MATEC Web of Conferences **388**, 04012 (2023) https://doi.org/10.1051/matecconf/202338804012
2023 RAPDASA-RobMech-PRASA-AMI Conference

Simulating object handover between collaborative robots

Beatrice van Eden[1*] and *Natasha* Botha[1]

[1]Centre for Robotics and Future Production, Manufacturing Cluster, Council for Scientific and Industrial Research, South Africa

Abstract. Collaborative robots are adopted in the drive towards Industry 4.0 to automate manufacturing, while retaining a human workforce. This area of research is known as human-robot collaboration (HRC) and focusses on understanding the interactions between the robot and a human. During HRC the robot is often programmed to perform a predefined task, however when working in a dynamic and unstructured environment this is not achievable. To this end, machine learning is commonly employed to train the collaborative robot to autonomously execute a collaborative task. Most of the current research is concerned with HRC, however, when considering the smart factory of the future investigating an autonomous collaborative task between two robots is pertinent. In this paper deep reinforcement learning (DRL) is considered to teach two collaborative robots to handover an object in a simulated environment. The simulation environment was developed using Pybullet and OpenAI gym. Three DRL algorithms and three different reward functions were investigated. The results clearly indicated that PPO is the best performing DRL algorithm as it provided the highest reward output, which is indicative that the robots were learning how to perform the task, even though they were not successful. A discrete reward function with reward shaping, to incentivise the cobot to perform the desired actions and incremental goals (picking up the object, lifting the object and transferring the object), provided the overall best performance.

1 Introduction

Industry 4.0 focuses on the development of a smart factory that is not only highly flexible but also offers reconfigurable facilities [1, 2]. To achieve this, autonomous, safe, and effective robotic systems are needed to allow for more rapid manufacturing practices [1]. These robotic systems should therefore be able to perform complex tasks in unstructured and dynamic environments [1, 3].

Robots will also be expected to collaborate with their human colleagues by interacting with their surroundings and assisting with the task at hand, while working in the same environment [3-6]. The most popular robot to consider in human-robot collaboration (HRC) tasks is the collaborative robot, better known as a cobot [3, 5, 6]. Cobots are equipped with safety systems according to ISO regulations, are easy to install, demand less space and require

* Corresponding author: bveden@csir.co.za

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

fewer modifications in a production environment [5]. One disadvantage of cobots is that when they are programmed for a specific task that they are unable to adapt to any changes in the environment [5]. For example, if their human colleague moves an object or doesn't place it in the exact location for collection the cobot will not be able to collect it. Autonomous behaviour through learning will solve this issue and increase flexibility and adaptability of cobots in a dynamic and unstructured manufacturing environment [1, 5].

In an industrial environment there are several HRC type tasks which normally involve a human, a cobot, an object and a manufacturing process [3]. In some of these tasks there is a true collaboration between the human and robot where they are working together to achieve a common goal: co-manipulation (object handling), handover, assembly, pick-and-place and fetching. These tasks would require some intelligence or automation from the cobot to interact seamlessly with the human. There are also instances where the cobot is simply used as a tool replacing older static technologies, for example, acting as a light source (illumination), holding soldering wire, inspection (quality control), drilling, sanding (surface finish) and screwing. In these cases, it would be sufficient to manually program the cobot's actions as they are not required to interact or respond to a changing environment.

Similar to humans, cobots learn through observation, demonstrations, trial and error, feedback and by asking questions [3]. The most common way to teach cobots, or any robot, intelligence is with reinforcement learning (RL) algorithms – one of three areas of machine learning (ML) [2, 3]. The basic concept of RL is that a robot is trained to take a series of desired actions within their environment by maximising the cumulative reward [6]. Leveraging high-performance computing, RL can be enhanced by combining it with deep neural networks which improves performance, adaptability, and time efficiency [2]. Deep reinforcement learning (DRL) allows robots to learn by themselves to make precise and fast decisions in dynamic and complex situations [2]. It is therefore ideal to use for learning on cobots as it improves its manipulation adaptability, controls the performance, and reduces its reliance on expert knowledge to train [2].

Object handover is a typical task to be performed in an industrial setting and is a core part of other collaborative tasks such as assembly and fetching [1, 4, 6]. In this instance a robot will pick up an object and hand the object over to its human colleague according to a predefined path. When a human colleague is involved, it is easy for the human to adjust their hand position to prevent the cobot from colliding with their hand or wait for release. This is not the case between two cobots as the handover timing needs to be precise. Poor timing could lead to the object being dropped either due to the second cobot's path colliding, or the releasing and gripping activities that are not synched properly. In fact, this is a seamless task among humans as we autonomously predict, perceive, perform an action, learn, and adjust to handover an object [4]. To achieve this with either HRC or robot-to-robot collaboration (R2R) would require a lot of training and algorithmic development.

Most studies in literature consider the object handover between HRC as discussed in Ortenzi et al.'s [4] review. There are far fewer studies that have considered object handover for R2R, or robot-to-robot collaboration [1,7]. This is largely due to a more complex scenario in R2R, as there are no humans involved that can easily adapt to the cobot's behaviour. In an R2R scenario the cobots need to learn to adapt to each other [7].

Sileo et al. [1] proposed an autonomous R2R object handover in the presence of uncertainties. Their experiments were performed using two Franka Panda robot arms. Each robot arm was equipped with an eye-in-hand depth camera for additional perception and object recognition capabilities. They used counter-rotating shafts of different lengths and shapes as the object, which is placed in a box in the field of view of the first robot's camera. Sileo et al. [1] proposed a 6-stage approach to complete the object handover task (Figure 1). To detect the object (stage 1), they investigated two convolutional neural networks (CNN) namely, Faster R-CNN and YOLOv4. YOLOv4 was found to be the best performing

algorithm, able to detect the three different object classes with a higher accuracy than Faster R-CNN. They developed an algorithm which used the visual data as input to estimate the grasping point of the object (stage 2) before picking up the object and moving towards the exchange point (stage 3). The second robot arm has an exteroceptive sensor to detect the presence of the object, before aligning itself to the object's axis using visual input (stage 4). It detects first contact with the object when the estimated force along the object is higher than a predefined threshold and closes its gripper (stage 5). Object handover then takes place where the second robot arm moves the object upward. The first robot arm experiences a force along the object and once a predefined threshold is reached it releases the gripper, and the object is successfully transferred (stage 6). Sileo et al.'s [1] proposed approach was successful in the absence of explicit communication, relying only on visual and sensor input for a predefined algorithm to perform this complex task. However, they were still reliant on input data and algorithmic calculations to perform the task.

