robotics

MDPI

Article

Simulated and Real Robotic Reach, Grasp, and Pick-and-Place Using Combined Reinforcement Learning and Traditional Controls

Andrew Lobbezoo * and Hyock-Ju Kwon

AI for Manufacturing Laboratory, Department of Mechanical and Mechatronics Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada
* Correspondence: ajlobbezoo@uwaterloo.ca

Abstract: The majority of robots in factories today are operated with conventional control strategies that require individual programming on a task-by-task basis, with no margin for error. As an alternative to the rudimentary operation planning and task-programming techniques, machine learning has shown significant promise for higher-level task planning, with the development of reinforcement learning (RL)-based control strategies. This paper reviews the implementation of combined traditional and RL control for simulated and real environments to validate the RL approach for standard industrial tasks such as reach, grasp, and pick-and-place. The goal of this research is to bring intelligence to robotic control so that robotic operations can be completed without precisely defining the environment, constraints, and the action plan. The results from this approach provide optimistic preliminary data on the application of RL to real-world robotics.

Keywords: reinforcement learning; proximal policy optimization; soft actor-critic; simulation environment; robot operating system; robotic control; Franka Panda robot; pick-and-place; real-world robotics

Citation: Lobbezoo, A.; Kwon, H.-J. Simulated and Real Robotic Reach, Grasp, and Pick-and-Place Using Combined Reinforcement Learning and Traditional Controls. *Robotics* **2023**, *12*, 12. https://doi.org/10.3390/robotics12010012

Academic Editors: Roman Mykhailyshyn and Ann Majewicz Fey

Received: 4 December 2022
Revised: 8 January 2023
Accepted: 9 January 2023
Published: 16 January 2023

1. Introduction

Over 50 years ago, the first electrically powered robot, the Stanford Arm, was built [1,2]. Surprisingly, the mechanics, control strategies, and applications of the Stanford Arm are similar to modern robots, such as the Panda research robot (Franka Emika, 2018). Both robots are electronically actuated and are implemented with conventional control requiring precise instructions.

1.1. Project Motivation

The difficulty with conventional control is that it requires individual programming on a task-by-task basis with no margin for error. These strategies rely on experienced technicians or robotics engineers sending commands on graphical or text-based programming interfaces to perform sequences of simple tasks [1–4]. Reinforcement learning (RL)-based control strategies have shown potential for replacing this manual approach [5–13]. In RL, agents are presented with a task, which they learn to solve by exploring various action sequences on internal simulated models of the environment, or in the real world [14]. Compared to other RL applications such as self-driving cars and video games [14–16], robotic control is difficult due to the high dimensionality of the problem and the continuous space of actions [17–19].

1.2. State of Research

To date, RL has been successfully applied to robotics for basic tasks such as target object reaching, grasping, placement, and basic manipulation [8,20–22] which gives some indication of its potential as a method for controlling robotic agents [23]. However, there

is room for further exploration and research for tasks with long action sequences (pick-and-place), there is a need for one-to-one comparisons between RL methods, and there is an absence of real-world testing to validate the applicability of RL outside of simulation. A detailed review on the current state of RL for robotic research can be found in Lobbezoo et al. [24], Mohammed and Chua [25], Liu et al. [11], and Tai et al. [6].

1.3. Objective

The principal objective of this research is to explore the application of RL to simulated and real-world robotic agents to develop a method for replacing high-level task programming. The objective has been broken down into the following subobjectives: (1) the development of a pipeline for training robotic agents in simulation, (2) the training of various models and the comparison of performance between each, and (3) testing of the RL control system in the real world.

1.4. Contribution

The novel contributions of this research to the field can be summarized as follows.

1. We developed a novel pipeline for combining traditional control with RL to validate the applicability of RL for end-to-end high-level robotic arm task planning and trajectory generation.
2. We modified and tuned the hyperparameters and networks of two existing RL frameworks to enable the completion of several standard robotics tasks without the use of manual control.
3. We completed validation testing in the real world to confirm the feasibility and potential of this approach for replacing manual task programming.

Other minor contributions include the following.

1. We created realistic simulation tasks for training and testing the application of RL for robotic control.
2. We completed direct comparisons between PPO and SAC to review the potential of each for task learning.

2. Materials and Methods

2.1. Simulation Methodology

To complete the simulation objectives, a physics engine was selected, custom tasks were designed, a codebase was implemented, and rewards were shaped according to the tasks.

2.1.1. Physics Engine

Three common environments for robotic representations are Gazebo, MuJoCo, and PyBullet as shown in Figure 1. Each package has strengths and weaknesses as evaluated and compared below.

Due to Gazebo's [26] functionality over a robot operating system (ROS), the communication between the control system and the simulated robot perfectly replicates the real-world communication. However, compared to MuJoCo and Pybullet, Gazebo has a higher computational cost on the GPU. Due to the requirement of parallelization of agents during training and GPU availability for network updates, Gazebo was rejected for this research.

MuJoCo is an intensive physics engine with the highest solver stability, consistency of results, accuracy of calculations, and energy conservation compared to other physics environments [27]. Due to the licensing issues (until 18 October 2021 [28]), difficulties with implementation, and the poor community support, MuJoCo was not selected for training.

PyBullet [29] is a Python-based environment, designed for rapid prototyping and testing of real-time physics. This environment is based on the Python Bullet Physics engine. PyBullet does not have prebuilt ROS communication; however, custom ROS nodes can be

written to allow for ROS integration. Due to the ease of modification and implementation of the PyBullet environment, in addition to the simplicity of parallelization for training [29], PyBullet was selected as the primary environment for training the RL agent.

