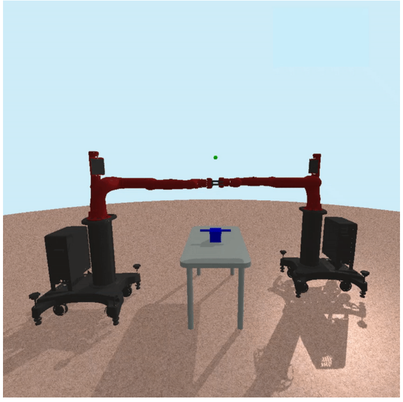
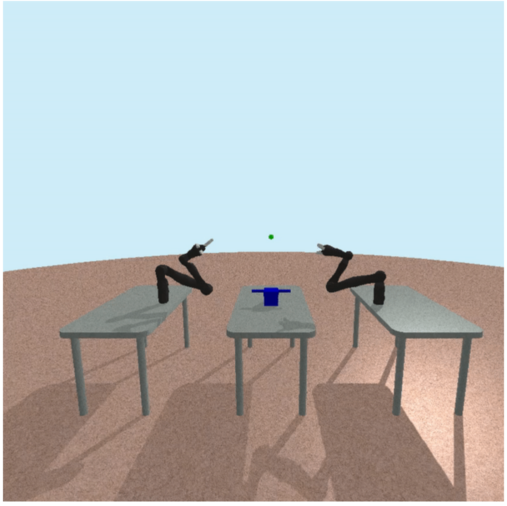
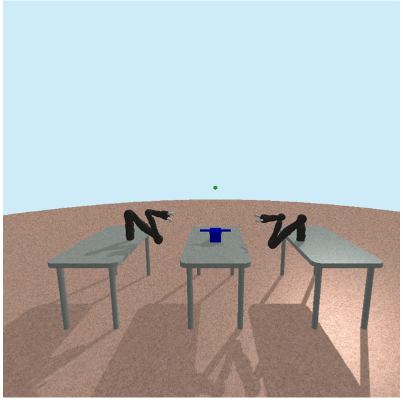
Untrained Task	Trained Task

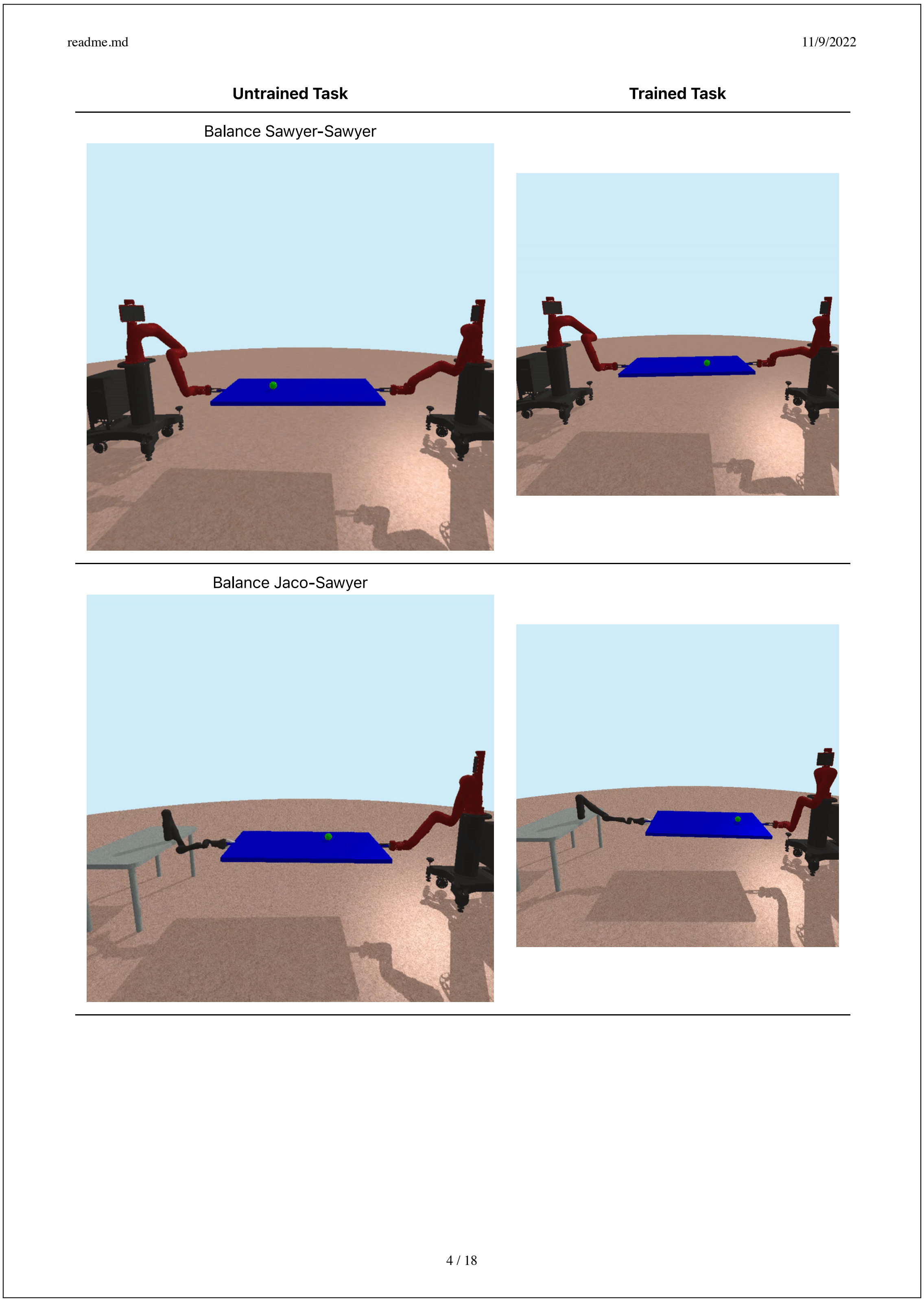


readme.md 11/9/2022

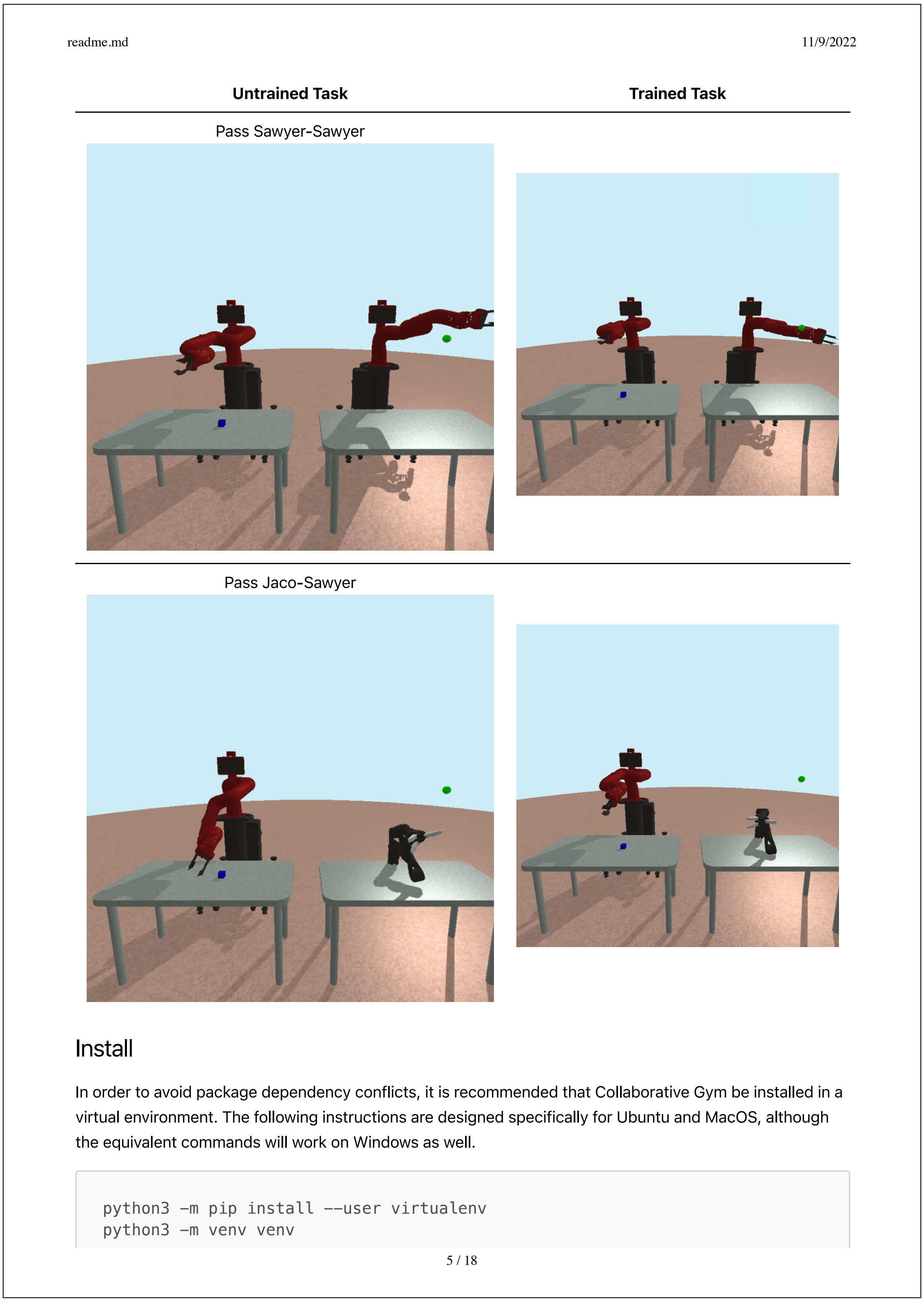
Untrained Task	Trained Task
Poke Sawyer-Sawyer	
Poke Panda-Sawyer	

2 / 18

Untrained Task	Trained Task
Lift Sawyer-Sawyer 	
Lift Jaco-Jaco 	



Untrained Task	Trained Task
Balance Sawyer-Sawyer	
Balance Jaco-Sawyer	



readme.md 11/9/2022

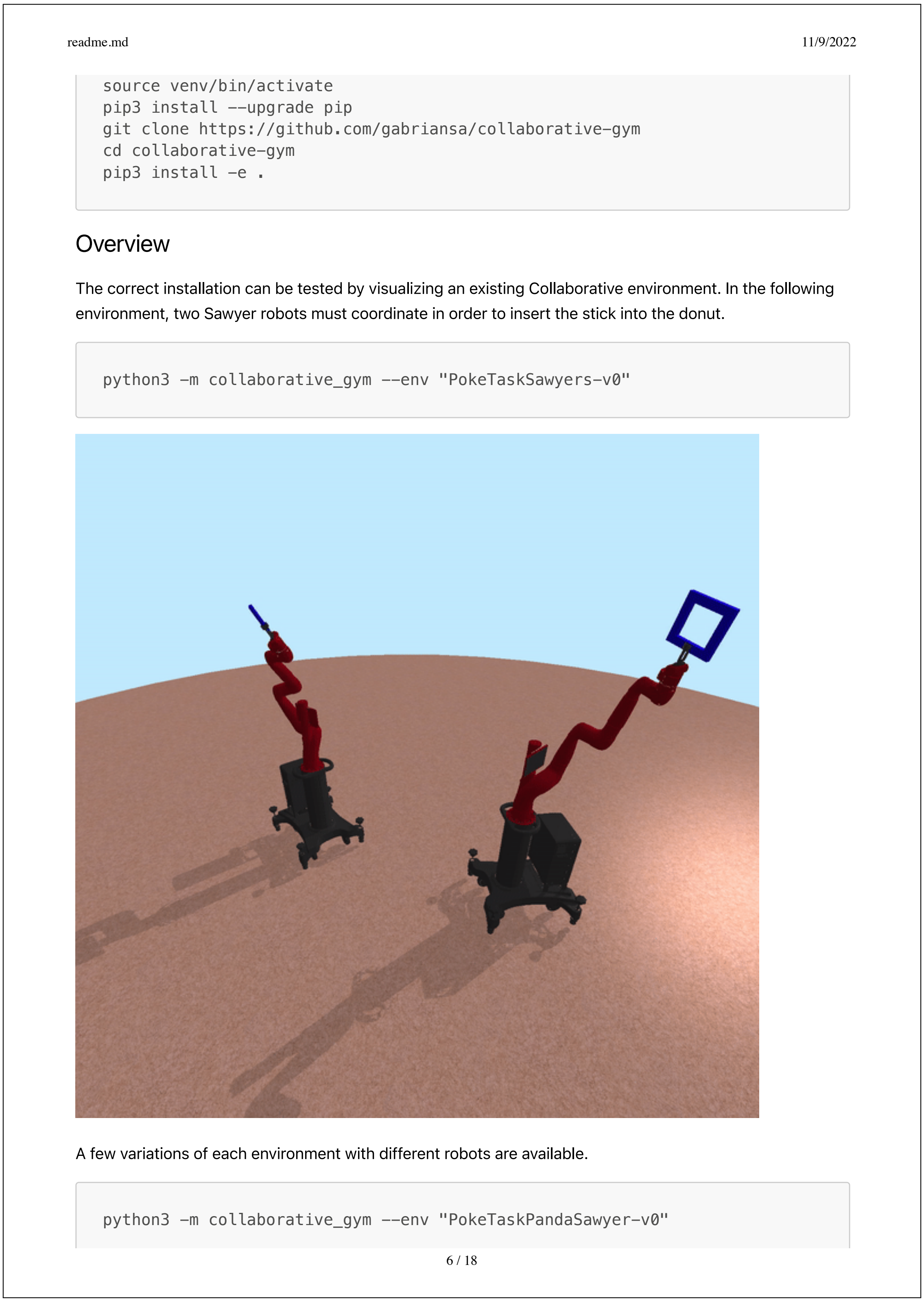
Untrained Task	Trained Task
Pass Sawyer-Sawyer	
Pass Jaco-Sawyer	

Install

In order to avoid package dependency conflicts, it is recommended that Collaborative Gym be installed in a virtual environment. The following instructions are designed specifically for Ubuntu and MacOS, although the equivalent commands will work on Windows as well.

```
python3 -m pip install --user virtualenv
python3 -m venv venv
```

5 / 18



```
source venv/bin/activate
pip3 install --upgrade pip
git clone https://github.com/gabriansa/collaborative-gym
cd collaborative-gym
pip3 install -e .
```