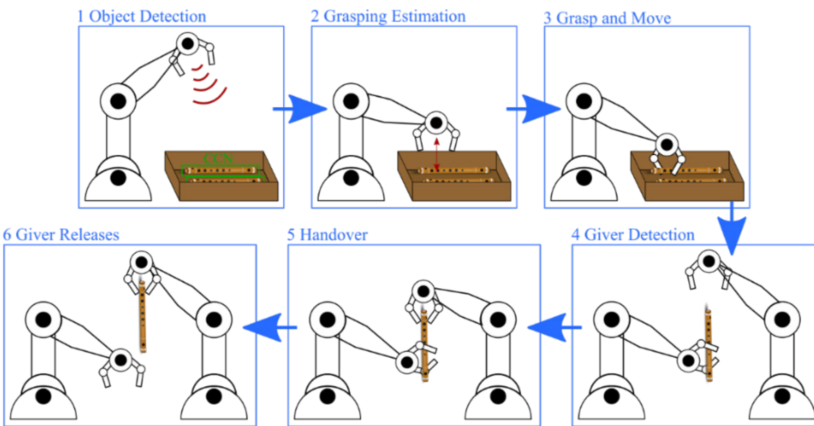


Fig. 1. Object handover 6-stage approach: (1) detecting the object, (2) estimating the grasping point, (3) grasping the object and moving to the exchange point, (4) detecting the object at the exchange point, (5) second robot arm grasps the object, and (6) first robot arm releases the object. (Adapted from Sileo et al. [1])

Costanzo et al. [7] experimentally investigated object handover for both HRC and R2R collaboration tasks. For the R2R task they equipped two Kuka LBR robots with a WSG-50 gripper, an eye-in-hand depth camera to track the object as well as force sensors to determine the gripping force. As with Sileo et al. [1] there is no active communication between the two robot arms and a motion is performed when certain cues are observed. The object is placed in the camera's line of sight of the first robot arm (giver) so that it can track the object and waits for a haptic cue from the second robot arm (receiver) to indicate that it intends on taking the object. Once the second robot arm has gripped the object the first arm performs a backward and forward motion to determine the pulling force. If this force exceeds a predefined threshold and satisfies a few conditions to determine slippage and contact with the gripper from the second robot arm, then the first robot arm releases its gripper, and the object is successfully transferred. A flow diagram for the handover process is illustrated in Figure 2.

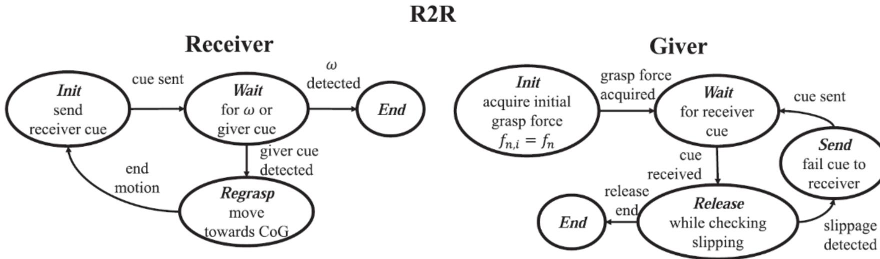


Fig. 2. Flow diagram for both robot arms to achieve an object handover task. (Reprinted from Costanzo et al. [7] with permission under the Creative Commons CC-BY license version 4.0)

Both Sileo et al. [1] and Costanzo et al. [7] used additional data from cameras and sensors as input into the algorithms which control the two cobots' behaviour. We propose an autonomous learning approach using DRL, with no additional sensor or environmental data, to perform object handover between two collaborative robots, safely and efficiently. This study will focus on a simulated environment as there are complexities involved in performing DRL realistically in a real-world environment [8, 9].

The contribution from this study is two-fold. Firstly, with the limited literature available on object handover between two collaborative robots, this study will add to the body of knowledge while also providing a detailed discussion on the development of a simulated environment. The second contribution focusses on the use of DRL to teach collaborative robots to perform object handover, which is a complex task, in a simulated environment. The study emphasises the importance of selecting the right DRL algorithm and reward function to facilitate successful learning.

2 Methodology

The collaborative robot system used in this study is the Franka Emika Panda, illustrated in Figure 3. The Panda arm has 7 degrees of freedom with torque sensors at each joint and a parallel finger gripper [10]. The Panda arm has similar manoeuvrability to a human arm and the torque sensors allow the Panda to handle objects delicately [10]. The Panda arm is a research tool able to interface easily with the robot operating software (ROS) through the *libfranka* and *franka_ros* libraries, transferring and updating data at rates up to 1 kHz [10].

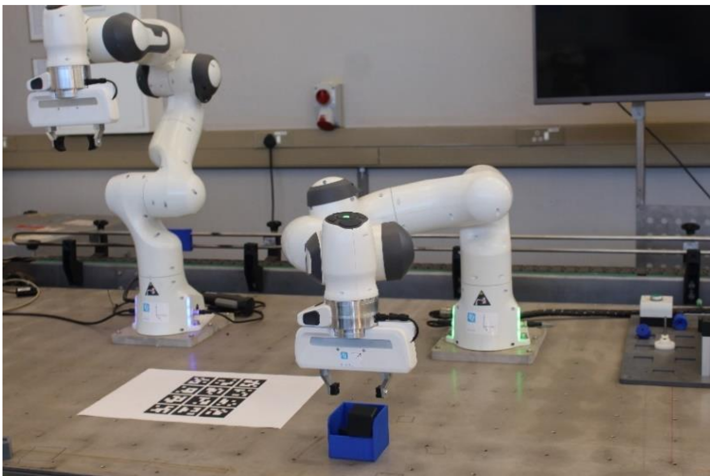


Fig. 3. Franka Emika Panda arms in the CSIR robotics laboratory.