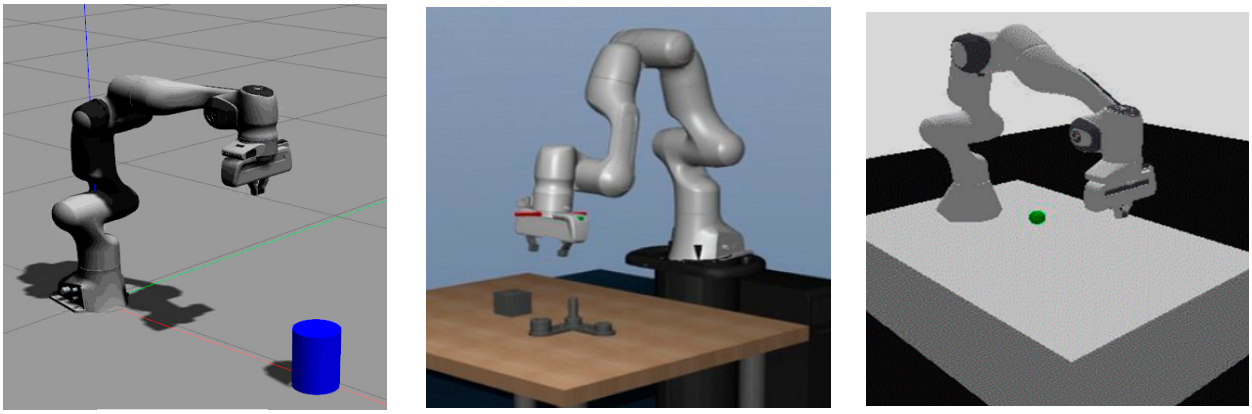
2.1.2. Framework and Custom Tasks

The simulation framework implemented for this project was the Gym API (0.19.0), developed by OpenAI Inc. The base PyBullet (3.2.1) panda model was cloned from the Github repository created by Gallouedec, et al. [30] and modified to suit this application.

The Gym-RL learning process is broken down into a series of episodes. The episodes are limited to 50–100 timesteps, to ensure that the agent focuses its exploration in the vicinity of the target. During each timestep in the environment, the agent can move for 1/240 s in the simulation. For the robotic RL framework, each episode begins with the agent initialized in a standardized home position, with the target object instantiated in front of the agent with a random position. The agent must learn to relate the input state information from the environment to the ideal action command based on the episodic learning cycles. If the task is completed before the maximum number of steps is reached, the episode is terminated early and the reward per episode is improved.

To make custom PyBullet environments inside of the Gym-PyBullet model, several interacting features of the model were modified. The main modifications of the base environment included reward shaping, target object block instantiation (for pick-and-place), episodic termination, and modifications to friction coefficients and maximum joint forces.

The three custom tasks created for testing the RL robotic system are Panda reach, Panda grasp, and Panda pick-and-place. The task environments can be viewed in Figure 2 for each task, end effector-based control strategies with prebuilt IK packages were implemented. As the goal of this project was to learn high-level task planning strategies, end effector-based control (as an alternative to joint-based control) was adopted to reduce the difficulty in training.



(**a**) (**b**) (**c**)

Figure 1. Simulation environments. (**a**) Gazebo, (**b**) MuJoCo [31], (**c**) Pybullet.

For these tasks, vector-based position feedback was applied. The agent was fed some combination of state positions, including the gripper (x, y, z, Vx, Vy, Vz, pitch, roll, yaw), the target object/objects (x, y, z, Vx, Vy, Vz, pitch, roll, yaw), and in the case of pick-and-place, the target block (x, y, z, Vx, Vy, Vz, pitch, roll, yaw) positions. The training operation involved the agent learning to provide xyzg (g being gripper) input action commands to the robot based on the vector of concatenated positions provided to the agent.

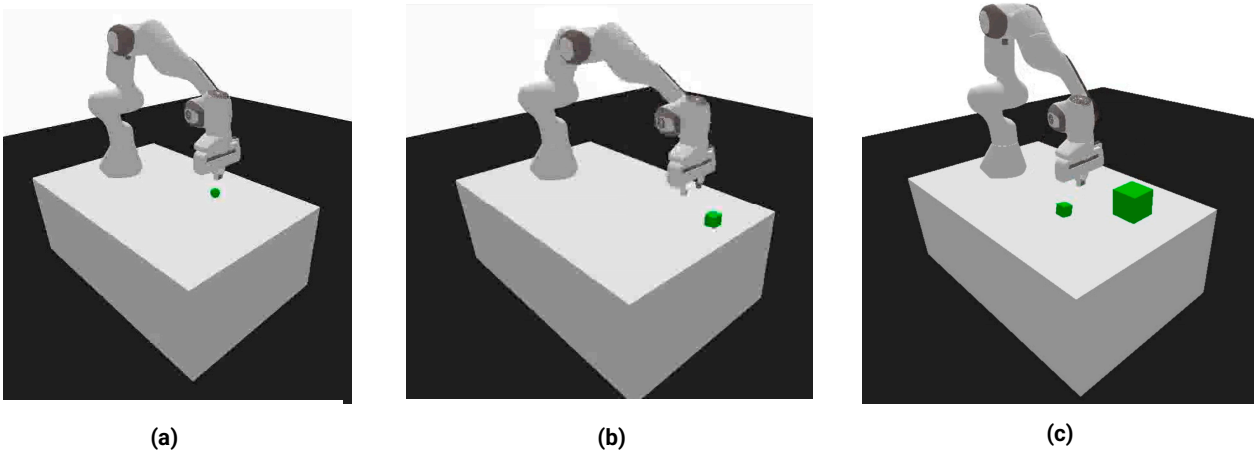


Figure 2. Various Panda environment configurations. (**a**) Closed gripper with end-effector control [x, y, z]. Target stationary and penetrable. (**b**) Open gripper (gripper width W) with end-effector control [x, y, z, W]. Target object is dynamic and impenetrable (**c**) Open gripper (gripper width W), end-effector control [x, y, z, W]. Target object is dynamic and impenetrable. Placement block on right.

Task difficulty progressively increases, with the first task, reach, being the simplest, and the third task, pick-and-place, being the most difficult. As shown in Figure 2, the first task, reach, only requires the control of the end effector (EE) XYZ position. The reach target object is stationary and penetrable, so the gripper cannot cause changes in the target object position with collisions. The second task, grasp, requires the agent to control the XYZ of the EE, as well as the gripper width (G). The task complexity increases because the target object is not stationary and is impenetrable, so any collisions between the EE and the target block will cause the block to move and/or slide off the table. For this task, the agent must learn to approach the block by following specific paths. The third task, pick-and-place, requires the agent to actuate the gripper similarly to grasp. The task complexity for pick-and-place is significantly higher than for grasp, as the agent must learn the grasp, lift and transportation action sequences. Additionally, for the pick-and-place task, the agent must learn to avoid collisions with the large placement block.