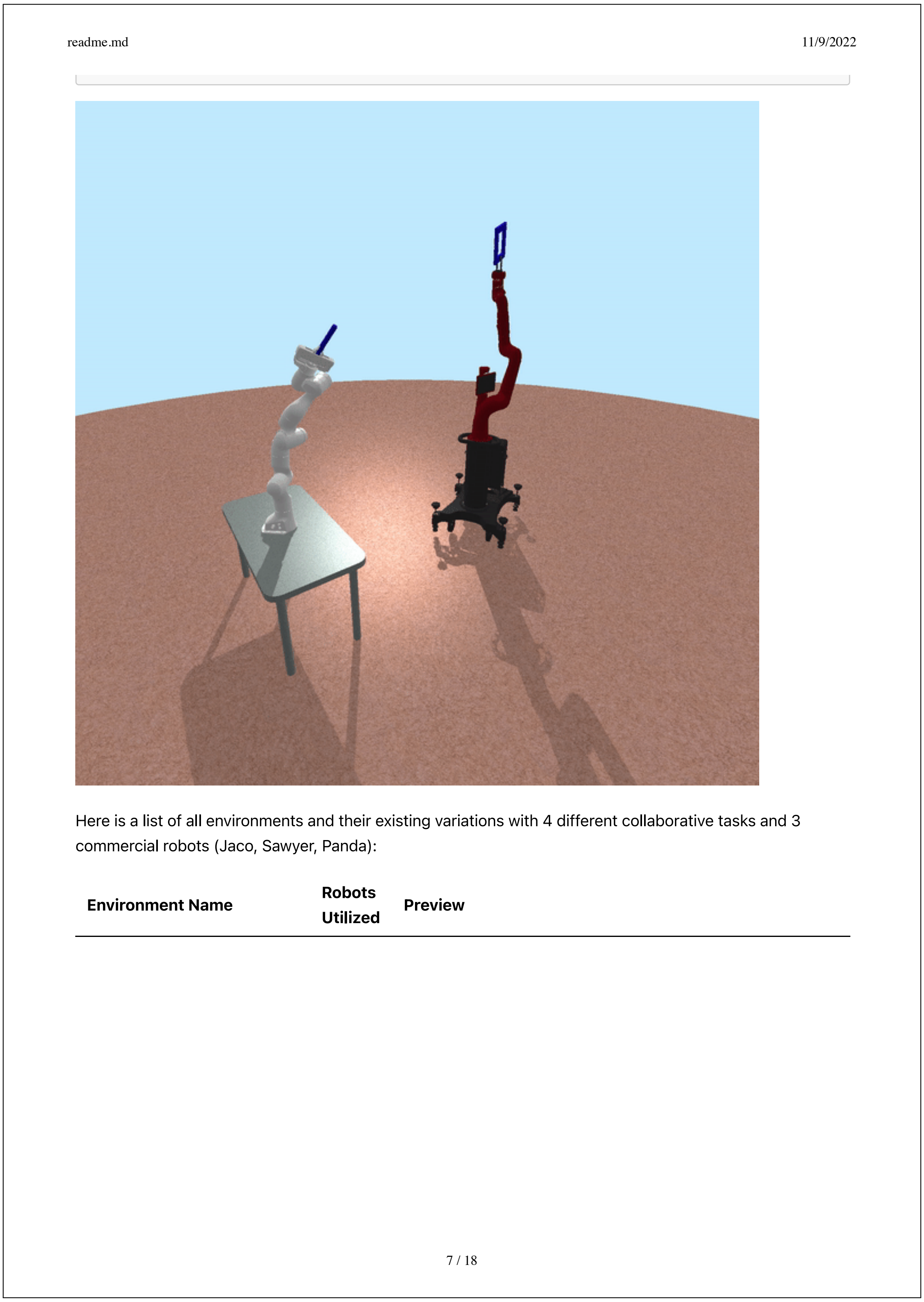
Overview

The correct installation can be tested by visualizing an existing Collaborative environment. In the following environment, two Sawyer robots must coordinate in order to insert the stick into the donut.

```
python3 -m collaborative_gym --env "PokeTaskSawyers-v0"
```

A few variations of each environment with different robots are available.

```
python3 -m collaborative_gym --env "PokeTaskPandaSawyer-v0"
```

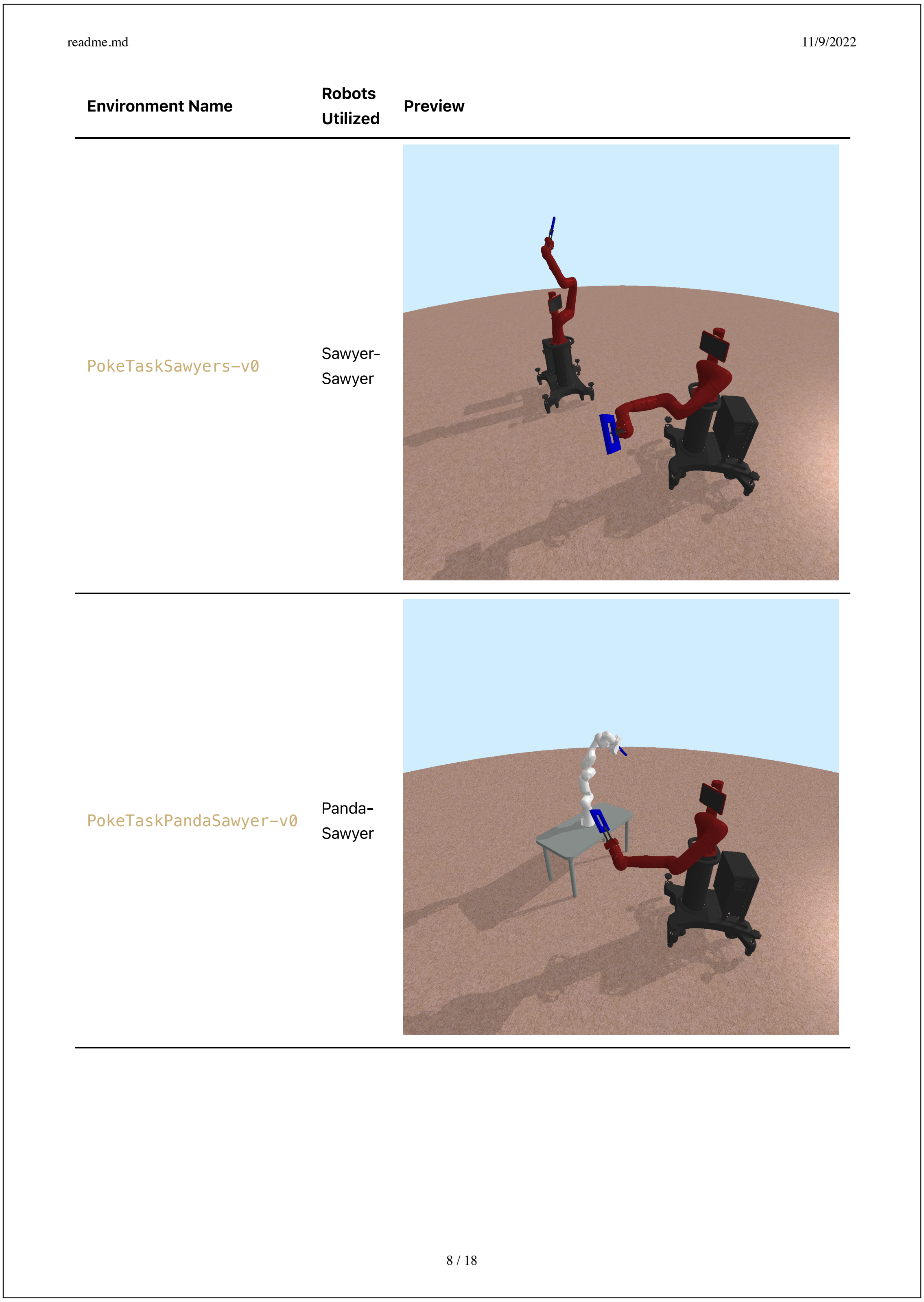


readme.md 11/9/2022

Here is a list of all environments and their existing variations with 4 different collaborative tasks and 3 commercial robots (Jaco, Sawyer, Panda):

Environment Name	Robots Utilized	Preview

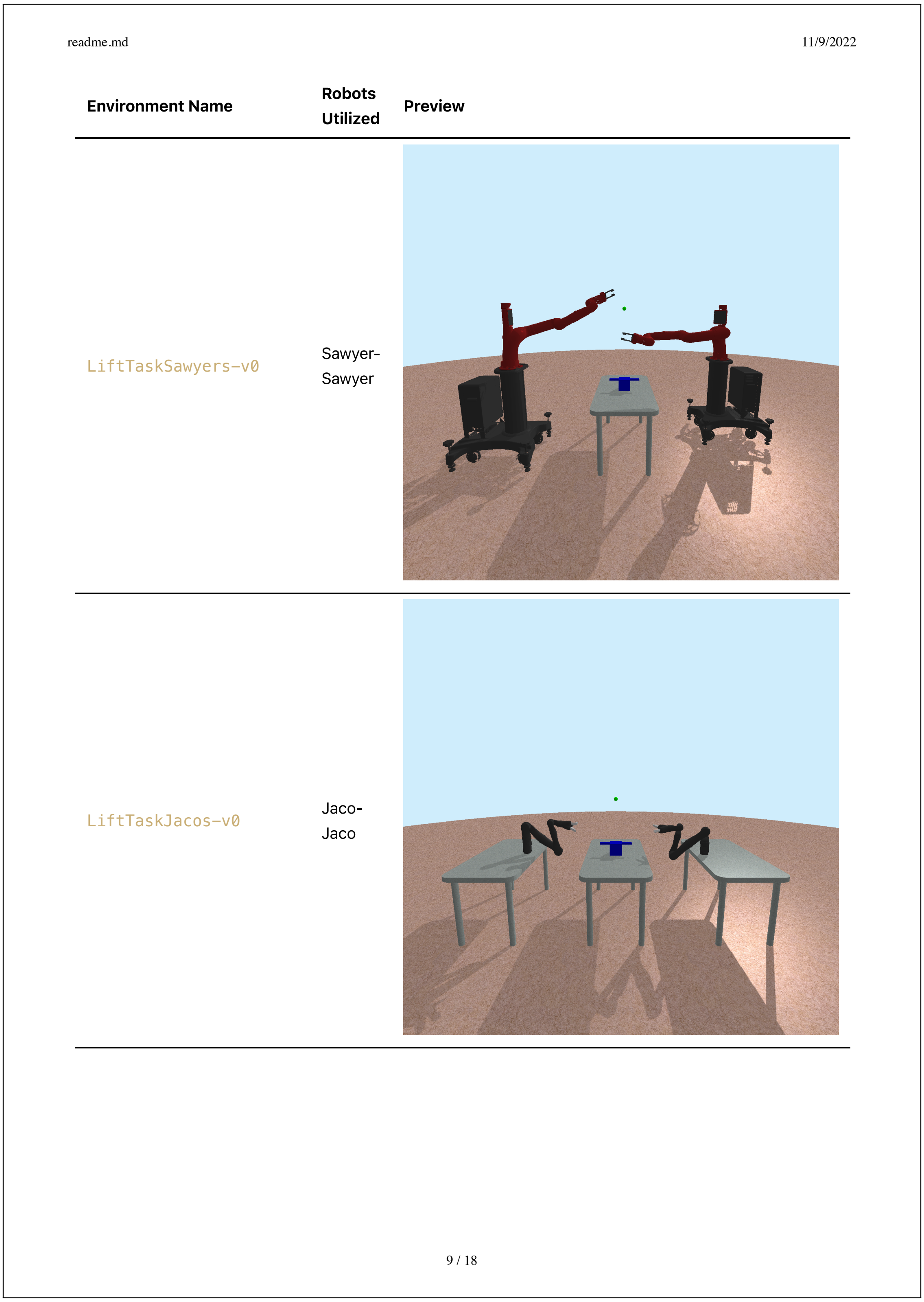
7 / 18



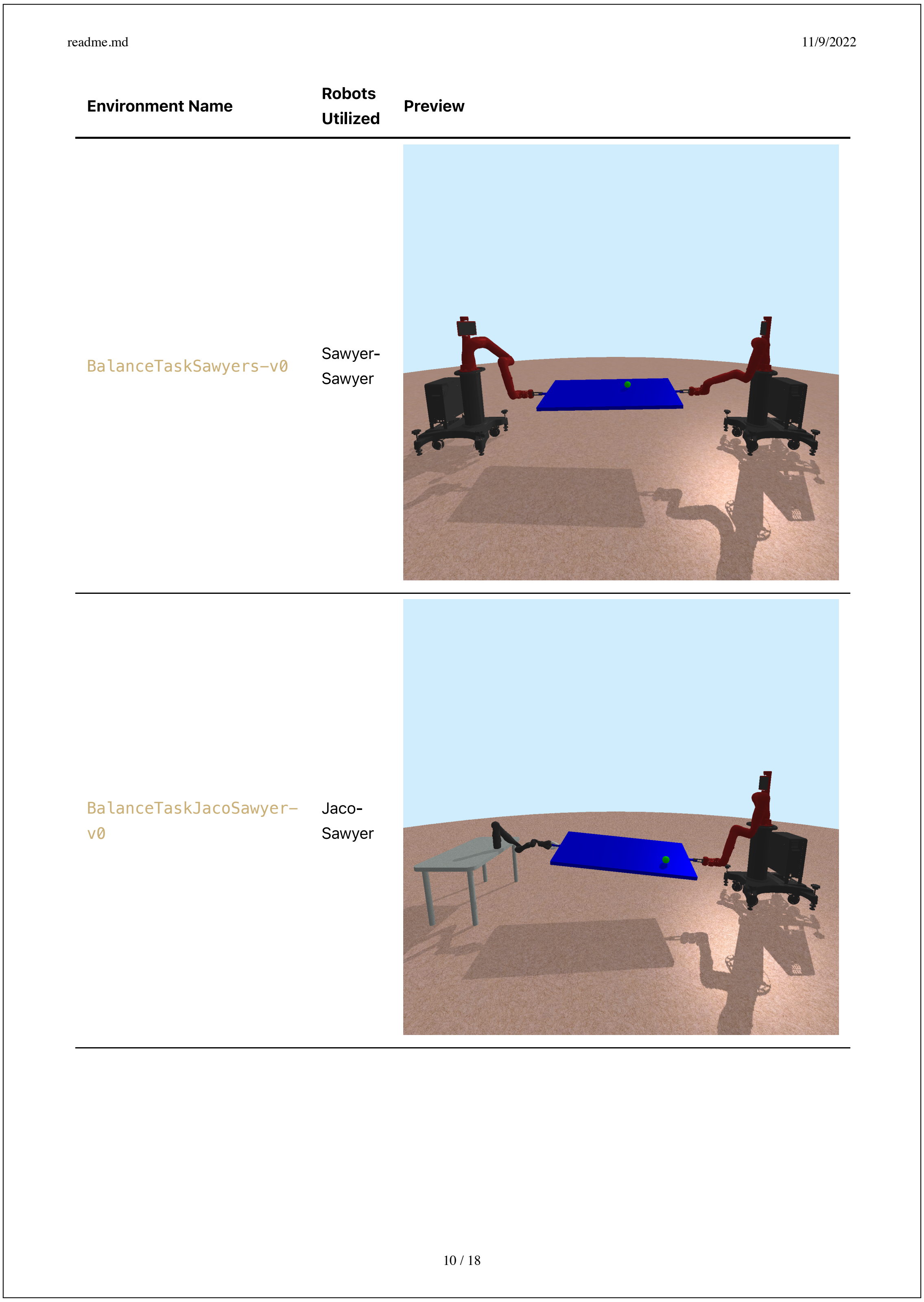
readme.md 11/9/2022

Environment Name	Robots Utilized	Preview
`PokeTaskSawyers-v0`	Sawyer-Sawyer	
`PokeTaskPandaSawyer-v0`	Panda-Sawyer	