2.1 Task Description

Ortenzi et al. [4] formally defines object handover as a joint action, or any form of social interaction to coordinate actions, between a giver and receiver. A successful joint action is dependent on their ability to share representations, predict actions, and integrate all actions.

For this study the object handover task is illustrated in Figure 4 and was kept very simple. Cobot arm 1 will first pick up the object (Figure 4(a)) and then lift it above the height threshold (Figure 4(b)). Once there cobot arm 1 will stop moving and cobot arm 2 will move towards the object. Once cobot arm 2 has gripped the object, cobot arm 1 will let go (Figure 4(c)). Cobot arm 2 then moves the object safely away until a positional threshold is reached (Figure 4(d)).

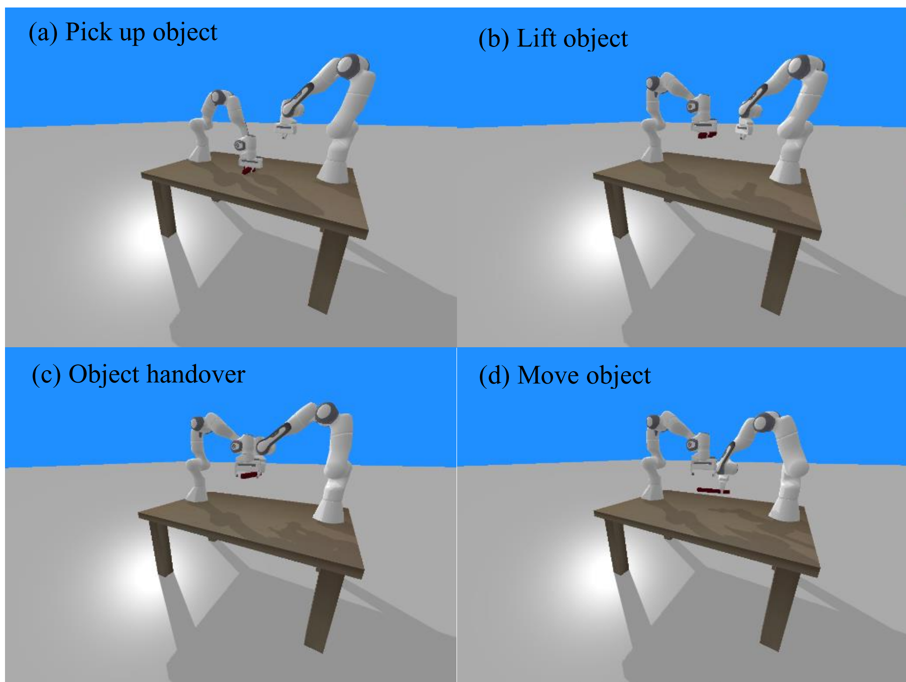


Fig. 4. Illustration of the object handover task within the simulation environment.

The most critical point is the object handover as cobot arm 1 needs to realise when cobot arm 2 has gripped the object. If it doesn't there are two potential failure scenarios that can occur:

1. Cobot arm 1 won't let go resulting in either a tug of the object where both arms could perhaps move together with the object, similar to co-manipulation.
2. Cobot arm 1's grip is stronger and when cobot arm 2 moves to place the object it leaves the transfer area without it.

2.2 Reinforcement Learning

Reinforcement learning (RL) is one of the three core machine learning groups and is often used to optimise the performance of sequential decision processes [2]. RL does this using Markov decision processes (MDP). MDP is a graphical model where an agent interacts with an environment under a policy, is rewarded based on its actions, and outputs an observation

about the state [5, 6]. This process is illustrated in Figure 5, where the *agent* is the Panda arms and the *environment* the object handover task.

There are two categories of RL: model-based and model-free. In a model-based RL the state transition probability of the system is known, and a model can therefore be developed [2]. Model-free RL is often used as it is difficult to develop a model of the environment transition beforehand [5, 6]. Deep reinforcement learning (DRL) uses deep neural networks (DNN) to improve the decision-making capabilities of RL [2, 5] and is often part of the agent framework as illustrated in Figure 5. A DNN is used to extract environmental information based on the state and reward to infer an optimal policy [2].

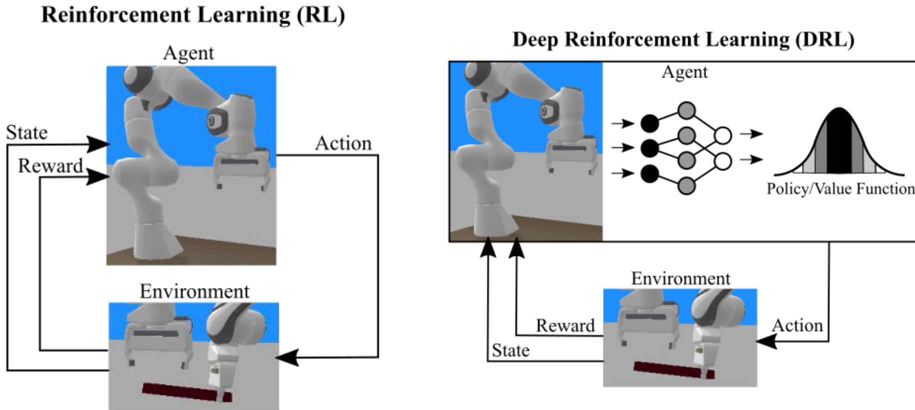


Fig. 5. Schematic illustration of a learning framework for reinforcement learning (RL) and deep reinforcement learning (DRL).

For the simulation environment we used OpenAI Gym (v0.21.0) and Pybullet. For compatibility we selected the *stable-baselines3* (v1.8.0) Python module which contains standard DRL models, including model-free actor-critic algorithms which are mainly used in DRL [2]. An actor-critic algorithm combines a state-action pair value function with policy-based learning algorithms and uses two DNN's simultaneously to approximate the value and policy functions [2]. The actor-network for the policy function generates actions to interact with the environment, and the critic-network for the value function evaluates the actor's performance and guides its next actions [2].

There are seven DRL algorithms implemented in the *stable-baselines3* module: Advantage Actor-Critic (A2C) [12], Deep Deterministic Policy Gradient (DDPG) [13], Deep Q Network (DQN) [14], Hindsight Experience Replay (HER) [15], Proximal Policy Optimisation (PPO) [16], Soft Actor Critic (SAC) [17], and Twin Delayed DDPG (TD3) [18]. The reader is referred to each of the papers for details surrounding the mathematical formulation and implementation for each algorithm.