2.1.3. RL Algorithms

The two algorithms tested and compared for this research were soft actor–critic (SAC) and proximal policy optimization (PPO). SAC was selected due to its sample efficiency for complex problems, and PPO was selected due to its hyperparameter insensitivity and stable convergence characteristics. Table 1 compares some of the key characteristics of these two methods.

Table 1. Comparison of SAC and PPO.

	PPO	**SAC**
Policy Type	On-Policy	Off-Policy
Optimization method	Policy Optimization	Q-Learning and Policy Optimization
Update stability	High	Low
Hyperparameter sensitivity	Low	High
Sample efficiency	Low	High

Proximal Policy Optimization

PPO is a policy gradient technique which was designed to provide faster policy updates then previously developed RL algorithms such as the advantage actor–critic (A2C) or deterministic policy gradient (DPG). PPO applies the DPG structure, but updates the policy parameter θ based on a simple surrogate objective function [32].

PPO is designed as an improvement to trust region policy optimization (TRPO). TRPO optimizes the return of the policy in the infinite-horizon MDP by implementing the loss function shown below [32],

$$Maximize_\theta E_s\left[\frac{\pi_\theta(a|s)}{\pi_{\theta old}(a|s)}\hat{A}\right] \tag{1}$$

where the policy parameter θ is maximized based on the product of the ratio of new and old policies and the advantage function $\hat{A}$. TRPO constrains the updates to the policy parameter θ through the introduction of the KL divergence constraint. The trust region constraint can be expressed as shown in the following equation [32,33],

$$E_t[KL(\pi_{\theta old}(\cdot|s), \pi_\theta(\cdot|s))] \leq \delta \tag{2}$$

where δ is the size of the constraint region. This constraint limits the difference between the new policy and the old policy to prevent large, unstable updates.

The final loss function for TRPO can be expressed as shown [33],

$$Maximize_\theta E\left[\frac{\pi_\theta(a|s)}{\pi_{\theta old}(a|s)}\hat{A} - \beta KL[\pi_{\theta old}(\cdot|s), \pi_\theta(\cdot|s)]\right] \tag{3}$$

where β is a fixed penalty coefficient. TRPO is overly complicated to solve, as it requires a conjugating gradient method. PPO has an advantage over the TRPO technique because it is simpler to solve, due to the reduction of the region constraint to a penalty in the loss function.

PPO introduces a clipped surrogate objective function, which penalizes any changes that move the ratio of the new and old policies away from 1. The object function is [33]

$$L^{clip}(\theta) = \hat{E}\left[\min\left(r(\theta)\hat{A},\ clip(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}\right)\right] \tag{4}$$

where ϵ is the clipping hyperparameter (0.1–0.2). By using this objective function, the action will be clipped in the interval $[1-\epsilon, 1+\epsilon]$ [34].

PPO is a stable training technique as it constantly learns the policy in an on-policy way through continuous exploration without the use of a replay buffer. The main disadvantage of PPO is the low sample efficiency, and the convergence to a single deterministic policy. An alternative to PPO is the SAC technique.

Soft Actor–Critic

SAC is an off-policy actor–critic method, founded on advantage actor–critic (A2C). SAC was selected over A2C and DPG approaches due to its effectiveness balancing the exploration–exploitation tradeoff, and its ease of parallelization. SAC balances the exploration–exploitation tradeoff with entropy regularization, which encourages the agent to explore based on the "temperature" (uncertainty) at a given time step. The formulation for the entropy is

$$\mathrm{H}(x) = \mathrm{E}[-\log(\pi(*|x))], \tag{5}$$

where π is the probability density function for the policy, and x is a random variable representing the state.

SAC works by learning the policy and two value functions simultaneously. The policy formulation is

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{\infty}\left[\gamma R(s, a, s') + \alpha H(s)\right], \tag{6}$$

where $H(\,)$ is the entropy of the policy at a given timestep, π^* is the optimal policy, γ is the discount rate (time dependent), and α is the entropy regularization coefficient [35,36]. Here,

the entropy serves as a reward for the agent at each time step to encourage or discouraged exploration. The formulation of the value functions V^* and Q^* are

$$Q^{\pi^*}(s,a) = r(s,a) + E_{s'}\left[V'^*(s')\right] \tag{7}$$

$$V^{\pi^*}(s) = E_{a\sim\pi}\left[Q^{\pi^*}(s,a) - \alpha\,\mathrm{H}(s)\right] \tag{8}$$

where α is a dual variable (fixed and varying) and r is the reward given a state action pairing [35,37]. In the case where alpha is varying, α is formulated as

$$\alpha \leftarrow \alpha + \lambda\, E[log(\pi^*(\mathrm{a}|s) + H(s)] \tag{9}$$

where λ is the learning rate.

2.1.4. The RL Training Codebase

The primary RL codebase implemented for this project was Stable Baselines 3 (SB3) [38]. SB3 (1.4.0) was selected as the primary codebase for testing PPO and SAC, because it can be easily modified to accommodate the custom Panda environment, and has prebuilt parallelization, visualization, and GPU integration features. Additionally, SB3 has excellent supporting documentation, many functioning examples, and is built on PyTorch.

Table 2 compares the primary open-source codebases implemented by the RL community. The table indicates that SB3 has a slight advantage over RLlib due to the additional documentation, and consistent PyTorch backbone. Tianshou was also considered but rejected due to the lack of documentation, a small user base and limited tutorials.

Table 2. Comparison of RL codebases.