8 / 18



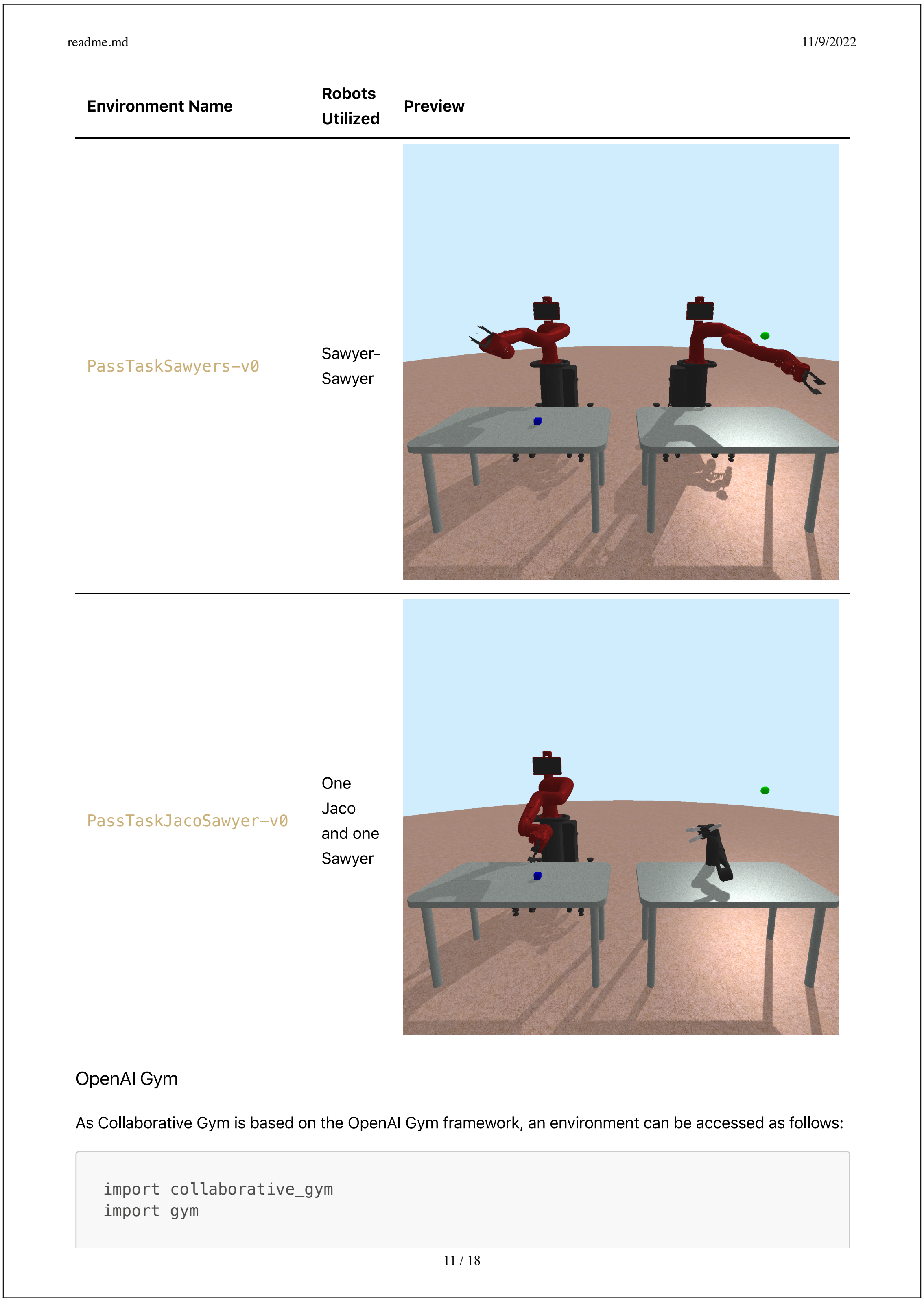
Environment Name	Robots Utilized	Preview
`LiftTaskSawyers-v0`	Sawyer-Sawyer	
`LiftTaskJacos-v0`	Jaco-Jaco	



readme.md 11/9/2022

Environment Name	Robots Utilized	Preview
`BalanceTaskSawyers-v0`	Sawyer-Sawyer	
`BalanceTaskJacoSawyer-v0`	Jaco-Sawyer	

10 / 18

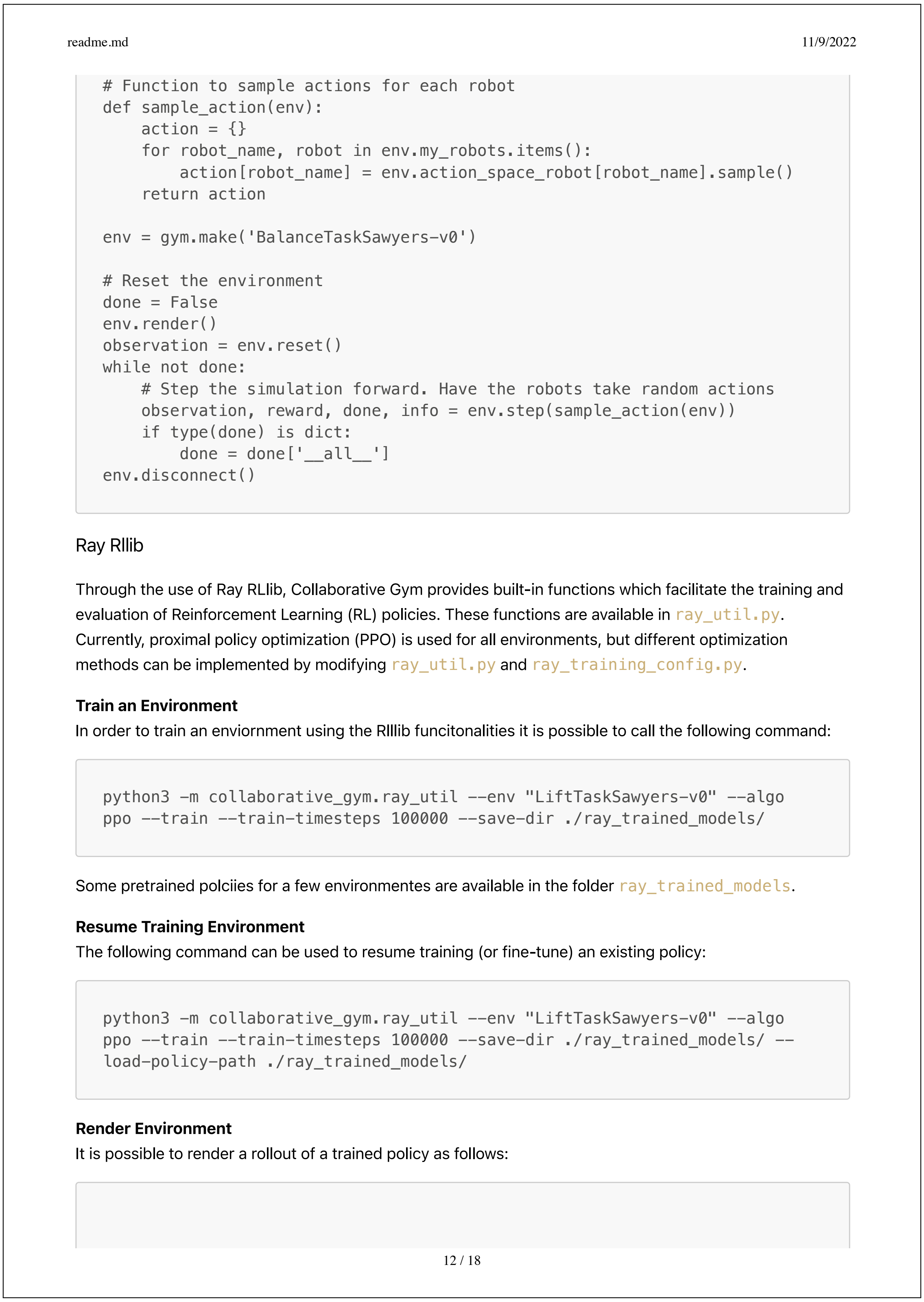


Environment Name	Robots Utilized	Preview
PassTaskSawyers-v0	Sawyer-Sawyer	
PassTaskJacoSawyer-v0	One Jaco and one Sawyer	

OpenAI Gym

As Collaborative Gym is based on the OpenAI Gym framework, an environment can be accessed as follows:

```
import collaborative_gym
import gym
```



```
# Function to sample actions for each robot
def sample_action(env):
    action = {}
    for robot_name, robot in env.my_robots.items():
        action[robot_name] = env.action_space_robot[robot_name].sample()
    return action

env = gym.make('BalanceTaskSawyers-v0')

# Reset the environment
done = False
env.render()
observation = env.reset()
while not done:
    # Step the simulation forward. Have the robots take random actions
    observation, reward, done, info = env.step(sample_action(env))
    if type(done) is dict:
        done = done['__all__']
env.disconnect()
```

Ray Rllib

Through the use of Ray RLlib, Collaborative Gym provides built-in functions which facilitate the training and evaluation of Reinforcement Learning (RL) policies. These functions are available in `ray_util.py`. Currently, proximal policy optimization (PPO) is used for all environments, but different optimization methods can be implemented by modifying `ray_util.py` and `ray_training_config.py`.

Train an Environment
In order to train an enviornment using the Rlllib funcitonalities it is possible to call the following command:

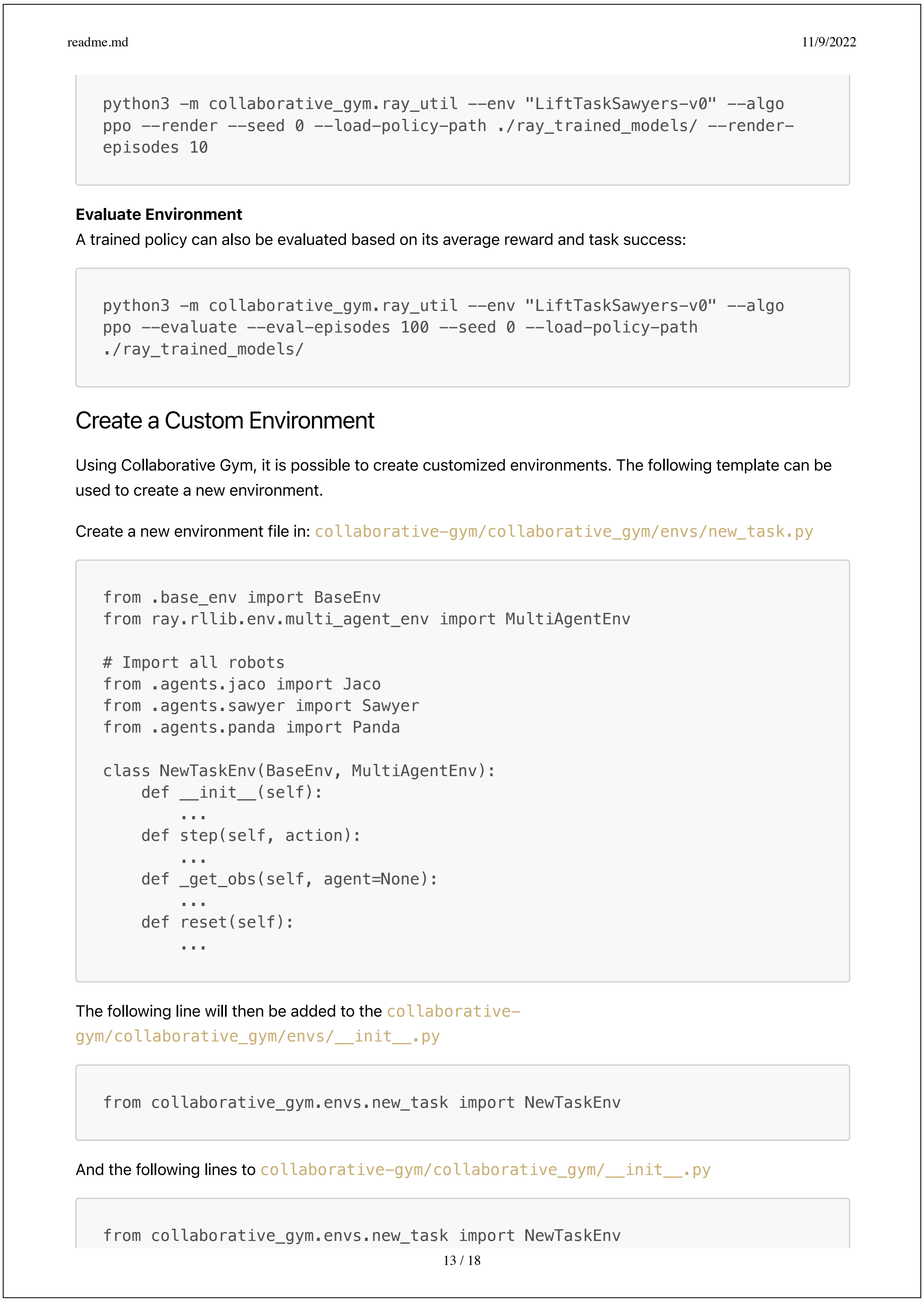
```
python3 -m collaborative_gym.ray_util --env "LiftTaskSawyers-v0" --algo
ppo --train --train-timesteps 100000 --save-dir ./ray_trained_models/
```

Some pretrained polciies for a few environmentes are available in the folder `ray_trained_models`.