For the purposes of this study, and computational resource constraints, only one algorithm from each family is considered. Unfortunately, DQN cannot be considered as it requires a discrete action space, and a continuous action space is used in this study. HER would require the definition of an achieved and desired goal within the observation space, which for this task might add unnecessary complexities to the implementation. A2C, DDPG and PPO are therefore selected for comparison in this study as they are more efficient in training compared to their siblings SAC and TD3.

A2C is known for its stability and sample efficiency. It updates the policy and value function simultaneously, which can lead to faster convergence and more stable learning. It can also optimally make use of parallel processing, allowing for more efficient use of computational resources and faster training. However, A2C is more challenging to train since

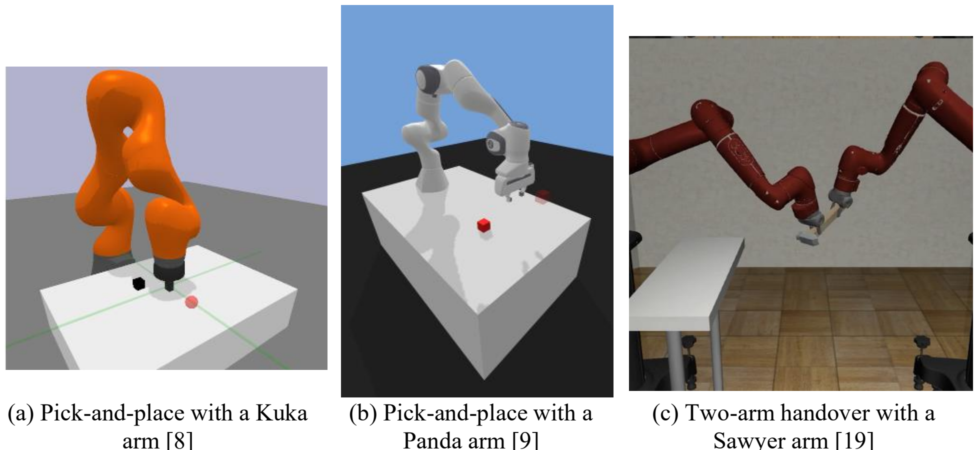
many policy gradient methods can suffer from high variance in the gradients (overfitting to noisy or unrepresentative training data). A2C is also sensitive to hyperparameter choices and care should be taken when choosing these for optimal performance.

DDPG is designed for problems with continuous action spaces and can handle tasks that involve controlling real-valued actions, such as robotics or autonomous driving. It combines Q-learning with policy gradients, leading to relatively stable training. DDPG requires many interactions with the environment, which can be time-consuming and computationally expensive. It may not perform well in tasks where exploration in a continuous action space is critical.

PPO is more sample-efficient than DDPG as it uses importance sampling and clip-based objective functions to stabilise training. It is considered a robust algorithm and has less hyperparameters to tune, making it easier to implement. Like A2C, PPO can also benefit from parallel environments, enhancing training speed. While PPO is more sample-efficient than some other algorithms, it can still be computationally intensive, especially in complex environments. PPO still suffers from noisy gradient estimates due to the use of importance sampling, which can affect training stability.

2.3 Simulated Environment

There are two OpenAI gym environments available which use Pybullet and have defined standard manipulation tasks such as reach, push, slide, pick and place, and stack [8, 9] for a single robotic arm. Only Zhu et al. [19] (using OpenAI gym and Mujoco) has an option, in addition to the standard tasks for single arms, for a two-arm object lifting, peg-in-hole and handover. These three potential simulation environments are illustrated in Figure 6. It was not possible to use the existing environment created by Zhu et al. [19] as there are errors upon installation that are difficult to debug. Initially the panda-gym [9] environment was considered as it already had a Panda arm incorporated, but after implementing a second arm it appeared that the second arm was purely a mirror of the first. This is due to the inheritance of a class which initiates a single robot arm. To modify this class would have taken longer than to just create a new custom environment.



(a) Pick-and-place with a Kuka arm [8] (b) Pick-and-place with a Panda arm [9] (c) Two-arm handover with a Sawyer arm [19]

Fig. 6. Examples of existing OpenAI gym manipulator environments. (Images reprinted from articles with permission under the Creative Commons CC-BY license version 4.0)

2.3.1 Creating a custom environment

To create the simulated environment, OpenAI gym was used with Pybullet as the physics engine which has a Franka Emika Panda arm already available as an asset. The custom environment is a Python class which inherits from *gym.Env* and has five basic definitions to complete: *__init__*, *step*, *reset*, *render* and *close*.

The *__init__* definition initialises the global variables, connects the Pybullet simulation and defines the action and observation spaces. The *action space* defines a set of the possible actions that the agent can take in the environment. Note that [-1, 1] is commonly used for continuous control RL, but it would be too large considering that there are two robot arms. Based on the desired task and potential actions a [-0.5,0.5] space would be sufficient. The *observation space* defines a set of possible observations in an environment. It is commonly defined as a [-10, 10] multi-dimensional continuous space.

The *render* definition defines the necessary camera related arguments to ensure that the simulation window is viewed correctly. This is normally standardised and therefore similar to other OpenAI gym environments. We used the same arguments and lines of code as defined in the panda-gym environment [9].

The *close* definition has a single line which disconnects the Pybullet simulation.

The *reset* definition is largely used to define the simulation environment by loading the different agents and objects as well as defining the initial positions and orientations. The simulated environment generated based on the *reset* definition is shown in Figure 7.

The *step* definition is by far the most important as it defines the actions (with an action array as input) that the agent should take, determines the state, and calculates the reward. A robot arm action is determined using built-in Pybullet functions. First the current pose is determined and then the new action (provided as input into the definition) is calculated in cartesian coordinates. This new position array is then used to calculate the inverse kinematics to output the joint positions which are then used to set the joint motor control. A step in the simulation is then performed and the state of each cobot arm is determined. The most important part of this definition is to calculate the reward function.