	SB3	**RLlib**	**Tianshou**
Backbone	PyTorch	PyTorch/TF	PyTorch
Documentation	Excellent (15 pages)	Excellent (11 pages)	Good (6 pages)
Number of codebase tutorials and worked examples	12	24	7
Last commit	<1 week	<1 week	<2 weeks
Pretrained models	Yes	Yes	No

2.1.5. Optuna

Due to the complex nature of RL and robotic control, hyperparameter tuning is crucial. RL has all the same hyperparameters as required in supervised learning such as number of epochs, batch size, choice of activation function, learning rate, and optimization algorithm. Additionally, RL problems have a range of hyperparameters not required in supervised learning, such as number of steps (time between updates), gamma (discount factor), and entropy coefficient (confidence parameter, encourages exploration). During training, it was noticed that network parameters played a significant role in the speed and stability of convergence, so the number of hidden layers and the neural network width were treated as hyperparameters to be optimized.

For RL problems, grid or random search would be unreasonable because >1 million hyperparameter combinations would be required to achieve solutions close to optimal. The intelligent hyperparameter search algorithm applied for this project was the Gaussian process-based tree-structured Parzen estimator (TSPE) [39]. For the Panda robot, the TSPE objective function contained the "reward" metric, which was maximized during training [40]. To improve learning speed, the median-tuner pruning technique was applied. The pruner was set to start pruning after completing 1/3 of the steps for each hyperparameter trial.

After each Optuna trial was completed, the results of the trial were viewed in the parallel coordinate plot (PCP) and the hyperparameter importance plot (HIP). PCPs were

implemented as a tool for comparing hyperparameters, to learn how specific hyperparameter ranges affected training accuracy. HIPs were implemented to determine which hyperparameters caused the most significant impact on training results. Figure 3 depicts an example of PCP and HIP plots from an Optuna trial. Note that the column values have different number formats for different hyperparameters, i.e., for some columns, the range is an integer (i.e., number of epochs), some are floating points (i.e., gamma, learning rate, etc.) and some are integers representing powers (i.e., batch size, ranging from 128 (2^7) to 512 (2^9)).

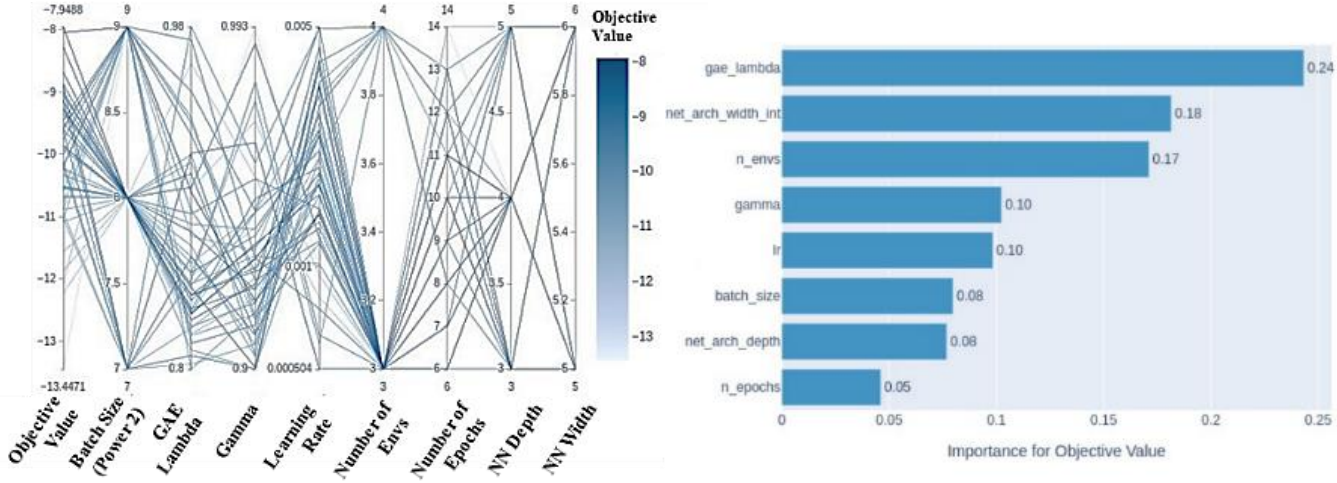


Figure 3. An example of parallel coordinate plot (PCP) and hyperparameter importance plot (HIP) from the training of PPO for vector-based Panda grasp with dense rewards.

The PCP is a useful tool when comparing the performance of specific ranges of hyperparameters, and the relationships between them. From the PCP shown in Figure 3, it is clear that GAE lambda in the range of 0.8–0.9 with gamma values in the range of 0.9–0.95 and learning rates in the range of 0.001–0.005 perform best for PPO for vector-based Panda-grasp with dense rewards.

For some trials, the relationship between the objective value and specific hyperparameters shown in the PCP were not clear. For such cases, HIPs were implemented to determine if the hyperparameters of concern had a significant impact on training. Hyperparameters with little impact on training were fixed once a realistic value for the hyperparameter was found from literature or from hyperparameter tuning. The results section presents figures of PCPs. Through the use of HIPs, values which had little effect on training performance were identified and removed during early rounds of hyperparameter tuning.

2.1.6. Reward Structure

Reward-shaping plays a critical role in solving RL problems. The two reward structures implemented for this project were dense (heterogeneous) and sparse (homogeneous) rewards, shown in Figure 4.

With the standard sparse reward scheme, the agent received a reward of -1 for all states except the final placement state. The agent had difficulty solving complex RL problems with this reward scheme due to the Monte Carlo (random) nature of this approach. The dense reward function implemented for this project was the heterogeneous reinforcement function [41]. In this approach, the reward improved as the agent executes the task correctly. For example, for reach, the reward improves proportionally to the distance between the EE and target sphere.