Resume Training Environment
The following command can be used to resume training (or fine-tune) an existing policy:

```
python3 -m collaborative_gym.ray_util --env "LiftTaskSawyers-v0" --algo
ppo --train --train-timesteps 100000 --save-dir ./ray_trained_models/ --
load-policy-path ./ray_trained_models/
```

Render Environment
It is possible to render a rollout of a trained policy as follows:



readme.md 11/9/2022

```
python3 -m collaborative_gym.ray_util --env "LiftTaskSawyers-v0" --algo
ppo --render --seed 0 --load-policy-path ./ray_trained_models/ --render-
episodes 10
```

Evaluate Environment

A trained policy can also be evaluated based on its average reward and task success:

```
python3 -m collaborative_gym.ray_util --env "LiftTaskSawyers-v0" --algo
ppo --evaluate --eval-episodes 100 --seed 0 --load-policy-path
./ray_trained_models/
```

Create a Custom Environment

Using Collaborative Gym, it is possible to create customized environments. The following template can be used to create a new environment.

Create a new environment file in: `collaborative-gym/collaborative_gym/envs/new_task.py`

```
from .base_env import BaseEnv
from ray.rllib.env.multi_agent_env import MultiAgentEnv

# Import all robots
from .agents.jaco import Jaco
from .agents.sawyer import Sawyer
from .agents.panda import Panda

class NewTaskEnv(BaseEnv, MultiAgentEnv):
    def __init__(self):
        ...
    def step(self, action):
        ...
    def _get_obs(self, agent=None):
        ...
    def reset(self):
        ...
```

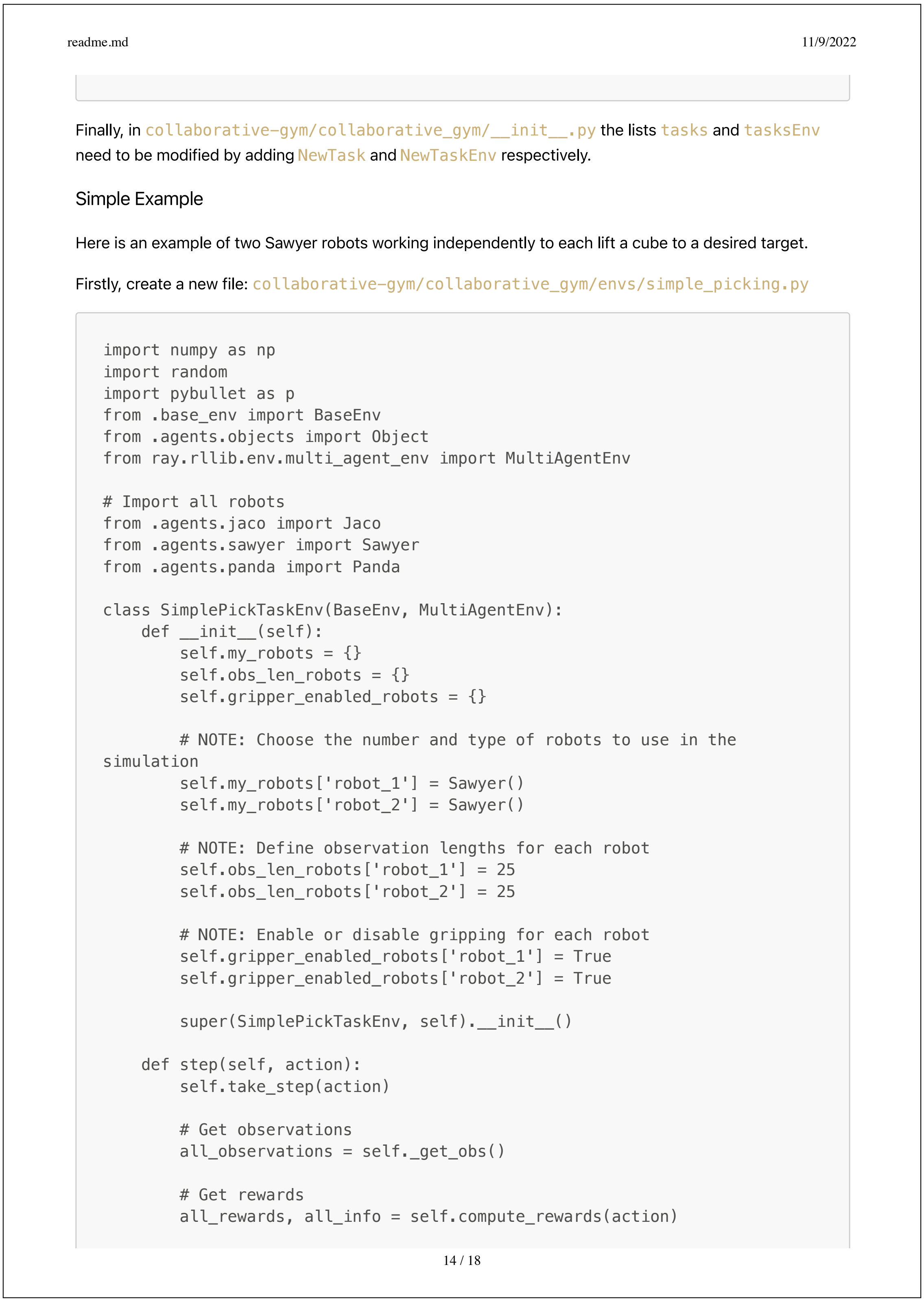
The following line will then be added to the `collaborative-gym/collaborative_gym/envs/__init__.py`

```
from collaborative_gym.envs.new_task import NewTaskEnv
```

And the following lines to `collaborative-gym/collaborative_gym/__init__.py`

```
from collaborative_gym.envs.new_task import NewTaskEnv
```

13 / 18



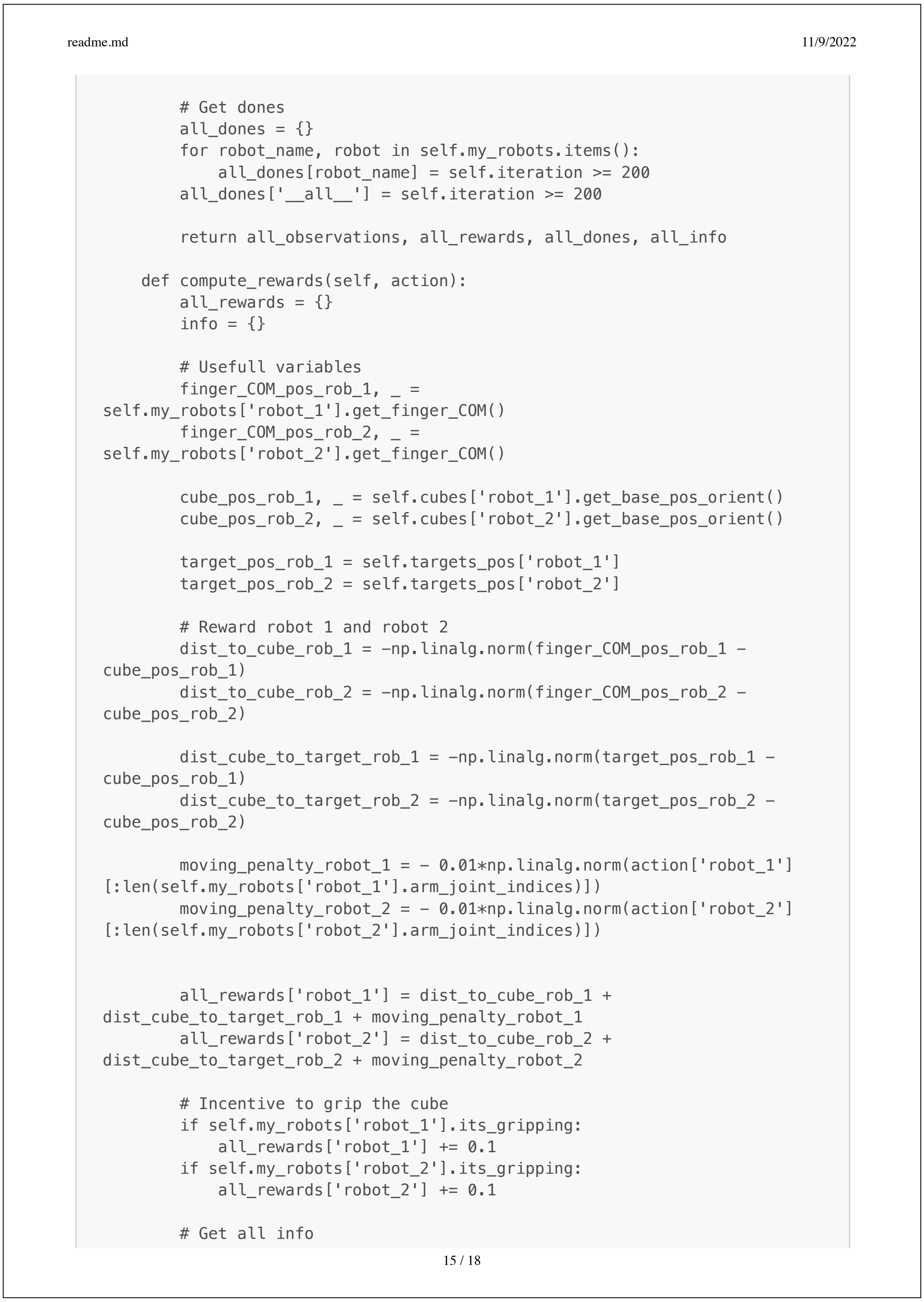
Finally, in `collaborative-gym/collaborative_gym/__init__.py` the lists `tasks` and `tasksEnv` need to be modified by adding `NewTask` and `NewTaskEnv` respectively.

Simple Example

Here is an example of two Sawyer robots working independently to each lift a cube to a desired target.

Firstly, create a new file: `collaborative-gym/collaborative_gym/envs/simple_picking.py`

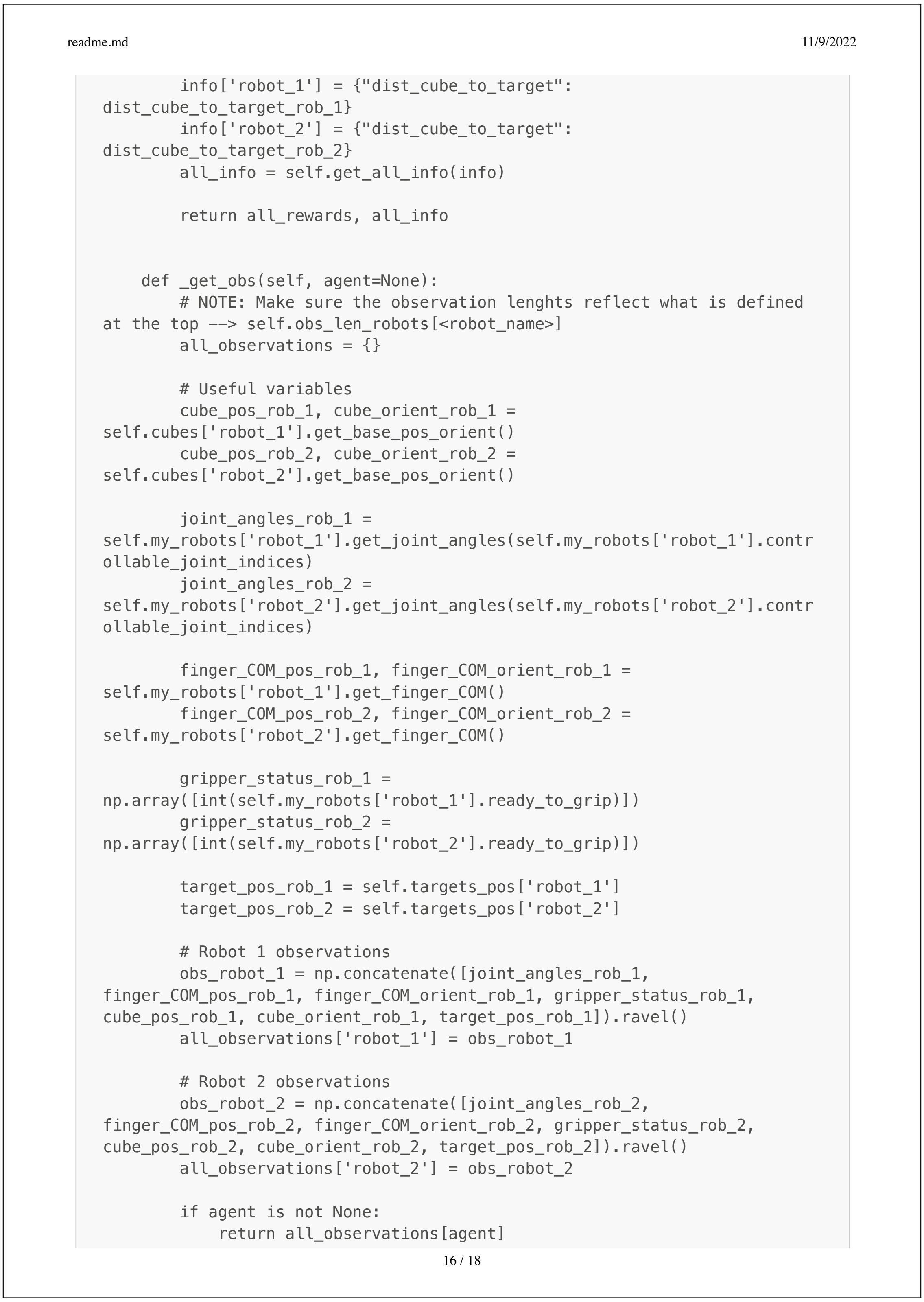
```
import numpy as np
import random
import pybullet as p
from .base_env import BaseEnv
from .agents.objects import Object
from ray.rllib.env.multi_agent_env import MultiAgentEnv

# Import all robots
from .agents.jaco import Jaco
from .agents.sawyer import Sawyer
from .agents.panda import Panda

class SimplePickTaskEnv(BaseEnv, MultiAgentEnv):
    def __init__(self):
        self.my_robots = {}
        self.obs_len_robots = {}
        self.gripper_enabled_robots = {}

        # NOTE: Choose the number and type of robots to use in the
simulation
        self.my_robots['robot_1'] = Sawyer()
        self.my_robots['robot_2'] = Sawyer()

        # NOTE: Define observation lengths for each robot
        self.obs_len_robots['robot_1'] = 25
        self.obs_len_robots['robot_2'] = 25

        # NOTE: Enable or disable gripping for each robot
        self.gripper_enabled_robots['robot_1'] = True
        self.gripper_enabled_robots['robot_2'] = True

        super(SimplePickTaskEnv, self).__init__()

    def step(self, action):
        self.take_step(action)

        # Get observations
        all_observations = self._get_obs()

        # Get rewards
        all_rewards, all_info = self.compute_rewards(action)
```