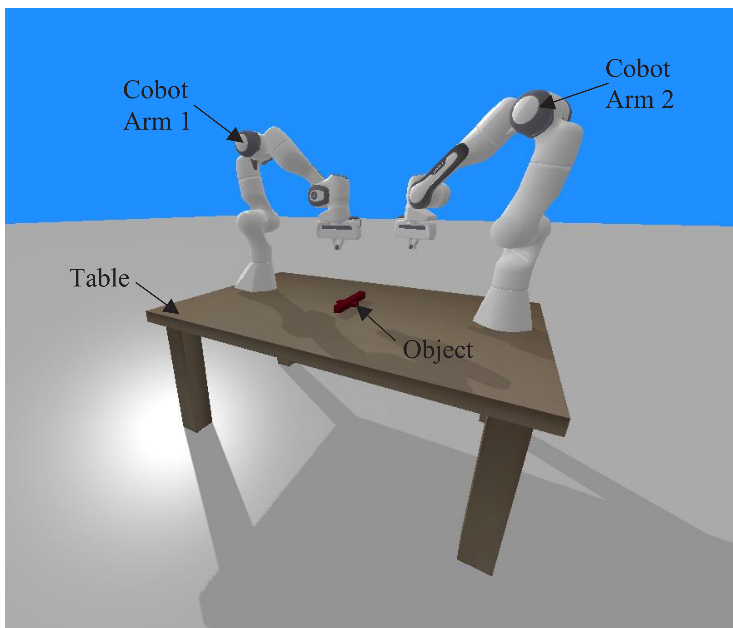


Fig. 7. Custom environment with two Panda arms and an object.

To determine the reward function there are two basic equations that will be considered. The first calculates the Euclidean distance between the cobot arm end effector and the object and compares it to a threshold (δ):

$$\|d_{cobot} - d_{object}\| \leq \delta \tag{1}$$

Reward shaping is also considered as the simulation has difficulty in solving the RL problem due to the random nature of the actions. To coax the cobot arm into reaching for the object a reward proportional to the distance from the object is given:

$$reward\ shape = 0.25\left[1 - \tanh\left(1 - \|d_{cobot} - d_{object}\|\right)\right] \tag{2}$$

The reward function is the most important component towards ensuring a successful simulation of the object handover task. As such we considered three different approaches: (1) implementing the reward function defined by Zhu et al. [19] for the two-arm handover task; (2) developing a reward function considering only discrete rewards at each intermediate goal; and (3) including reward shaping in the defined discrete reward function.

With the two arms it was also decided to implement flags within the reward function to determine when an arm is active, and therefore performing actions, and when an arm is deactivated i.e., remains stationary. This will assist with potential collisions between the two arms and increase the probability of a successful simulation.

(1) Zhu et al. [19] Reward Function

The complete reward function is illustrated in Algorithm 1. The reward function has six steps:

1. ***Robot Arm 1 reaching for the object:*** A reward of [0, 0.25] is given that is proportional to the distance between the arm and the object.
2. ***Robot arm 1 grasping the object:*** A reward of 0.5 is given if the arm is gripping the object and 0 otherwise.
3. ***Robot arm 1 lifting the object:*** A reward of 1.0 is given if the object is lifted past the height threshold of 0.25 from the table and 0 otherwise.
4. ***Robot arm 1 moves object to target location:*** A reward of [1.0, 1.25] is given if the arm is actively lifting the object. This reward is proportional to the distance between the object and the second arm, and 0 otherwise. A flag is initiated for arm 2 to start taking actions and arm 1 to remain stationary.
5. ***Object transfer:*** A reward of 1.5 is given if both arms are gripping the object where the first arm is lifting the object above the table, and 0 otherwise.
6. ***Robot arm 2 grasping the object:*** A reward of 2.0 is given when the arm is gripping the object and keeps it lifted above the table, and the simulation completes successfully.

(2) Discrete Reward Function

The discrete reward function developed in this study is shown in Algorithm 2. The logic is similar to Algorithm 1 but approached in a different way by establishing whether the arms made contact with anything, whereas Algorithm1 is purely dependent on the object position. The reward function has four steps:

1. ***Robot arm 1 grasping the object:*** A reward of 0.25 is given if the arm grasps the object.
2. ***Robot arm 1 lifting the object:*** A reward of 0.5 is given if the arm lifts the object above the 0.25 height threshold. Initialise the movement of arm 2 and prevent any further movement of arm 1.

Alg. 1. Object handover reward function from Zhu et al. [19]

```
1:  Initialise the reward to 0.
2:  Determine the object height from its current state.
3:  if object_height > 0.25 then
4:    Initiate movement of arm 2 and prevent further movement of arm 1
5:    if distance_transfer using Equation (1) < 0.1 then
6:    if distance_pickup using Equation (1) < 0.1 then
7:      reward = 1.5
8:    else then
9:      reward = 2.0
10:     simulation_done = True
11:  else then
12:    reward = 1.0
13:    distance_object using Equation (1)
14:    reward_shape using Equation (2)
15:    reward = reward + reward_shape
16:  else then
17:    if distance_pickup using Equation (1) < 0.1 then
18:      reward = 0.5
19:    else then
20:      distance_object using Equation (1)
21:       reward = reward + reward_shape using Equation (2)
```

3. ***Robot arm 2 grasping the object:*** A reward of 0.75 is given if the arm grasps the object and robot arm 1 releases its grip on the object.
4. ***Robot arm 2 safely moving object away:*** A reward of 1.0 is given if the arm moves the object safely away, and the simulation completes successfully.

Alg. 2. Object handover discrete reward function developed in this study.

```
1:  Initialise the reward to 0.
2:  if arm 1 has made contact then
3:    if distance_pickup using Equation (1) < 0.1 then
4:      reward = 0.25
5:      if object_height > 0.05 then
6:        if object_height > 0.25 then
7:          reward = 0.5
8:          Initialise movement of arm 2 and prevent further movement of arm 1
9:  if arm 2 has made contact then
10:   if distance_transfer < 0.1 then
11:     reward = 0.75
12:     Open gripper fingers to release object
13:     if object_height < 0.1 or object_y > 0.2 then
14:       reward = 1.0
15:       simulation_done = True
```

(3) Discrete Reward Function with Reward Shaping

The same discrete reward function (Algorithm 2) is considered but reward shaping is added as shown in Algorithm 3. The reward function steps are therefore the same as for the discrete reward function with the addition of reward shaping to coax the arms into reaching for the object:

1. ***Robot arm 1 reaching for object:*** A reward of [0, 0.25] is given proportional to the distance between the object and arm 1.
2. ***Robot arm 1 is lifting object:*** A reward of [0.25, 0.50] is given proportional to the distance between the height threshold of 0.25 and the height of the object. The movement of arm 2 is initialised, and arm 1 is deactivated to remain stationary.