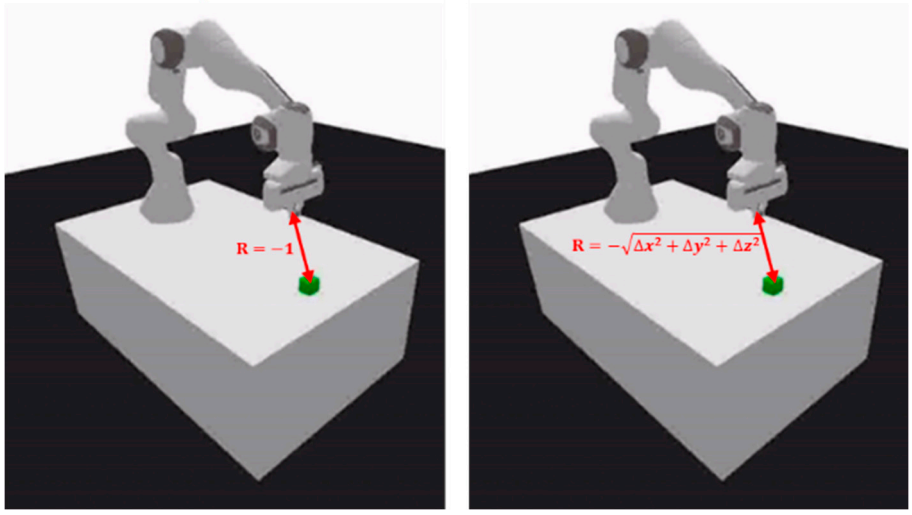


Figure 4. Sparse (**left**) vs dense (**right**) reward shaping.

For this project, dense and sparse rewards were compared for reach and grasp. The pick-and-place task required the agent to pick up, transport, and release the target block at the correct location. For this task, a standard sparse reward function was too difficult to solve, so only the results for the dense reward scheme are shown.

Hierarchical RL was considered for this project; however, to constrain project scope, only standard sparse and dense rewards were compared. Alternative research [21,42] investigates the use of hierarchical RL for similar applications.

2.2. Experiment Design and Robotic Control

The robot implemented for this project is the Panda Research Robot developed by Franka Emika. The Panda robot was purchased as a packaged system which contained the arm, the gripper, the control box, communication controller, joint lock, and a kill switch, which all can be viewed in Figure 5. Additional widgets added include the Azure Kinect (for remote viewing and control) and the workstation display monitor. The Panda system was selected for the project, as it is a research tool which allows for quick implementation and testing for various control strategies [43].

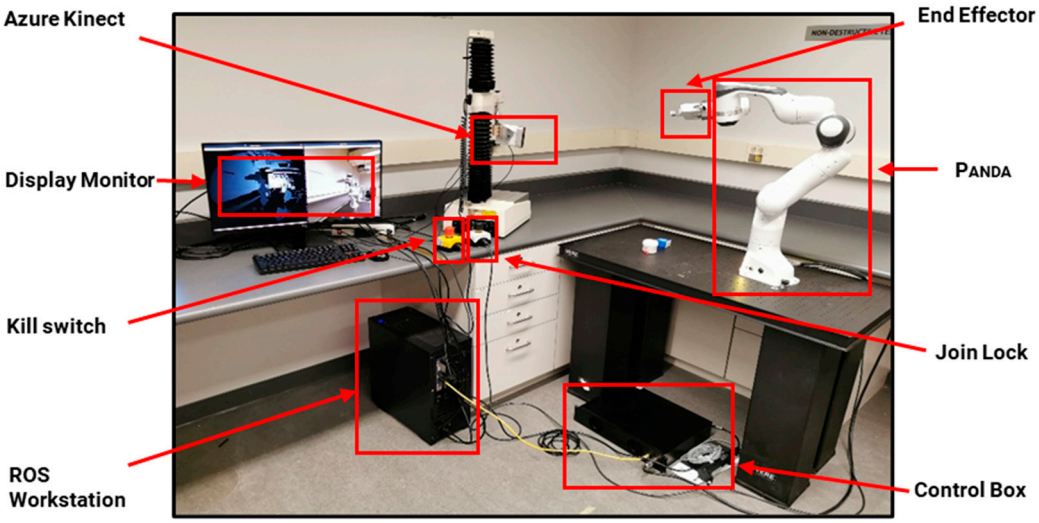


Figure 5. Panda robot lab setup.

2.2.1. Panda Robot

The Panda robot consists of the arm: a kinematic chain with seven articulated joints, and a gripper end effector. A kill switch is included for operator safety, and the joint lock protects the joints from being back driven while the robot is not in use. Each joint implements high-precision encoders and torque sensors which enable the robot to have a pose repeatability of 0.1 mm and a force resolution of 0.05 N. With a maximum gripping force of 140 N, the robot can support a payload of 3 kg [43], making it an ideal choice for tasks such as pick-and-place.

The Panda was procured alongside the control unit for the device, which is a part of the Franka control interface (FCI). The FCI is the interface for controlling the motion of the robotic arm from a local workstation via an ethernet connection. The FCI interface allows for bidirectional communication between the agent and the workstation for positional readings (joint measurements, desired joint goals, external torques, collision information), and commands (desired torque, joint position, or velocity, cartesian pose or velocity, and task commands). The communication framework for the FCI can be viewed in Figure 6.

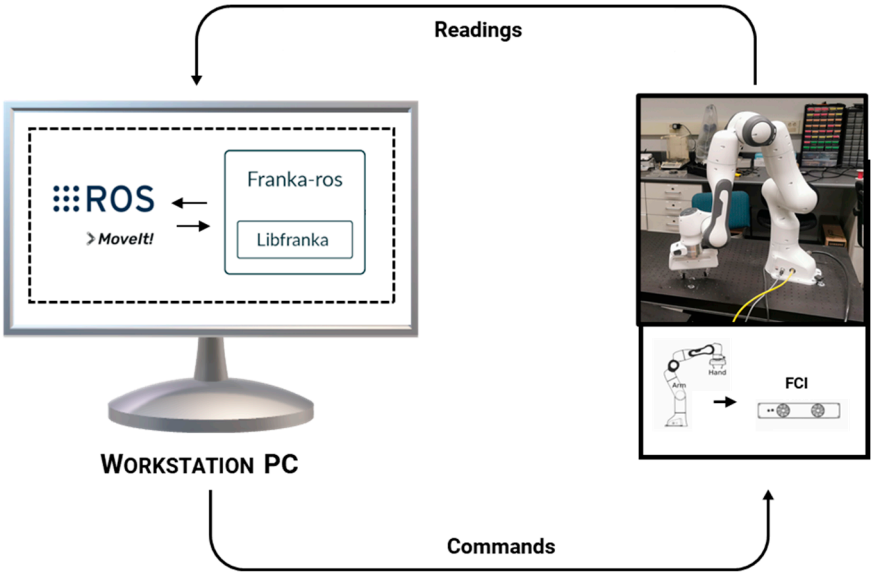


Figure 6. The FCI communication framework.