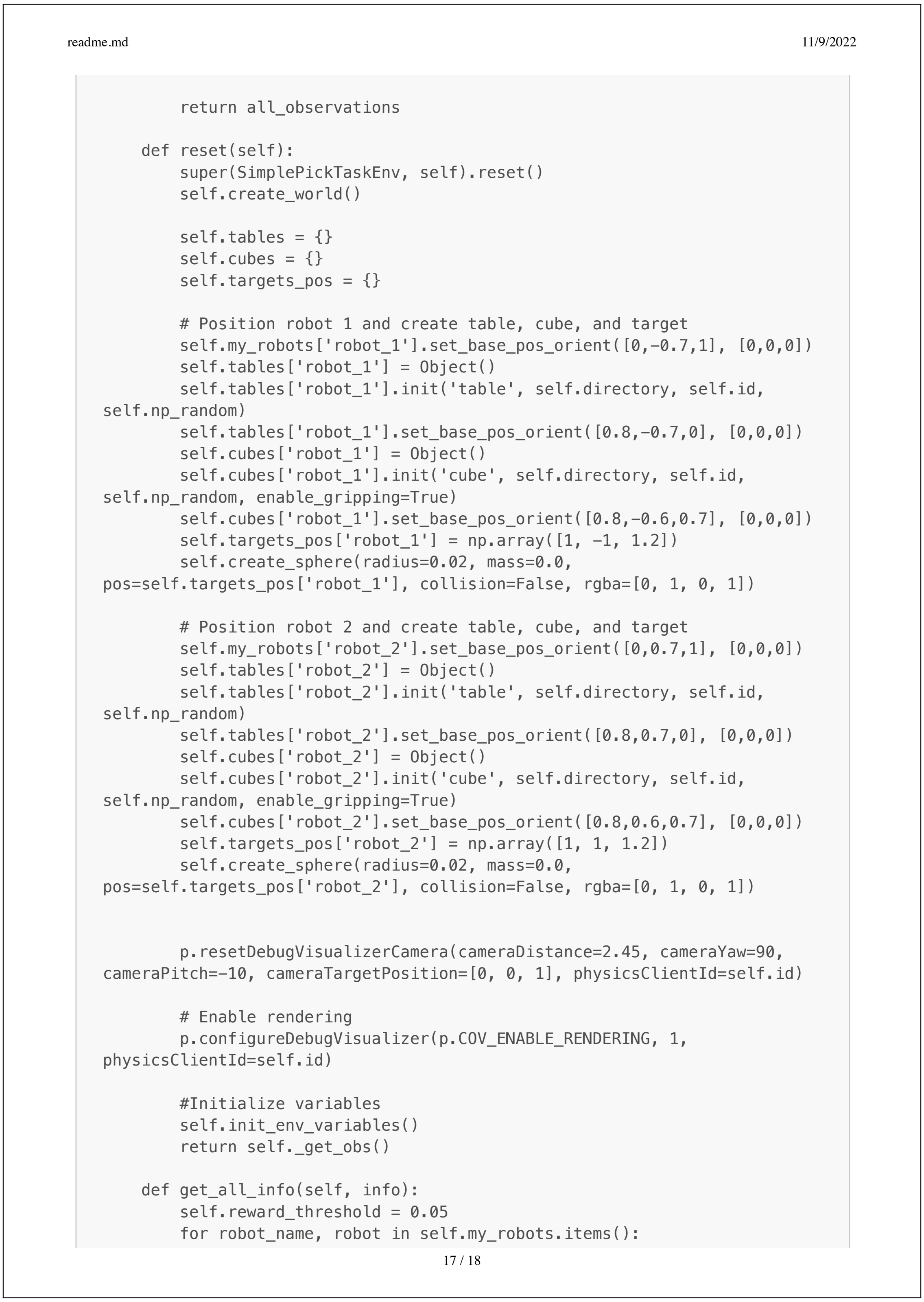
readme.md 11/9/2022

```
        # Get dones
        all_dones = {}
        for robot_name, robot in self.my_robots.items():
            all_dones[robot_name] = self.iteration >= 200
        all_dones['__all__'] = self.iteration >= 200

        return all_observations, all_rewards, all_dones, all_info

    def compute_rewards(self, action):
        all_rewards = {}
        info = {}

        # Usefull variables
        finger_COM_pos_rob_1, _ =
self.my_robots['robot_1'].get_finger_COM()
        finger_COM_pos_rob_2, _ =
self.my_robots['robot_2'].get_finger_COM()

        cube_pos_rob_1, _ = self.cubes['robot_1'].get_base_pos_orient()
        cube_pos_rob_2, _ = self.cubes['robot_2'].get_base_pos_orient()

        target_pos_rob_1 = self.targets_pos['robot_1']
        target_pos_rob_2 = self.targets_pos['robot_2']

        # Reward robot 1 and robot 2
        dist_to_cube_rob_1 = -np.linalg.norm(finger_COM_pos_rob_1 -
cube_pos_rob_1)
        dist_to_cube_rob_2 = -np.linalg.norm(finger_COM_pos_rob_2 -
cube_pos_rob_2)

        dist_cube_to_target_rob_1 = -np.linalg.norm(target_pos_rob_1 -
cube_pos_rob_1)
        dist_cube_to_target_rob_2 = -np.linalg.norm(target_pos_rob_2 -
cube_pos_rob_2)

        moving_penalty_robot_1 = - 0.01*np.linalg.norm(action['robot_1']
[:len(self.my_robots['robot_1'].arm_joint_indices)])
        moving_penalty_robot_2 = - 0.01*np.linalg.norm(action['robot_2']
[:len(self.my_robots['robot_2'].arm_joint_indices)])


        all_rewards['robot_1'] = dist_to_cube_rob_1 +
dist_cube_to_target_rob_1 + moving_penalty_robot_1
        all_rewards['robot_2'] = dist_to_cube_rob_2 +
dist_cube_to_target_rob_2 + moving_penalty_robot_2

        # Incentive to grip the cube
        if self.my_robots['robot_1'].its_gripping:
            all_rewards['robot_1'] += 0.1
        if self.my_robots['robot_2'].its_gripping:
            all_rewards['robot_2'] += 0.1

        # Get all info
```

15 / 18



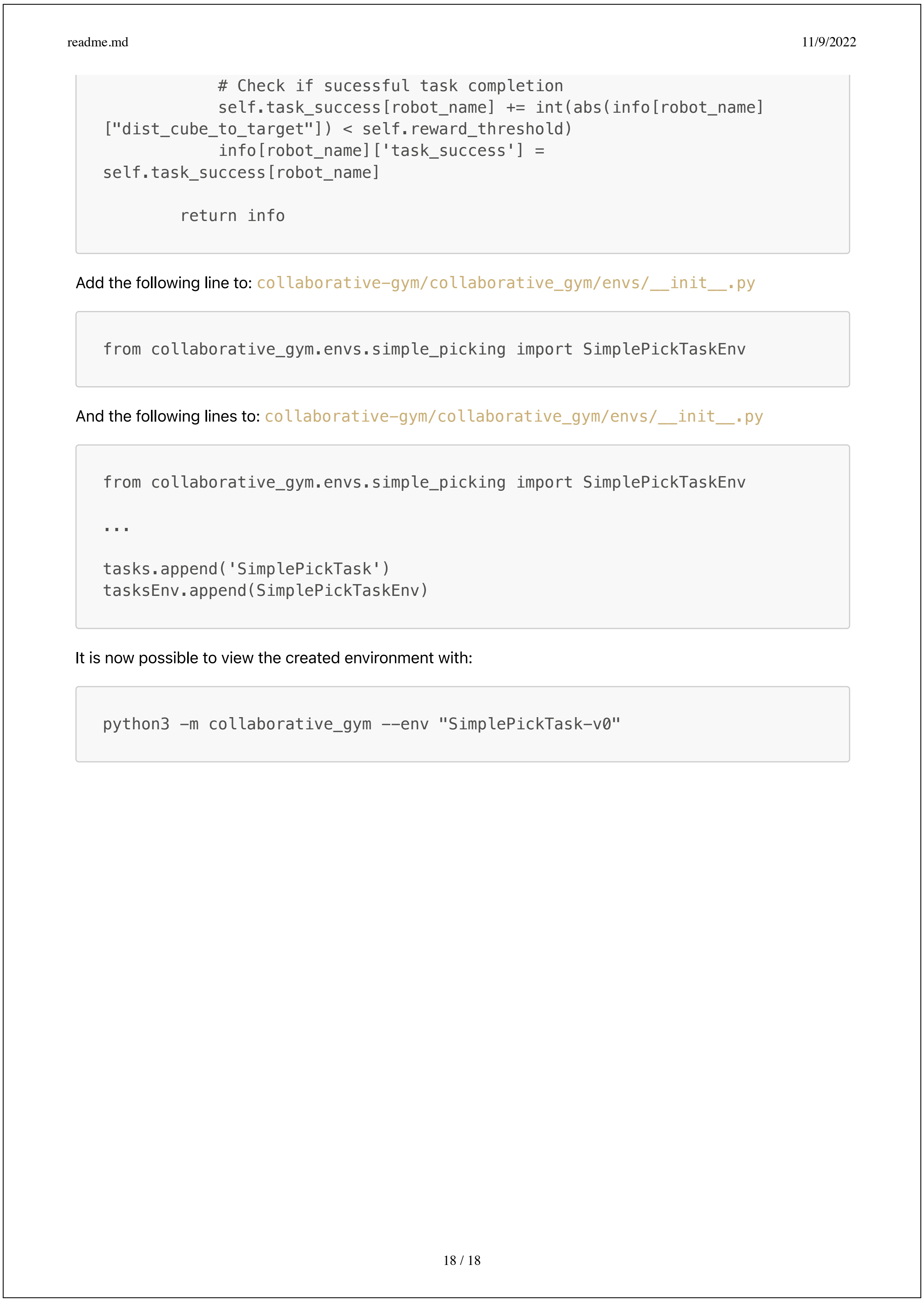
```
        info['robot_1'] = {"dist_cube_to_target":
dist_cube_to_target_rob_1}
        info['robot_2'] = {"dist_cube_to_target":
dist_cube_to_target_rob_2}
        all_info = self.get_all_info(info)

        return all_rewards, all_info


    def _get_obs(self, agent=None):
        # NOTE: Make sure the observation lenghts reflect what is defined
at the top --> self.obs_len_robots[<robot_name>]
        all_observations = {}

        # Useful variables
        cube_pos_rob_1, cube_orient_rob_1 =
self.cubes['robot_1'].get_base_pos_orient()
        cube_pos_rob_2, cube_orient_rob_2 =
self.cubes['robot_2'].get_base_pos_orient()

        joint_angles_rob_1 =
self.my_robots['robot_1'].get_joint_angles(self.my_robots['robot_1'].contr
ollable_joint_indices)
        joint_angles_rob_2 =
self.my_robots['robot_2'].get_joint_angles(self.my_robots['robot_2'].contr
ollable_joint_indices)

        finger_COM_pos_rob_1, finger_COM_orient_rob_1 =
self.my_robots['robot_1'].get_finger_COM()
        finger_COM_pos_rob_2, finger_COM_orient_rob_2 =
self.my_robots['robot_2'].get_finger_COM()

        gripper_status_rob_1 =
np.array([int(self.my_robots['robot_1'].ready_to_grip)])
        gripper_status_rob_2 =
np.array([int(self.my_robots['robot_2'].ready_to_grip)])

        target_pos_rob_1 = self.targets_pos['robot_1']
        target_pos_rob_2 = self.targets_pos['robot_2']

        # Robot 1 observations
        obs_robot_1 = np.concatenate([joint_angles_rob_1,
finger_COM_pos_rob_1, finger_COM_orient_rob_1, gripper_status_rob_1,
cube_pos_rob_1, cube_orient_rob_1, target_pos_rob_1]).ravel()
        all_observations['robot_1'] = obs_robot_1

        # Robot 2 observations
        obs_robot_2 = np.concatenate([joint_angles_rob_2,
finger_COM_pos_rob_2, finger_COM_orient_rob_2, gripper_status_rob_2,
cube_pos_rob_2, cube_orient_rob_2, target_pos_rob_2]).ravel()
        all_observations['robot_2'] = obs_robot_2

        if agent is not None:
            return all_observations[agent]
```



readme.md 11/9/2022

```
        return all_observations

    def reset(self):
        super(SimplePickTaskEnv, self).reset()
        self.create_world()

        self.tables = {}
        self.cubes = {}
        self.targets_pos = {}

        # Position robot 1 and create table, cube, and target
        self.my_robots['robot_1'].set_base_pos_orient([0,-0.7,1], [0,0,0])
        self.tables['robot_1'] = Object()
        self.tables['robot_1'].init('table', self.directory, self.id,
self.np_random)
        self.tables['robot_1'].set_base_pos_orient([0.8,-0.7,0], [0,0,0])
        self.cubes['robot_1'] = Object()
        self.cubes['robot_1'].init('cube', self.directory, self.id,
self.np_random, enable_gripping=True)
        self.cubes['robot_1'].set_base_pos_orient([0.8,-0.6,0.7], [0,0,0])
        self.targets_pos['robot_1'] = np.array([1, -1, 1.2])
        self.create_sphere(radius=0.02, mass=0.0,
pos=self.targets_pos['robot_1'], collision=False, rgba=[0, 1, 0, 1])

        # Position robot 2 and create table, cube, and target
        self.my_robots['robot_2'].set_base_pos_orient([0,0.7,1], [0,0,0])
        self.tables['robot_2'] = Object()
        self.tables['robot_2'].init('table', self.directory, self.id,
self.np_random)
        self.tables['robot_2'].set_base_pos_orient([0.8,0.7,0], [0,0,0])
        self.cubes['robot_2'] = Object()
        self.cubes['robot_2'].init('cube', self.directory, self.id,
self.np_random, enable_gripping=True)
        self.cubes['robot_2'].set_base_pos_orient([0.8,0.6,0.7], [0,0,0])
        self.targets_pos['robot_2'] = np.array([1, 1, 1.2])
        self.create_sphere(radius=0.02, mass=0.0,
pos=self.targets_pos['robot_2'], collision=False, rgba=[0, 1, 0, 1])


        p.resetDebugVisualizerCamera(cameraDistance=2.45, cameraYaw=90,
cameraPitch=-10, cameraTargetPosition=[0, 0, 1], physicsClientId=self.id)

        # Enable rendering
        p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 1,
physicsClientId=self.id)

        #Initialize variables
        self.init_env_variables()
        return self._get_obs()

    def get_all_info(self, info):
        self.reward_threshold = 0.05
        for robot_name, robot in self.my_robots.items():
```