3. ***Robot arm 2 reaching for object:*** A reward of [0.5, 0.75] is given proportional to the distance between the object and arm 2.
4. ***Robot arm 2 is moving object:*** A reward of [0.75, 1.0] is given proportional to the distance between the object and the target position, and the simulation is successful once the target end position is reached.

Alg. 3. Object handover reward function with reward shaping developed in this study.

```
1:  Initialise the reward to 0.
2:  if arm 1 has made contact then
3:     if distance_pickup using Equation (1) < 0.1 then
4:        reward = 0.25
5:        if object_height > 0.05 then
6:           if object_height > 0.25 then
7:              reward = 0.5
8:              Initialise movement of arm 2 and prevent further movement of arm 1
9:           else then
10:             reward = 0.25
11:             distance_object_height using Equation (1)
12:             reward = reward + reward_shape using Equation (2)
13:       else then
14:          distance_object_arm1 using Equation (1)
15:          reward = reward + reward_shape using Equation (2)
16: if arm 2 has made contact then
17:    if distance_transfer < 0.1 then
18:       reward = 0.75
19:       Open gripper fingers to release object
20:       if object_height < 0.1 or object_y > 0.2 then
21:          reward = 1.0
22:          simulation_done = True
23:       else then
24:          reward = 0.75
25:          distance_object_position using Equation (1)
26:          reward = reward + reward_shape using Equation (2)
27:    else then
28:       reward = 0.5
29:       distance_object_arm2 using Equation (1)
30:       reward_shape using Equation (2)
```

2.3.2 Integrating with reinforcement learning

As mentioned in Section 2.2 the *stable-baselines3* Python module, which includes several DRL algorithms, is used in this study. To integrate the custom environment with DRL is very simple and only requires making the gym environment, initialising the DRL algorithm and setting up the model to train.

Based on literature [8, 9] at least 1 million timesteps are required to reach a successful task completion. Each algorithm will run until it either successfully completes the task or 2 million timesteps are reached.

The algorithm implementations as provided in *stable-baselines3* was considered with no changes made and the default values for the hyperparameters were used. For all DRL's the *MultiInputPolicy* was considered which allows for handling multiple policy input types using a dictionary. For this study, no hyperparameter tuning was considered as the aim was to develop a simulation that would result in a successful object handover between the two cobot arms. That being said, with hyperparameter tuning the chances of success are increased,

however without properly understanding the mathematical implementation it might not be as successful, especially with a trial-and-error approach.

3 Results and Discussion

A summary of the results from different combinations of reward function and DRL algorithm are provided in Table 1. None of the simulations terminated on their own which would have been the case if the task was successfully completed. The simulations were therefore stopped around 2 million timesteps, except for the DDPG algorithm which was stopped around 1 million timesteps. None of the simulations terminated on their own which is a clear indication that the object handover was not successful. Nonetheless, the different DRL algorithms and reward functions are compared and interrogated in the following subsections.

Table 1. Summary of the comparative DRL and reward function results.

DRL Algorithm	Reward Function	Total Timesteps	Maximum Reward	Successful
A2C	Zhu et al. [15]	2 000 500	30.85	No
	Discrete	2 000 500	0.28	No
	Discrete with Reward Shaping	2 006 500	42.14	No
DDPG	Zhu et al. [15]	1 337 600	17.84	No
	Discrete	1 185 186	0.04	No
	Discrete with Reward Shaping	1 881 483	61.54	No
PPO	Zhu et al. [15]	2 000 896	48.5	No
	Discrete	2 004 992	22.07	No
	Discrete with Reward Shaping	2 000 896	62.96	No

3.1 Evaluation of the Deep Reinforcement Learning Algorithms

It was observed that the DDPG algorithm is more computationally expensive than either A2C or PPO. On an Ubuntu 21.10 Virtual Machine with 4 CPUs A2C ran 4 s per 1000 timesteps, PPO 5 s, and DDPG 26 s.

The maximum reward for the different algorithms is illustrated in Figure 8, where the PPO algorithm generally provided the overall best reward, followed by A2C and DDPG. It is difficult to say whether this increased reward is indicative of the Panda arms performing some aspects of the object handover task, and to what extent. Unfortunately, it was not possible to interrogate all the saved models for each considered case as there were errors during loading which could not be resolved. However, based on those that could be interrogated, only the PPO algorithm (with discrete reward function) managed to grip the object, but wasn't able to lift it to the desired height. Both A2C and DDPG were not even close to gripping the object.

The overall performance of each algorithm is presented in Figure 9 as a function of the different reward functions. The PPO algorithm outperforms both the A2C and DDPG algorithms, showing a consistent linear increase in the reward function before converging. This is indicative of learning and shows the most promise of being able to successfully complete the object handover task. However, a noisy gradient is observed, which can be improved upon through hyperparameter tuning to stabilise training. These results clearly

indicate that PPO is more sample-efficient in complex environments, such as object handover.

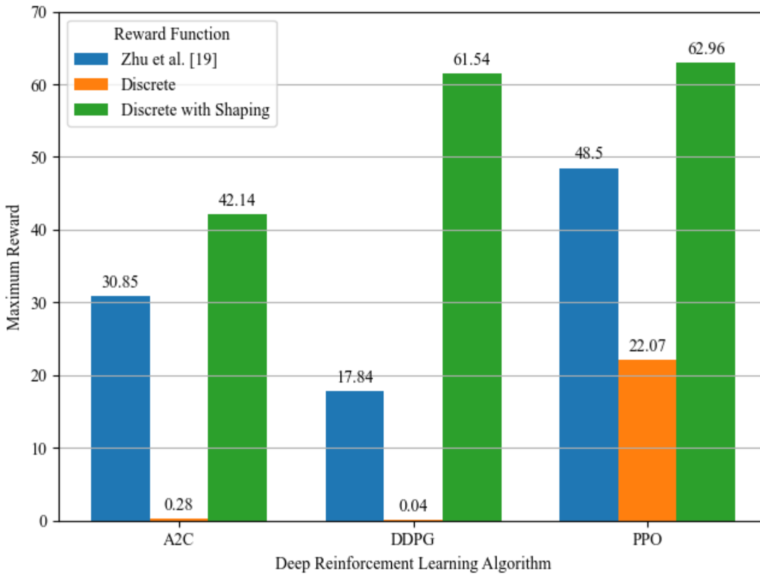


Fig. 8. Maximum reward as a function of the different DRL algorithms and reward functions.