The FCI allows for 1 kHz signals to be communicated between the workstation PC and the Panda robot. To enable the use of this high-frequency signal communication, the Ubuntu workstation requires a real-time Linux Kernel (5.14.2-rt21).

2.2.2. Control Interfaces

The control interface implemented for the Panda is Franka-ROS(noetic). Franka-ROS is a package which communicates with the FCI via libfranka, a prebuilt C++ network communication package.

Libfranka(0.8.0) is a package used for basic non-real-time and real-time functions such as setting controller parameters, processing feedback signals, reading robot states, generating basic motion paths, and sending torque commands. Franka-ROS was implemented to wrap libfranka to allow for integration of the FCI into the ROS environment (Figure 6). Once FCI was integrated inside the ROS environment, other ROS packages such as MoveIt were applied.

MoveIt(0.1.0) is an open-source library designed for implementing advanced motion planning, kinematics, control, and navigation strategies [44]. The MoveIt control interface is more advanced than most other planners, as it incorporates a collision-detection pipeline prior to allowing action execution.

2.2.3. Control Implementation

The framework for controlling the Panda robot is presented in Figure 7. The explanation for the control cycle is as follows. The Panda sends the joint state information over a ROS node to PyBullet. PyBullet receives the joint information and instantiates the simulated Panda in the associated position. After the PyBullet environment is created, the Panda state information is fed into the RL agent, and the agent outputs an action command (XYZG). The joint states required to follow the particular action command are calculated inside of PyBullet with the use of the inverse kinematic package (Samuel Buss Inverse Kinematics Library) [45]. After the joint trajectories are calculated, the action is executed in the PyBullet environment, and the joint positions are published to the MoveIt framework. MoveIt accepts the joint positions and executes the action after checking the safety of the action with the collision-detection pipeline. Once the action is completed in the real world, and the position of the end effector is within the accuracy threshold, the process repeats. The agent iteratively steps through this control cycle, until the task is completed in the simulated world.

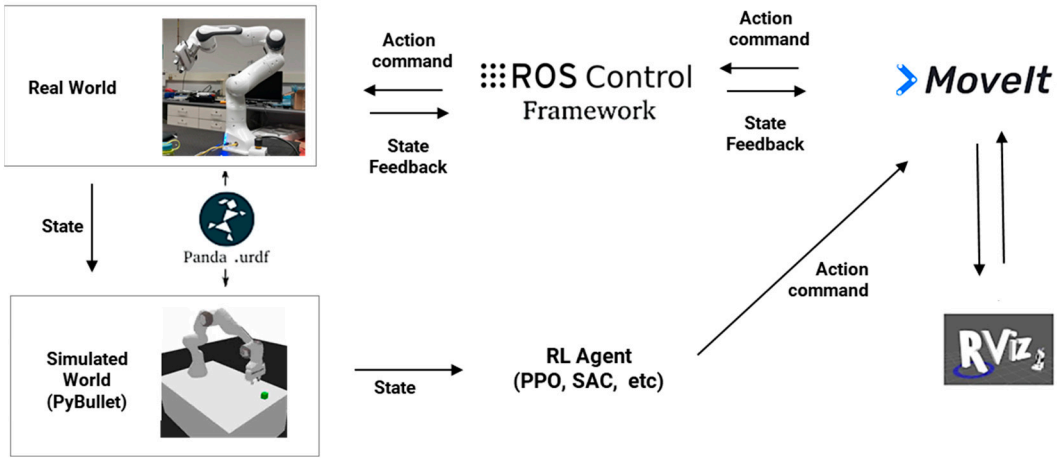


Figure 7. Framework for control communication between the PyBullet simulator, the real-world robot, the ROS system and the RL agent.

Both simulation and real-world testing were completed for this project. Section 3.1 depicts the results from hyperparameter tuning and simulation training. Section 3.2 shows the accuracy of real-world testing after simulation training.

3. Results and Discussion

3.1. Simulation Results

The section below presents the simulation analysis completed for training the robotic agent. Parallel coordinate plots were implemented as a tool for hyperparameter tuning for each environment. After tuning, convergence plots were implemented to depict the success of each tuned model.

Task complexity had a major impact on network design and hyperparameter values. A range of hyperparameter combinations could be found quickly for the simple reach and grasp tasks. For the pick-and-place task, extensive tuning (100 + trials) was required.

3.1.1. Panda Reach and Panda Grasp with Dense Rewards

This section reviews the results for the Panda reach and grasp tasks with dense reward functions. The grasp problem was more complex than the reach problem because the EE had to learn to grasp the target while not bumping it off the table. The reward structure for this problem is dense. Every time the agent takes a step in a direction, the reward is increased or decreased based on the relative distance between the end effector and the target.

Several sets of hyperparameter studies were completed for these tasks. The reach task was relatively simple to solve and required minimal hyperparameter tuning. The grasp task had higher complexity and required several studies. The grasp hyperparameters performed well when applied to the reach task, so the PCP for the grasp task is presented in Figure 8.