17 / 18



```
            # Check if sucessful task completion
            self.task_success[robot_name] += int(abs(info[robot_name]
["dist_cube_to_target"]) < self.reward_threshold)
            info[robot_name]['task_success'] =
self.task_success[robot_name]

        return info
```

Add the following line to: `collaborative-gym/collaborative_gym/envs/__init__.py`

```
from collaborative_gym.envs.simple_picking import SimplePickTaskEnv
```

And the following lines to: `collaborative-gym/collaborative_gym/envs/__init__.py`

```
from collaborative_gym.envs.simple_picking import SimplePickTaskEnv

...

tasks.append('SimplePickTask')
tasksEnv.append(SimplePickTaskEnv)
```

It is now possible to view the created environment with:

```
python3 -m collaborative_gym --env "SimplePickTask-v0"
```

Collaborative Gym:
A Simulation Benchmark for Multi-Robotic Tasks

Gabriele Ansaldo

Department of Mechanical and Industrial Engineering,
Northeastern University, Boston, MA 02115

Abstract

The design of multi-robot systems has gained increasing attention in recent years. The field of cooperative Multi-Agent Robot Systems (MARS) has shown the potential to provide reliable and cost-effective solutions to a wide range of automated applications. Communication and coordination between autonomous agents require robust and intelligent control systems in order to achieve high-quality performance. This paper presents Collaborative Gym, an open-source, physics-based simulation framework for multi-robot interaction. This simulation environment differs from existing robotic simulation environments in that it is designed to model the interaction between multiple robots. Despite the presence of a large number of single robotic environments, multi-robotic simulation environments for reinforcement learning are rare. Collaborative Gym contains four simulated tasks in which different commercial robots work in collaboration: poking, lifting, balancing, and passing. For each of the four tasks, baseline policies are presented for various combinations of commercial robots which have been trained using reinforcement learning. The study demonstrated that Collaborative Gym is a promising open-source framework for the development of multi-robotic collaborative robotic tasks and policies.

1. Introduction

Modern robotics involves robot-robot cooperation in unstructured environments. For instance, in automated logistics and manufacturing scenarios, such as warehouses, distribution centers, and automotive companies, repetitive and sequential operations can be performed more efficiently by transferring objects between robots equipped with object handover abilities [1]. Recent years have seen a surge in research in the field of cooperative Multi-Agent Robot Systems (MARS). Having efficient coordination among autonomous agents in order to complete tasks is of primary concern in order to achieve high quality performance [2].

Among the most challenging aspects of achieving proper coordination is the design of control architectures that command the robots. It is true that robust and intelligent control systems are required to efficiently and effectively communicate and coordinate among agents in order to accomplish a variety of tasks. Therefore, developing the control architecture has been identified as one of the most relevant aspects. Recent developments in Artificial Intelligence (AI) and Reinforcement Learning (RL) may make it possible to develop robust and efficient intelligent control systems capable of enabling robust and efficient coordination and collaboration between robots in unstructured environments.

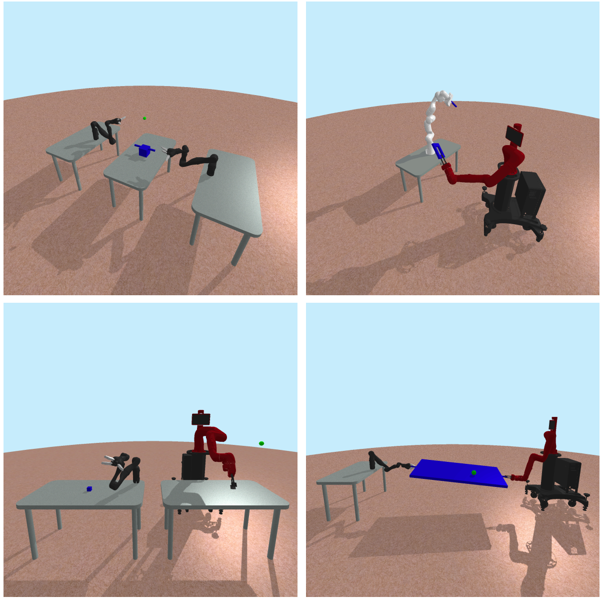


Figure 1: *Four collaborative tasks in Collaborative Gym: lifting, poking, passing, and balancing*

This study contributes to the development of the current RL environment library by introducing a simulation benchmark for MARS. In this paper, Collaborative Gym [1], an open-source, physics-based simulation framework for multi-robot interaction, is presented. Specifically, this study attempts to design various multi-robotic tasks that require collaboration and coordination among robots. Collaborative Gym differs from existing robotic simulation environments in that it focuses on modeling the interaction between multiple robots rather than focusing on a single robot. Collaborative Gym contains four simulated tasks in which different commercial robots such as Sawyer, Panda, and Jaco work in collaboration: poking, lifting, balancing, and passing.

The OpenAI Gym framework integrates directly with Collaborative Gym, enabling the use of control policy learning algorithms, including deep reinforcement learning. This paper provides and evaluates baseline control policies for the four tasks with different combinations of robots.

Collaborative Gym offers several applications for the research community. Firstly, Collaborative Gym can be used as a benchmark for comparing reinforcement learning algorithms for multi-robotic systems. Furthermore, it provides the basis for researchers to create environments and control systems for their own collaborative tasks. Explicitly, through this work the following contributions are made:

- A simulation framework, Collaborative Gym, is developed for multi-robot interaction.
- The various multi-robotic designed tasks can be used as a benchmark to compare control algorithms for multi-agent interaction.

2. Literature Review

2.1. RL Simulation Environments

2.1.1 OpenAI Gym Framework

In reinforcement learning, an agent learns through interaction with its environment. As a result, testbed environments are necessary in order to test and compare the results of different reinforcement learning algorithms. The OpenAI Gym framework provides a collection of reinforcement learning environments for developing and testing reinforcement learning algorithms. Open AI Gym is an open-source toolkit which provides a range of environments for games, control problems, constructing algorithms, performing control tasks, robotics, text games, and more [3]. Among its benchmark environments are Atari games and physics-based locomotion agents. The Open AI framework forms the foundation for Collaborative Gym.

2.1.2 OpenAI Gym Physics Engines

PyBullet, DART, and MuJoCo are three physics engines commonly used in OpenAI Gym for simulating robotic environments [4, 5, 6]. PyBullet is a simulation environment established on Bullet physics-based simulation, which simulates collision detection along with the dynamics of rigid and soft bodies. In addition to reinforcement learning applications, this physics engine has been used for training and validating real robots utilizing physics simulations [7, 8, 9, 10]. This library provides forward dynamics simulation, inverse dynamics computation, forward and inverse kinematics, as well as collision detection and ray intersection queries. As an open-source project, PyBullet has attracted a large community of contributors who continue to develop the simulation environment and provide support for beginners [11]. The Dynamic Animation and Robotics Toolkit (DART) is an open source library that is collaborative and cross-platform. As part of the library, data structures and algorithms are provided for kinematic and dynamic applications in robotics and computer animation. A variety of locomotive environments have been implemented using DART in conjunction with the OpenAI framework [12, 13]. Multi-Joint Dynamics with Contact (MuJoCo) is a simulation environment and physics engine dedicated to robotics, biomechanics, animation, and machine learning. MuJoCo is known for its deep learning applications that enable virtual animals and humanoid models to walk and perform other complex movements [14]. It is not possible to install this simulation environment without a license, as opposed to the other simulation environments described in this work.

2.2. Multi-Agent Robot Systems

There are situations in which Multi-Agent Robot Systems (MARSs) or Multi-Robot Systems (MRSs) may be used to accomplish tasks that would otherwise be difficult for an individual robot to perform, such as when there are uncertainties, incomplete information,

[1] https://github.com/gabriansa/collaborative-gym

asynchronous computations, and distributed control [15]. There has been considerable interest in MRSs and MARSs over the last decades [16, 24, 25, 26, 27, 28, 29, 30, 31, 17, 18, 19, 20, 21, 22, 23]. Several examples of cooperative multi-agent robot applications include soccer robots [32, 33], unmanned guided vehicles (UGV's) and unmanned aerial vehicles (UAV's) [34]. The research in cooperative multi-agent robot systems has focused on three main elements, according to [2]: (1) the types of agents, homogeneous and heterogeneous, (2) the control architectures, reactive, deliberative, and hybrid, and (3) the type of communication, explicit and implicit. Nevertheless, developing the control architecture has been identified as one of the most relevant aspects for achieving efficient coordination among multi-agent robots. Developing MRSs can be challenging due to the fact that it is not possible to predict all possible situations that robots might encounter as well as to specify their behavior in advance. Therefore, it is crucial that robots in MRSs learn from their operating environment and adapt to their counterparts. MRSs are faced with the challenge of addressing learning as a key issue. It has become increasingly popular in recent years to extend individual reinforcement learning (RL) to multiagent systems, especially multi-robot systems (MRSs) [35, 36, 37, 38, 39]. Through the use of multi-agent reinforcement learning, participating robots are able to learn the mapping between their states and the actions they take in response to rewards or payoffs obtained by interacting with their environment [40]. In many ways, MRSs can benefit from RL, where robots are expected to coordinate their behavior in order to achieve their objectives.

2.3. Multi-Agent Reinforcement Learning Overview

2.3.1 Multi-Agent Reinforcement Learning Environments

In recent years, a number of studies have investigated Multi-Agent Reinforcement Learning (MARL) [41, 42, 43, 44]. As a result, it is necessary to create simulation environments that can be utilized as benchmarks in order to test and evaluate MARL algorithms and techniques. Table 1 gives a concise overview of the most relevant multi-agent environments that have been developed.

Aside from a large number of single robotic environments, multi-robotic simulation environments for RL are scarce. Assistive Gym and Robosuite are the only two environments that provide a taste of robotic multi-agent tasks. On the one hand, Assistive Gym has the ability of training policies for robots collaborating with an active human. On the other hand, Robosuite provides some tasks that require collaborative efforts between two robots. Nevertheless, no RL environment is currently available that is fully dedicated to studying multi-agent robotic tasks. Comparatively to existing multi-agent simulation environments, Collaborative Gym solely focuses on multi-robot systems, thus enabling multiple robots to collaborate on a variety of tasks to achieve a common goal.

2.3.2 MARL Techniques

In terms of task type, it is possible to classify Multi-Agent Reinforcement Learning into three categories, namely fully cooperative, fully competitive, and hybrid [54, 55, 56]. For instance, an agent's reward function is the same in a fully cooperative random game. Therefore, the objective of agents is to maximize their mutual returns. It is however possible to have a complete competition scenario when the objectives of two agents are opposite.

- *Fully cooperative learning*: A fully cooperative random game has the same reward function and learning objective for each agent. In the same manner as single agents environments, in a multi-agent setting agents are likely to pursue greedy strategies in order to maximize their returns. However, the agents cannot make their decisions independently, since they share the same objective, they must consider the other agents and cooperate as a group.

- *Fully competitive learning*: In contrast with complete cooperation, a complete competition environment has many similar characteristics, with the exception that the reward function is the opposite for each agent. It is also possible for competition to arise when there are more than two agents involved. However, most of the literature concerning RL in fully competitive games pertains only to two-agent games.

- *Hybrid learning*: In contrast to fully cooperative and competitive agents, hybrid agents tend to be selfish since their reward function is not restricted. A large number of algorithms in this category are devoted exclusively to static problems and are based on the concept of game theory equilibrium.

Environment Name	Description
Robosuite [45]	Simulation framework powered by the MuJoCo physics engine for robot learning. It also offers a suite of benchmark environments for reproducible research.
Assistive Gym [46]	Physics-based simulation framework for physical human-robot interaction and robotic assistance.
Massive Multi-Agent Game Environment [47]	It considers MMORPGs (Massive Multiplayer Online Role Playing Games) the best proxy for the real world among human games: they are complete macrocosms featuring thousands of agents per persistent world, diverse skilling systems, global economies, complex emergent social structures, and ad-hoc high stakes single and team based conflict.
PettingZoo [48]	Python library for conducting research in multi-agent reinforcement learning, akin to a multi-agent version of Gym.
RoboSumo [49]	Sumo-wrestling between two ants using continuous control.
Stratcraft [50] [51]	StarCraft is a 1998 military science fiction real-time strategy game.
OpenAI Multi-Agent Hide and Seek [52]	In our environment, agents play a team-based hide-and-seek game. Hiders (blue) are tasked with avoiding line-of-sight from the seekers (red), and seekers are tasked with keeping vision of the hiders. There are objects scattered throughout the environment that hiders and seekers can grab and lock in place, as well as randomly generated immovable rooms and walls that agents must learn to navigate.
OpenAI Multi-Agent Competition Environments [53]	A collection of various continuous control, multi-agent tasks.