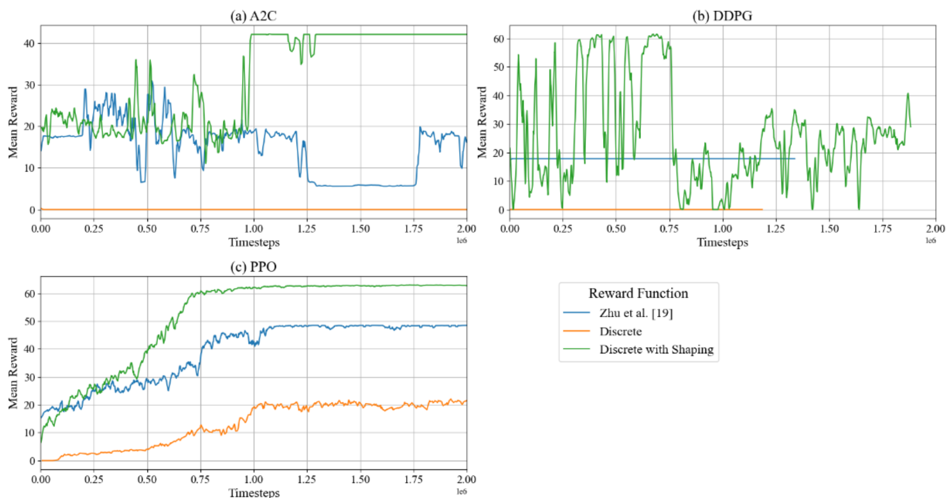


Fig. 9. Overall performance of each algorithm as a function of (a) Zhu et al. [19], (b) discrete, and (c) discrete with reward shaping reward functions.

The A2C and DDPG algorithms have a more erratic pattern which is indicative of the agent performing actions but not really learning from the associated rewards. As a policy gradient method, A2C, can suffer high variance in the gradients as observed in Figure 9. This is due to overfitting noisy or unrepresentative training data which can be addressed with hyperparameter tuning. A2C also normally performs better for discrete action spaces, whereas this study utilised a continuous action space, so the poorer performance is to be expected.

DDPG was expected to perform much better than A2C as it is ideally suited to problems with continuous action spaces. However, it is computationally expensive and requires a lot more timesteps to achieve the desired result.

3.2 Evaluation of the Reward Functions

The maximum reward obtained for each of the different reward functions is illustrated in Figure 8. The discrete reward function with reward shaping provided the highest reward followed by Zhu et al. [19], and the discrete reward function performed poorly. When interrogating the models that were able to load, it was found that only the discrete reward function with the PPO algorithm was able to grip the object. The discrete reward with reward shaping collided with the second arm and was closer to the object but never grasped it. No models with the Zhu et al. [19] reward function could be interrogated further.

The overall performance of each reward function is presented in Figure 10 as a function of the different DRL algorithms. Note that the discrete reward function with shaping was the most successful reward function for all three DRL algorithms considered, followed by Zhu et al. [19]. Both algorithms had a form of reward shaping which rewards the arm when it is close to the object as well, not only for performing certain tasks like the discrete reward function. The only difference between Zhu et al.'s [19] reward function and the one developed in this study is the way in which the function determines if the object has moved or not. Zhu et al. [19] uses the object position to assign rewards, whereas the reward function developed in this study determines if the arm has made contact with the object. This approach seems to result in more overall success.

These results have shown that the choice of reward function is critical to the probability of successfully completing the desired task. Moreso than the choice of DRL, as the PPO performed well for all reward functions.

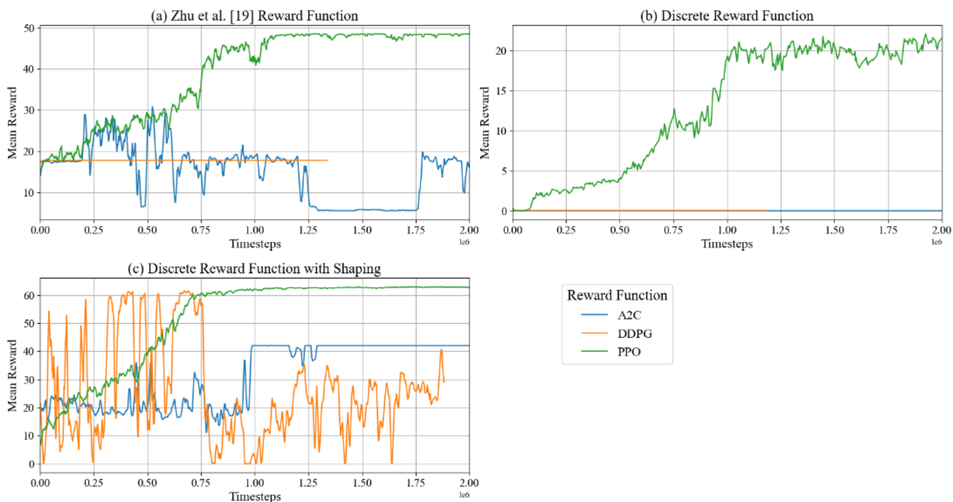


Fig. 10. Overall performance of each reward function as a function of the (a) A2C, (b) DDPG, and (c) PPO deep reinforcement learning algorithms.