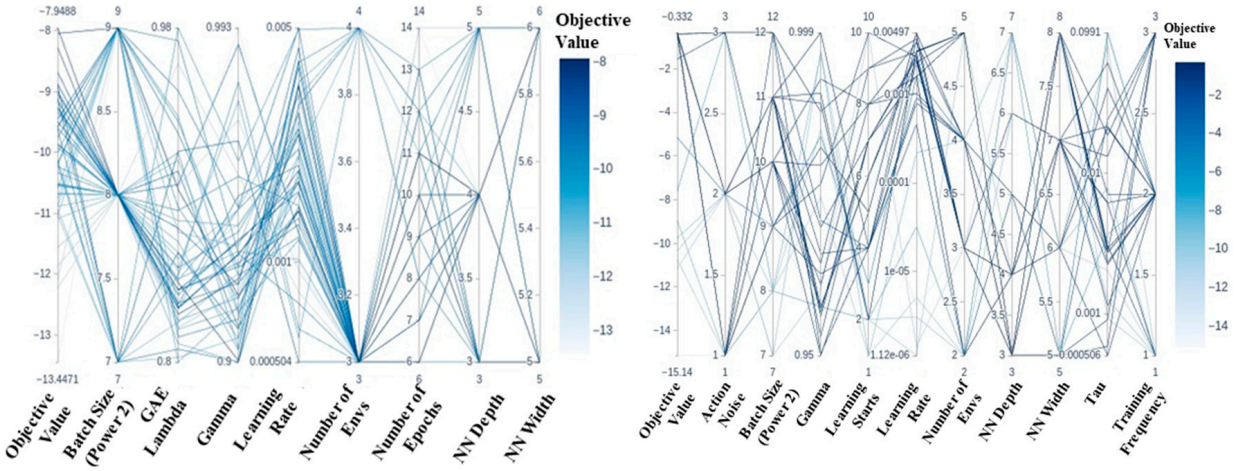


Figure 8. Left: PCP for PPO Panda grasp with dense rewards. Agent receives the highest objective value with a batch size of 2^8 and three environments. **Right**: PCP for SAC Panda grasp with dense rewards. Agent receives the highest objective value with a batch size of 2^8 or 2^{10} and a learning rate of ~0.0075.

As shown in Figure 9, Panda reach was trained efficiently for both PPO and SAC. The PPO model took 2.9 million-time steps to converge to a reach accuracy greater than −0.75. The SAC model converged very quickly, breaking the average reward of −0.75 after only 0.04 million steps. With this reward range, both the PPO and SAC agents had a 100% success rate on the reach task.

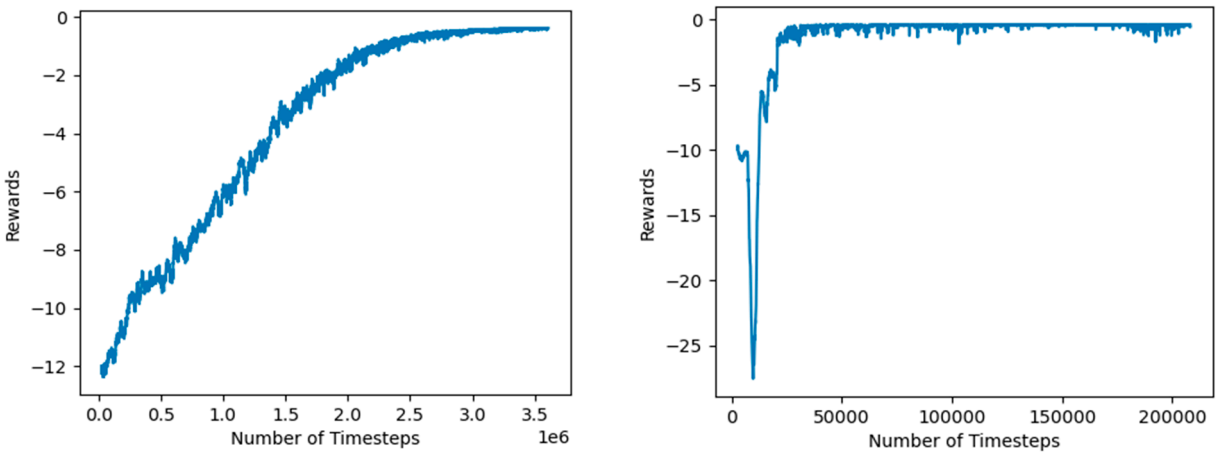


Figure 9. Left: PPO convergence for Panda reach with dense rewards. **Right**: SAC convergence for Panda reach with dense rewards.

As shown in Figure 10. Panda grasp was trained efficiently for both PPO and SAC. The PPO model took 4.0 million-time steps to converge to an average accuracy greater than −0.5. The SAC model converged very quickly, breaking the average reward of −0.5 after only 0.08 million steps. With these rewards, PPO was able to successfully complete the task for 89% of the attempts, compared to the SAC agent which was able to complete the task for 92% of the attempts.

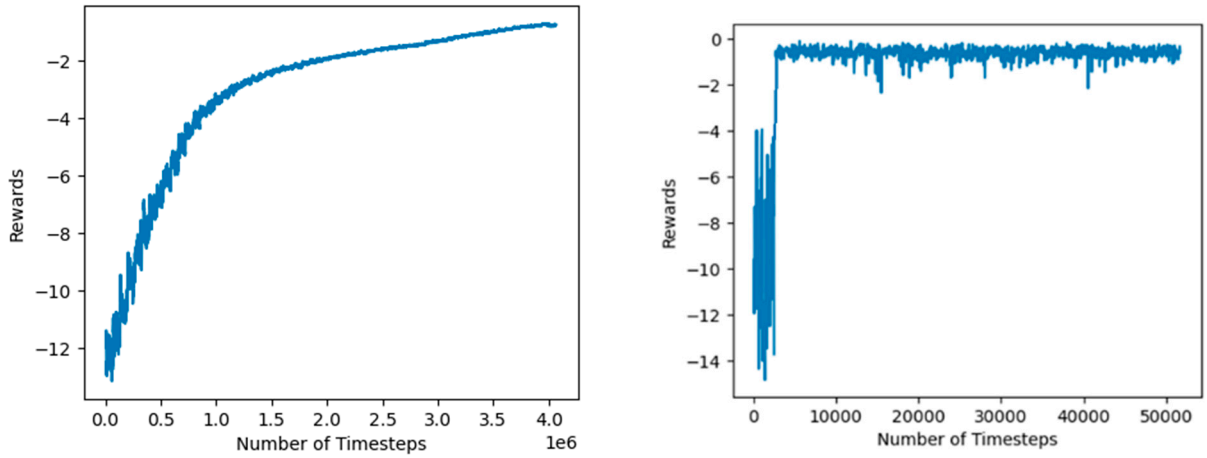


Figure 10. Left: PPO convergence for Panda grasp with dense rewards. **Right**: SAC convergence for Panda grasp with dense rewards.