Table 1: *Overview of some of the most relevant Multi-Agent Reinforcement Learning Environments*

As part of a Multi-Agent System (MAS), multiple agents coexist in the same environment and all learn simultaneously. It is not uncommon for one agent's behavior to be affected by the behavior of other agents in the same environment. As a result, Multi-Agent Reinforcement Learning (MARL) methods are typically utilized to the solution of such problems. In general, there are three types of MARL methods: distributed independent learning, centralized learning, and distributed collaborative learning.

- *Distributed independent learning methods*: MARL research can be effectively approached through distributed independent learning methods [57, 58]. Distributed independent learning involves each agent taking other agents into account as part of their environment and learning strategies independently [59]. Due to the forced decomposition of the decision process into multiple Markov decision processes, this method significantly reduces the representation of the state-action space [60]. However, there are two disadvantages to this type of learning:

  1. There is an absence of a coordination mechanism between agents
  2. A relatively poor and suboptimal strategy is obtained

  As a way to mitigate the absence of coordination mechanisms, the limited information of other environmental agents can be used as input to the learning process [61].

- *Centralized learning methods*: As opposed to the distributed independent learning method, the centralized learning method requires each agent to communicate with a central controller and to choose actions in accordance with the controller's instructions [62]. To control the learning process synchronously, the central controller must also be able to perceive the global environment [63]. There are several advantages and disadvantages to this method [61]:

  Advantages:

1. As a Markov decision-making process, the global decision-making process is regarded as a static and closed environment where all agents learn in the same manner
2. It is possible to develop a globally optimal strategy with sufficient learning time

Disadvantages:

1. A slow convergence rate
2. Dimensional disaster
3. Scalability and robustness issues

In relation to the three disadvantages, the centralized learning method usually faces the issue of slow convergence since the agent must explore all joint state-action spaces when the problem scale is large. With regards to dimensional disaster, the joint state-action space increases exponentially as the scale of the problem increases, which leads to an exponential increase in the number of agents. As a final point, centralized learning methods typically face scalability and robustness issues [64]. Since the agent is limited in its observation ability, it is able to perceive the environment only locally, and it cannot obtain information about the surrounding environment at a global level. Considering the limited communication capabilities of agents, centralized learning depends on effective communication between the central controller and all agents.

- *Distributed collaborative learning methods*: Collaborative distributed learning methods combine the benefits of distributed and centralized methods of learning. As part of the method, a collaborative relationship is maintained between the agents and collaborative learning is introduced between the agents based on distributed independent learning [65, 66]. As a result, this method improves the performance of the Multi-Agent Learning System (MALS) at the same time as alleviating the dimensional disaster, robustness, and scalability issues associated with centralized learning techniques.

Collaborative Gym provides a variety of tasks that utilize distributed collaborative learning methods, where agents learn independently while sharing information with other agents and/or sharing reward functions. Nevertheless, Collaborative Gym allows users to design cooperative, competitive, and hybrid tasks that can be trained using either distributed, centralized, or distributed collaborative learning methods.

3. Collaborative Gym

The Collaborative Gym simulation framework provides high-level interfaces for creating and customizing simulation environments for multirobot systems. In addition to integrating directly into the open source PyBullet physics engine, Collaborative Gym environments are built upon the OpenAI Gym interface which allows the use of existing control policy learning algorithms including deep reinforcement learning. A comprehensive analysis of Collaborative Gym is further provided in terms of its control mechanisms, action and observation spaces, and policy learning.

3.1. Actions & Observations

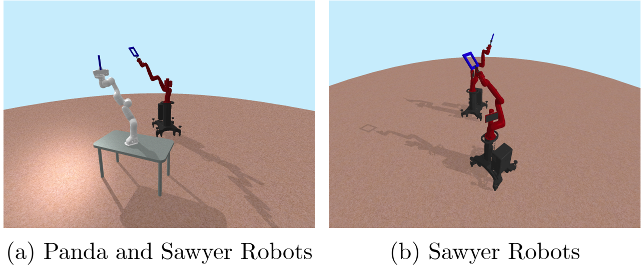
A position control system is provided by Collaborative Gym for controlling the various robots. Actions for each robot's $n - DoF$ arm are represented as changes in joint positions, $\Delta J \in R^n$ depending on the robotic arm. Additionally, a binary variable that selects between gripping and ungripping simplifies the gripping action. Physical limits of an arm affect the ability of a robot to perform actions at any given location. Indeed, each joint of a robot is able to rotate till a certain limit. Actions for the robots' arm are defined as $a = (\Delta J_0, \Delta J_1, \Delta J_2, ..., \Delta J_n, g) \in A$, where $J_0, J_1, J_2, ..., J_n$ are the robot's joint angles and $\Delta J_0, \Delta J_1, \Delta J_2, ..., \Delta J_n$ are delta joint angles that are added to the robot's current joint angles in order to move the robot's arm. Additionally, g is a binary decision variable that allows a robot to choose between grasping and ungrabbing objects in the environment. An object is grasped by the robot when it is close enough to it, and the binary variable g assumes value of 1. For the various robots in the environment, gripping has been simplified to facilitate a simpler and faster learning process. In other words, robot gripping is a complex task that falls within a separate research field, therefore, it was simplified for use in the Collaborative Gym project. Additionally, the process by which gripping occurs plays little role in this study. It is important to note that not all Collaborative Gym tasks require gripping, and therefore the g action may sometimes be neglected depending on the nature of the task.

Robots record observations from the state of the system at each time step, perform actions according to the control policy, and receive rewards at the end of the time step. Observations given to the robots are comparable to those obtained in a real-world scenario involving multiple robots. As an example, these include the position and orientation of the robot's end effector in 3D, the position of the robotic arm's joints in 7D, and the position of the task relevant objects in 3D. Positions are defined in terms of a global coordinate system. It is important to note that observations are task-dependent and may vary according to the study's objectives.

3.2. Collaborative Tasks

A set of four tasks are readily available on Collaborative Gym. These multi-robotic tasks, ranging from simple to more complex, which require collaboration and coordination among robots are further described in details.

- ***1. Poking Task***: In this task one robot holds a stick object while the other holds a "donut" shaped object. The goal is for the two robots to coordinate and insert the stick inside the donut hole.



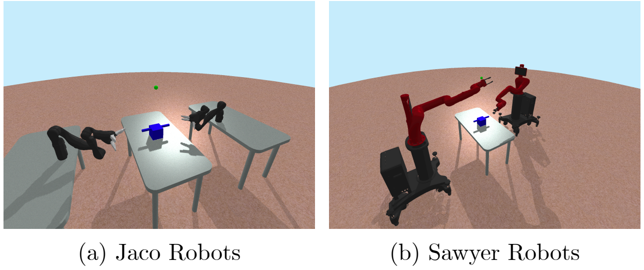
(a) Panda and Sawyer Robots (b) Sawyer Robots

Figure 2: *Poking Task Environment*

  The action space for this task is defined as changes in joint positions ΔJ. Since grasping does not occur in this task, the action binary variable g is not utilized. The observation space for each robot includes the position of the robotic arm's joints in 7D and the position and orientation of the stick and donut objects.

- ***2. Lifting Task***: This task consists of two robots that need to coordinate to lift a heavy pot object. Two robots are required to lift the object since it is too heavy to be lifted by one robot. Indeed, the goal is for the two robots to coordinate and lift the pot object to a specific target position.

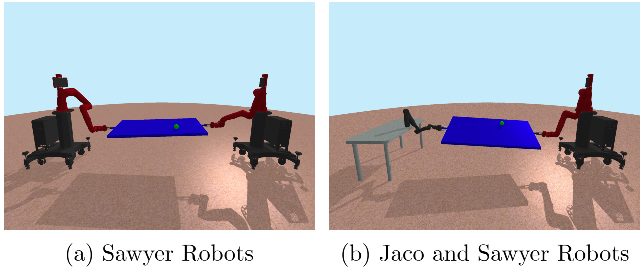
  The action space for this task is defined as changes in joint positions ΔJ. Since grasping occurs in this task, the action binary variable g is required. The observation space for each robot includes the positions of both robotic arm's joints in 7D, the position and orientation of the robot's end effectors in 3D, the status of each gripper (gripping or ungripping), the position and orientation of the pot object and its handles, and the target position.



(a) Jaco Robots (b) Sawyer Robots

Figure 3: *Lifting Task Environment*

- ***3. Balancing Task***: In this task, two robots must coordinate in order to balance a moving sphere on a flat surface. The goal is for the two robots to coordinate and keep the sphere at the center of the board as long as possible.



(a) Sawyer Robots (b) Jaco and Sawyer Robots

Figure 4: *Balancing Task Environment*

  The action space for this task is defined as changes in joint positions ΔJ. Considering that grasping does not occur in this task, the action binary variable g is not taken into account. The observation space for each robot includes the position of both robotic arm's joints in 7D, the position and orientation of the center of the balancing board in 3D, the position orientation of the sphere object, and the linear and angular velocities of the sphere object.

- ***4. Passing Task***: This task consists of two robots that need to coordinate to pass a cube object. Specifically, one robot picks up a cube then hands it over to the second robot which consequently needs to move it to a specific target. Indeed, the goal is for the two robots to coordinate and pass the cube object.

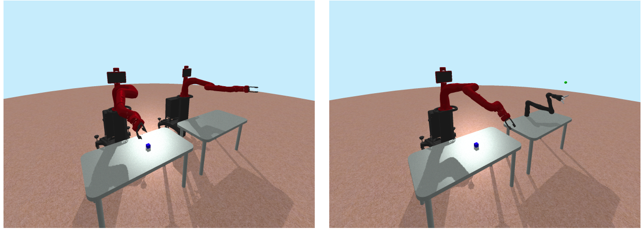

(a) Sawyer Robots (b) Jaco and Sawyer Robots

Figure 5: *Passing Task Environment*

The action space for this task is defined as changes in joint positions ΔJ. As grasping occurs in this task, the action binary variable g must be used. The observation space for each robot includes the positions of the robotic arm's joints in 7D, the position and orientation of both robot's end effectors in 3D, the status of each gripper (gripping or ungripping), the position and orientation of the cube object, and the target position.

4. Policy Training

A variety of deep reinforcement learning techniques can be used to train Collaborative Gym tasks. As this environment is built on the OpenAI Gym framework, any deep reinforcement learning technique can be utilized. It is noteworthy that Collaborative Gym is already connected with RLlib, a freely available open source library that offers support for production-level, highly distributed reinforcement learning workloads with simple and unified APIs that can be used for a variety of industry applications [67]. Due to recent research, PPO-based multi-agent algorithms have shown surprisingly high performance in multiple popular multi-agent testbeds [68]. Collaborative Gym utilizes proximal policy optimization (PPO) as the base algorithm for training robots. However, different RL techniques can be utilized, depending on the type of task. In most Collaborative Gym tasks, a fully-connected neural network is employed with two hidden layers of 256 nodes with relu activations. It is possible to train the multi-agent RL task using (1) cooperative learning with shared or separate policies and/or value functions, (2) adversarial learning using self-play and league-based training, and (3) independent learning of neutral/co-existing agents. Due to the collaborative nature of the designed tasks cooperative learning is utilized.

5. Evaluation

As part of this section, baseline control policies are presented and analyzed for various robots performing the four collaborative tasks. Table 2 shows the combination of robots used for each task. PPO is used to train robot controllers. It was necessary for the algorithm to run for fifty million timesteps in order to generate the policies. As a practical matter, this number proved to be sufficient for the robots to devise a useful policy. Each episode of each task was 200 timesteps long, which means that the robots could observe 200 observations and take 200 actions in order to complete the task. Each episode ended with the robots and environment being reset to their original positions. It is through a series of training episodes that the robot is able to learn which actions are appropriate for completing the task and which are not until an effective policy is established. Depending on the task, training times ranged between 30 and 60 hours. Co-optimization is used to accomplish collaboration, where all robots are trained simultaneously with independent control policies and shared value functions.

Task Name		**Robot 1**	**Robot 2**
Poke Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Panda	Sawyer
Lift Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Jaco	Jaco
Balance Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Jaco	Sawyer
Pass Task	*Variant 1*	Sawyer	Sawyer
	Variant 2	Jaco	Sawyer

Table 2: *Combination of Robot Types for each Task*

Due to the fact that Collaborative Gym uses a variety of robots, it offers the opportunity to study and compare the collaboration between homogeneous and heterogeneous robots in solving various collaborative tasks. A comparison between a variety of tasks with different combinations of robots was conducted by holding all parameters and settings for PPO and the simulation environments constant. It was possible to evaluate the control policies over 200 simulations based on the trained control policies for the specific robots and collaborative task. Based on the 200 simulation rollouts, Table 3 displays the average reward achieved for each task. In addition, Collaborative Gym defines *task completion* and *task performance* for each collaborative task. While *task completion* is defined as the ability to complete the desired task by reaching the desired goal, *task performance* is defined for each collaborative task as follows:

Task Name		**Robot 1**	**Robot 2**	**Mean Reward**	**Task Completion**	**Task Performance**
Poke Task	*Variant 1*	Sawyer	Sawyer	254.23	98.5%	98.49%
	Variant 2	Panda	Sawyer	215.94	100%	99.98%
Lift Task	*Variant 1*	Sawyer	Sawyer	-356.88	98%	29.01%
	Variant 2	Jaco	Jaco	-144.53	98.5%	69.40%
Balance Task	*Variant 1*	Sawyer	Sawyer	-341.90	92%	72.00%
	Variant 2	Jaco	Sawyer	-404.53	73.50%	64.97%
Pass Task	*Variant 1*	Sawyer	Sawyer	-95.22	72.30%	59.93%
	Variant 2	Jaco	Sawyer	198.16	96.00%	95.46%

Table 3: *Average Reward and Success Rate on 200 Trials*

- *Task Performance Poke Task*: The success of the poking task is determined by the ratio between the number of time-steps the stick stays inside the donut and the number of time-steps starting when the stick first enters the donut. As such, this metric does not focus on how long it takes the robots to complete the task, but rather on how well the task has been completed.
- *Task Performance Lift Task*: Similar to the poking task, the success of the lifting task is determined by the ratio of the number of time steps the pot stays within a certain distance to the target position and the number of time steps starting when the pot first reaches the target position. Once again, this metric does not take into consideration how long it takes the robots to complete the task, but instead focuses on the quality at which the task is successfully completed.
- *Task Performance Balance Task*: The success of the balancing task is determined by the ratio between the number of time-steps the ball stays on the balancing board and the total number of time-steps of an episode.
- *Task Performance Pass Task*: Similar to the lifting task, the success of the passing task is determined by the ratio of the number of time steps the cube object stays within a certain distance to the target position and the number of time steps starting when the cube first reaches the target position. Again, the metric does not account for the amount of time required for the robots to complete the task, but instead is concerned with the quality at which the task is successfully completed.