3.3 Summary

The task is clearly more difficult than initially expected as there are a few intermediate goals to achieve, instead of only a simple goal for a single arm. For example, lifting an object is more likely to succeed, than lifting an object and moving it. With the addition of a second arm the task and learning difficulty exponentially increase. To improve the probability of successfully completing the task, the following could be considered:

- **Increase the timesteps:** It is possible that at some point the robot will learn the task, however, this is highly unlikely as we note a plateau in the results for all algorithms after a certain point.
- **Hyperparameter tuning:** The default parameters used in this study might not be the optimal parameters for this specific task. A sensitivity study on the hyperparameters could add value and increase the probability of successful learning.
- **Additional data:** Provide additional data to the learning process by means of vision-based sensors which will give the agent the advantage of knowing where the object is instead of randomly taking actions in the dark.
- **Rework the reward function:** Improve the definition of the reward function to explore more defined points or goals for incremental rewards.
- **Consider other DRL algorithms:** Perhaps including Hindsight Experience Replay (HER) which assigns small rewards if the agent has achieved or is close to the other goals.

It is possible that DRL is not the best approach to consider for an object handover task as there are few cases where it has been used in literature [6]. In fact, as the task is not continuous a model-free approach might not be sufficient. A model-based approach using unsupervised learning might be better suited where there is more control over the behaviour and a higher probability for success [6].

4 Conclusions

This study presented a deep reinforcement learning (DRL) approach for object handover between two collaborative robots. The task was simulated using OpenAI Gym and Pybullet with two Franka Emika Panda arms facing one another and an object between them. The aim of the task is for one robot arm to pick up the object, lift it past a height threshold before the second robot arm collects the objects and moves safely away.

DRL was considered as the learning mechanism to allow the cobot arms to learn the task at hand without any prior knowledge. The cobot arm is therefore rewarded, by means of a reward function, when it performs an action in line with achieving the task. Three DRL algorithms were investigated: Advantage Actor-Critic (A2C), Deep Determinisitic Policy Gradient (DDPG) and Proximal Policy Optimisation (PPO). Three approaches to defining the reward function were also considered: Zhu et al. [19] (existing implementation in literature) as well as two reward functions developed in this study namely, discrete and discrete with reward shaping.

The PPO algorithm showed the most promise and consistently improved its reward with an increase in timesteps. The DDPG algorithm performed poorly, unable to increase the initial zero reward, except for the discrete reward function with shaping.

The discrete reward function with shaping, developed in this study, provided the best overall performance with the highest rewards across all three DRL algorithms. The choice of reward function is critical to the probability of successfully completing the desired task.

The contribution of this study is to emphasise the importance of selecting the right DRL algorithm and reward function to facilitate successful learning, as well as the algorithmic development of a discrete reward function with reward shaping.

Future work will consider hyper parameter tuning to improve PPOs ability to successfully complete the object handover task. Additionally, improvements to the reward function can be made to explore more discrete goals for incremental rewards.

References

1. Sileo, M., Nigro, M., Bloisi, D.D., Pierri, F. 2021. Vision based robot-to-robot object handover, 20th International Conference on Advanced Robotics (ICAR), 6-10 December, Ljubljana, Slovenia, pp. 664-669.
2. Li, C., Zheng, P., Yin, Y., Wang, B., Wang, L. 2023. Deep reinforcement learning in smart manufacturing: A review and prospects, CIRP Journal of Manufacturing Science and Technology, 40, pp. 75-101.
3. El Zaatari, S., Marei, M., Li, W., Usman, Z. 2019. Cobot Programming for Collaborative Industrial Tasks: An Overview, Robotics and Autonomous Systems, 116, pp. 162-180.
4. Ortenzi, V., Cosgun, A., Pardi, T., Chan, W.P., Croft, E., Kulić, D. 2021. Object Handovers: A review for robotics, IEEE Transactions on Robotics, 37(6), pp. 1855-1873.
5. Gomes, N.M., Martins, F.N., Lima, J., Wörtche, H. 2022. Reinforcement Learning for Collaborative Robots Pick-and-Place Applications: A Case Study, Automation, 3, pp. 223-241.
6. Semeraro, F., Griffiths, A., Cangelosi, A. 2023. Human-robot collaboration and machine learning: A systematic review of recent research, Robotics and Computer-Integrated Manufacturing, 79, pp. 102432(1-16)
7. Costanzo, M., De Maria, G., Natale, C. 2021. Handover control for human-robot and robot-robot collaboration, Frontiers in Robotics and AI, 8, pp. 672995 (1-17).
8. Yang, X., Ji, Z., Wu, J., Lai, Y.-K. 2021. An open-source multi-goal reinforcement learning environment for robotic manipulation with Pybullet, Preprint arXiv:2105.05985v1. Code available on: https://github.com/IanYangChina/pybullet_multigoal_gym
9. Gallouédec, Q., Cazin, N., Dellandrea, E., and Chen, L. 2021. Panda-gym: Open-source goal-conditioned environments for robotic learning, 4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS. Code available at https://github.com/qgallouedec/panda-gym.
10. Franka Emika. 2023. The new FRANKA RESEARCH 3: The platform of choice for cutting edge AI & Robotics research. Available on: https://www.franka.de/research. Last accessed: 27 June 2023.
11. Stable Baselines3. 2022. Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations. Available on: https://stable-baselines3.readthedocs.io/en/master/index.html. Last accessed on 27 June 2023.
12. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning, Preprint arXiv:1602.01783v2.
13. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D. 2019. Continuous control with deep reinforcement learning, Preprint arXiv: 1509.02971v6.

14. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W. 2018. Hindsign Experience Replay, Preprint arXiv: 1707.01495v3.
15. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. 2013. Playing Atari with Deep Reinforcement Learning, Preprint arXiv: 1312.5602v1.
16. Schulman, J., Wolsi, F., Dhariwal, P., Radford, A., Klimov, O. 2017. Proximal Policy Optimization Algorithms, Preprint arXiv: 1707.06347v2.
17. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S. 2018. Soft Actor-Critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, Preprint arXiv: 1801.01290v2.
18. Fujimoto, S., van Hoof, H., Meger, D. 2018. Addressing function approximation error in actor-critic methods, Preprint arXiv: 1802.09477v3.
19. Zhu, Y., Wong, J., Madlekar, A., Martín- Martín, R., Joshi, A., Nasiriany, S., Zhu, Y. 2009. robosuite: A modular simulation framework for robot learning, Preprint arXiv:2009.12293. Code available on: https://github.com/ARISE-Initiative/robosuite