As can be seen in Table 3, *task completion* is close to 100% for most of the trained tasks. There is no doubt that *task completion* is lower for the second variant of the balancing task as well as the first variant of the passing task. It is possible that a lower completion rate is due to both the robots used as well as the difficulty of the task. Indeed, it could be argued that a specific combination of robots might perform better for a certain task than others. As a matter of fact, a relatively high *task completion* rate indicates that the robots were able to learn an effective policy to complete the designed tasks. It appears, however, that *task performance* varies significantly between tasks. As before, the difficulty of the task has a direct relationship with this. In fact, poking is considered to be a trivial task for robots, as a result of the high performance rate. However, when it comes to tasks such as lifting, balancing, and passing, success rates decrease because of task complexity. Interestingly, while most variants of tasks appear to have similar task performance rates, the lift task does not. According to the results, the combination of Jaco robots outperforms the Sawyers by almost 50%. A similar scenario is present for the passing task, whereas the Jaco-Sawyer combination outperforms the Sawyer-Sawyer one. As a result, it is evident that while all robots are capable of achieving the predetermined goal, the performance could be strongly dependent on the type of robot and the combination of robots available for the particular task.

6. Conclusion

As part of this research, a Multi-Agent Robotic System (MARS) environment was developed where robotic arms are capable of coordinating and collaborating to meet a common goal across a wide range of tasks. In this paper, a framework for multi-robot interaction has been presented, called Collaborative Gym, which is an open-source, physics-based simulation framework.

An overview of the action and observation spaces has been provided. Particularly, assumptions related to the grasping of objects have been addressed, which simplifies the learning complexity for the tasks that have been designed. In terms of the learning process, PPO was utilized as the base algorithm for training robots. Results indicate that robots can learn effective collaborative control policies. The purpose of Collaborative Gym is to encourage robots to interact cooperatively in a variety of different tasks. A significant difference between Collaborative Gym and existing robotic simulation environments is that it emphasizes the modeling of the interaction between multiple robots. In addition, Collaborative Gym has been shown to be a valuable tool for benchmarking and developing multiple collaborative environments. The results of the study show that Collaborative Gym is a promising open-source framework for the development of collaborative robots that are capable of solving complex tasks.

Ultimately, this study can be extended to explore the performance of heterogeneous and homogeneous robots in various collaborative tasks. In particular, it would be interesting to assess the learning rate and performance of different robots in different scenarios. It would also be beneficial to examine aspects of simulation-to-reality. As a matter of fact, implementing and transferring learned control policies to a real-world task could provide interesting results. A further consideration would be the design of different collaborative tasks pertaining to industrial applications such as order picking, manufacturing, and logistics. Moreover, it would be interesting to examine which alternative reinforcement learning algorithms are most suitable for multi robotic collaboration. Furthermore, a possible extension of this work could include aspects of human teaming by incorporating simulated humans into the simulation. Using computer vision, real motion data from humans could be used in Collaborative Gym to explore aspects of human-robot collaboration.

References

[1] Marco Costanzo, Giuseppe De Maria, and Ciro Natale. “Handover control for human-robot and robot-robot collaboration”. In: *Frontiers in Robotics and AI* 8 (2021), p. 132.

[2] Zool Hilmi Ismail, Nohaidda Sariff, and EG Hurtado. “A survey and analysis of cooperative multi-agent robot systems: challenges and directions”. In: *Applications of Mobile Robots*. IntechOpen, 2018, pp. 8–14.

[3] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[4] Erwin Coumans and Yunfei Bai. “Pybullet, a python module for physics simulation for games, robotics and machine learning”. In: (2016).

[5] Jeongseok Lee et al. “Dart: Dynamic animation and robotics toolkit”. In: *Journal of Open Source Software* 3.22 (2018), p. 500.

[6] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[7] Jie Tan et al. “Sim-to-real: Learning agile locomotion for quadruped robots”. In: *arXiv preprint arXiv:1804.10332* (2018).

[8] Andy Zeng et al. “Tossingbot: Learning to throw arbitrary objects with residual physics”. In: *IEEE Transactions on Robotics* 36.4 (2020), pp. 1307–1319.

[9] Fereshteh Sadeghi et al. “Sim2real view invariant visual servoing by recurrent control”. In: *arXiv preprint arXiv:1712.07642* (2017).

[10] Konstantinos Bousmalis et al. “Using simulation and domain adaptation to improve efficiency of deep robotic grasping”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[11] *PyBullet Community*. `https://github.com/bulletphysics/bullet3/issues`. 2013.

[12] *GymDart*. `https://github.com/dartsim/gym-dart`. 2018.

[13] *DartEnv*. `https://github.com/DartEnv/dart-env`. 2016.

[14] Joe Booth and Jackson Booth. “Marathon environments: Multi-agent continuous control benchmarks in a modern video game engine”. In: *arXiv preprint arXiv:1902.09097* (2019).

[15] Erfu Yang and Dongbing Gu. “A Survey on Multiagent Reinforcement Learning Towards Multi-Robot Systems.” In: *CIG*. Citeseer. 2005.

[16] Y Uny Cao, Andrew B Kahng, and Alex S Fukunaga. “Cooperative mobile robotics: Antecedents and directions”. In: *Robot colonies*. Springer, 1997, pp. 7–27.

[17] Jiming Liu and Jianbing Wu. *Multiagent robotic systems*. CRC press, 2018.

[18] Maja J Matarić. “Learning in behavior-based multi-robot systems: Policies, models, and other agents”. In: *Cognitive Systems Research* 2.1 (2001), pp. 81–93.

[19] Michael Bowling and Manuela Veloso. “Simultaneous adversarial multi-robot learning”. In: *IJCAI*. Vol. 3. 2003, pp. 699–704.

[20] Imad H Elhajj et al. “Design and analysis of internet-based tele-coordinated multi-robot systems”. In: *Autonomous Robots* 15.3 (2003), pp. 237–254.

[21] Luca Iocchi et al. “Distributed coordination in heterogeneous multi-robot systems”. In: *Autonomous robots* 15.2 (2003), pp. 155–168.

[22] Maja J Matarić, Gaurav S Sukhatme, and Esben H Østergaard. “Multi-robot task allocation in uncertain environments”. In: *Autonomous Robots* 14.2 (2003), pp. 255–263.

[23] Claude F Touzet. “Distributed lazy Q-learning for cooperative mobile robots”. In: *International Journal of Advanced Robotic Systems* 1.1 (2004), p. 1.

[24] Maja J Matarić. “Reinforcement learning in the multi-robot domain”. In: *Robot colonies*. Springer, 1997, pp. 73–83.

[25] François Michaud and Maja J Matarić. “Learning from history for behavior-based mobile robots in non-stationary conditions”. In: *Machine Learning* 31.1 (1998), pp. 141–167.

[26] Tucker Balch and Ronald C Arkin. “Behavior-based formation control for multirobot teams”. In: *IEEE transactions on robotics and automation* 14.6 (1998), pp. 926–939.

[27] Minoru Asada, Eiji Uchibe, and Koh Hosoda. “Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development”. In: *Artificial Intelligence* 110.2 (1999), pp. 275–292.

[28] Marco Wiering, Rafał Sałustowicz, and Jürgen Schmidhuber. “Reinforcement learning soccer teams with incomplete world models”. In: *Autonomous Robots* 7.1 (1999), pp. 77–88.

[29] Sjoerd Van Der Zwaan, JosE AA Moreira, and Pedro U Lima. “Cooperative learning and planning for multiple robots”. In: *Proceedings of the 2000 IEEE International Symposium on Intelligent Control. Held jointly with the 8th IEEE Mediterranean Conference on Control and Automation (Cat. No. 00CH37147)*. IEEE. 2000, pp. 351–356.

[30] Claude F Touzet. “Robot awareness in cooperative mobile robot learning”. In: *Autonomous Robots* 8.1 (2000), pp. 87–97.

[31] Fernando Fernandez and Lynne E Parker. “Learning in large cooperative multi-robot domains”. In: (2001).

[32] Ciprian Candea et al. “Coordination in multi-agent RoboCup teams”. In: *Robotics and Autonomous Systems* 36.2-3 (2001), pp. 67–86.

[33] Bruno Brandão et al. “Multi-Agent Reinforcement Learning for Strategic Decision Making and Control in Robotic Soccer through Self-Play”. In: *IEEE Access* (2022).

[34] Lorenzo Rosa et al. “Multi-task cooperative control in a heterogeneous ground-air robot team”. In: *IFAC-PapersOnLine* 48.5 (2015), pp. 53–58.

[35] Tingxiang Fan et al. “Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios”. In: *The International Journal of Robotics Research* 39.7 (2020), pp. 856–892.

[36] Lin Zhang et al. “Decentralized control of multi-robot system in cooperative object transportation using deep reinforcement learning”. In: *IEEE Access* 8 (2020), pp. 184109–184119.

[37] Junyan Hu et al. “Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning”. In: *IEEE Transactions on Vehicular Technology* 69.12 (2020), pp. 14413–14423.

[38] Yang Yang, Li Juntao, and Peng Lingling. “Multi-robot path planning based on a deep reinforcement learning DQN algorithm”. In: *CAAI Transactions on Intelligence Technology* 5.3 (2020), pp. 177–183.

[39] Guohui Ding et al. “Distributed reinforcement learning for cooperative multi-robot object manipulation”. In: *arXiv preprint arXiv:2003.09540* (2020).

[40] Erfu Yang and Dongbing Gu. *Multiagent reinforcement learning for multi-robot systems: A survey*. Tech. rep. tech. rep, 2004.

[41] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354.

[42] Shariq Iqbal and Fei Sha. “Actor-attention-critic for multi-agent reinforcement learning”. In: *Inter-*

national conference on machine learning. PMLR. 2019, pp. 2961–2970.

[43] Tianshu Chu et al. "Multi-agent deep reinforcement learning for large-scale traffic signal control". In: *IEEE Transactions on Intelligent Transportation Systems* 21.3 (2019), pp. 1086–1095.

[44] Jingjing Cui, Yuanwei Liu, and Arumugam Nallanathan. "Multi-agent reinforcement learning-based resource allocation for UAV networks". In: *IEEE Transactions on Wireless Communications* 19.2 (2019), pp. 729–743.

[45] Yuke Zhu et al. "robosuite: A modular simulation framework and benchmark for robot learning". In: *arXiv preprint arXiv:2009.12293* (2020).

[46] Zackory Erickson et al. "Assistive gym: A physics simulation framework for assistive robotics". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 10169–10176.

[47] Joseph Suarez et al. "Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents". In: *arXiv preprint arXiv:1903.00784* (2019).

[48] J Terry et al. "Pettingzoo: Gym for multi-agent reinforcement learning". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15032–15043.

[49] Maruan Al-Shedivat et al. "Continuous adaptation via meta-learning in nonstationary and competitive environments". In: *arXiv preprint arXiv:1710.03641* (2017).

[50] Nicolas Usunier et al. "Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks". In: *arXiv preprint arXiv:1609.02993* (2016).

[51] Oriol Vinyals et al. "Starcraft ii: A new challenge for reinforcement learning". In: *arXiv preprint arXiv:1708.04782* (2017).

[52] Bowen Baker et al. "Emergent tool use from multi-agent autocurricula". In: *arXiv preprint arXiv:1909.07528* (2019).

[53] Trapit Bansal et al. "Emergent complexity via multi-agent competition". In: *arXiv preprint arXiv:1710.03748* (2017).

[54] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. "Multi-agent reinforcement learning: A selective overview of theories and algorithms". In: *Handbook of Reinforcement Learning and Control* (2021), pp. 321–384.

[55] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. "Multi-agent reinforcement learning: An overview". In: *Innovations in multi-agent systems and applications-1* (2010), pp. 183–221.

[56] Wei Du and Shifei Ding. "A survey on multi-agent deep reinforcement learning: from the perspective of challenges and applications". In: *Artificial Intelligence Review* 54.5 (2021), pp. 3215–3238.

[57] Baoqian Wang, Junfei Xie, and Nikolay Atanasov. "DARL1N: Distributed multi-Agent Reinforcement Learning with One-hop Neighbors". In: *arXiv preprint arXiv:2202.09019* (2022).

[58] Baoqian Wang, Junfei Xie, and Nikolay Atanasov. "Coding for Distributed Multi-Agent Reinforcement Learning". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 10625–10631.

[59] Gerhard Weiß. "Distributed reinforcement learning". In: *The Biology and technology of intelligent autonomous agents*. Springer, 1995, pp. 415–428.

[60] Martin Lauer and Martin Riedmiller. "An algorithm for distributed reinforcement learning in cooperative multi-agent systems". In: *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer. 2000.

[61] Chao Yu et al. "Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit". In: *The Journal of Engineering* 2020.13 (2020), pp. 499–504.

[62] Arbaaz Khan et al. "Scalable centralized deep multi-agent reinforcement learning via policy gradients". In: *arXiv preprint arXiv:1805.08776* (2018).

[63] Piyush K Sharma et al. "Survey of recent multi-agent reinforcement learning algorithms utilizing centralized training". In: *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*. Vol. 11746. SPIE. 2021, pp. 665–676.

[64] Xueguang Lyu et al. "Contrasting centralized and decentralized critics in multi-agent reinforcement learning". In: *arXiv preprint arXiv:2102.04402* (2021).

[65] Pranav M Pawar and Amir Leshem. "Distributed Deep Reinforcement Learning for Collaborative Spectrum Sharing". In: *arXiv preprint arXiv:2104.02059* (2021).

[66] Soummya Kar, José M. F. Moura, and H. Vincent Poor. "$\mathcal{QD}$-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement

Learning Through Consensus + Innovations". In: *IEEE Transactions on Signal Processing* 61.7 (2013), pp. 1848–1862. DOI: 10.1109/TSP.2013.2241057.

[67] Eric Liang et al. "RLlib: Abstractions for distributed reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3053–3062.

[68] Chao Yu et al. "The surprising effectiveness of ppo in cooperative, multi-agent games". In: *arXiv preprint arXiv:2103.01955* (2021).