3.1.2. Panda Reach and Panda Grasp with Sparse Rewards

This section reviews the results for the Panda reach and grasp tasks with sparse reward functions. Like Section 3.1.1, this problem required the Panda reach agent to learn the relationship between the reward and the EE XYZ coordinates, and the target XYZ coordinates. The difference between this Section, and Section 3.1.1 is the reward the agent receives during exploration. For the sparse reward scheme, the agent receives a reward of -1 after each step, unless the agent reaches the target position and correctly completes the task.

Several sets of hyperparameter studies were completed for these tasks (Figure 11). Compared to the same tasks with sparse rewards, the networks for these tasks had to be deeper and wider. The grasp hyperparameters performed well when applied to the reach task, so only one set of hyperparameter tuning was required.

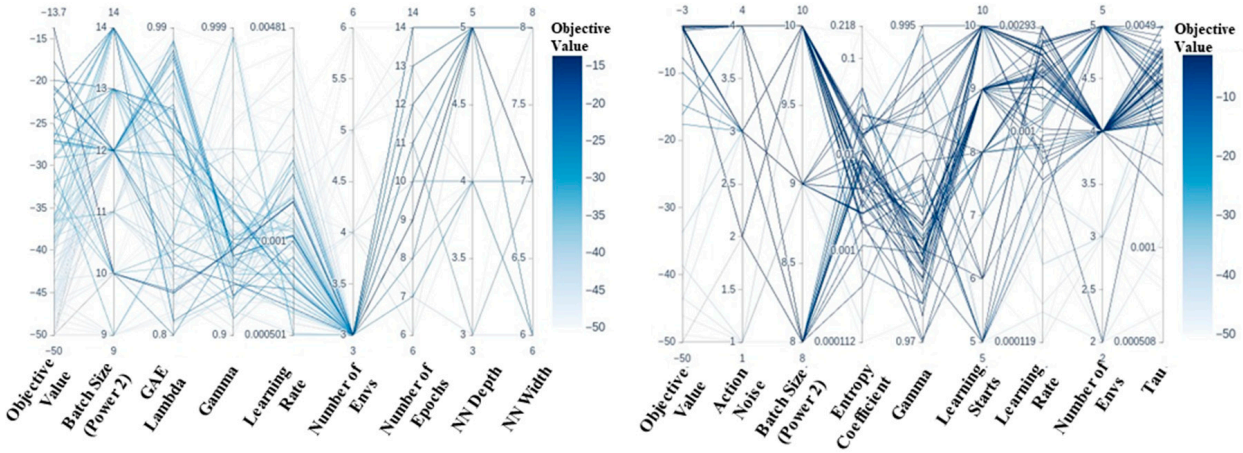


Figure 11. Left: PCP for PPO Panda grasp with sparse rewards. Agent receives the highest objective value with a learning rate ~0001, two environments, and a network depth of five layers. **Right**: PCP for SAC Panda grasp with sparse rewards. Agent receives the highest objective value with an entropy coefficient ~0.002, batch size of 2^{10}, learning rate ~0.0015, and gamma of ~0.96.

As shown in Figure 12, Panda reach was trained efficiently for both PPO and SAC. The PPO model took 5.6 million-time steps to converge to an accuracy greater than -2.75. The SAC model converged quickly, breaking the average reward of -1.9 after only 0.16 million steps. With this reward range, PPO and SAC agents had 100% success rates.

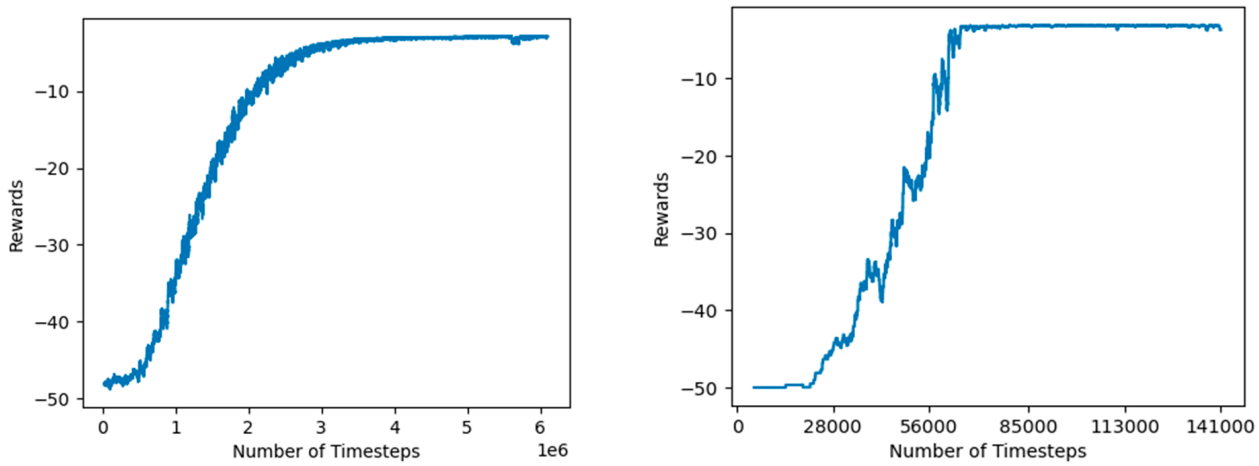


Figure 12. Left: PPO convergence for Panda reach with sparse rewards. **Right**: SAC convergence for Panda reach with sparse rewards.

As can be seen in Figure 13, Panda grasp was trained efficiently for both PPO and SAC. The PPO model took 5.5 million-time steps to converge to an average accuracy greater than −2.75. The SAC model converged quickly, breaking the average reward of −3.1 after only 0.14 million steps. With this reward range, the PPO and SAC agents were able to complete the task with 90% and 95% success rates, respectively.

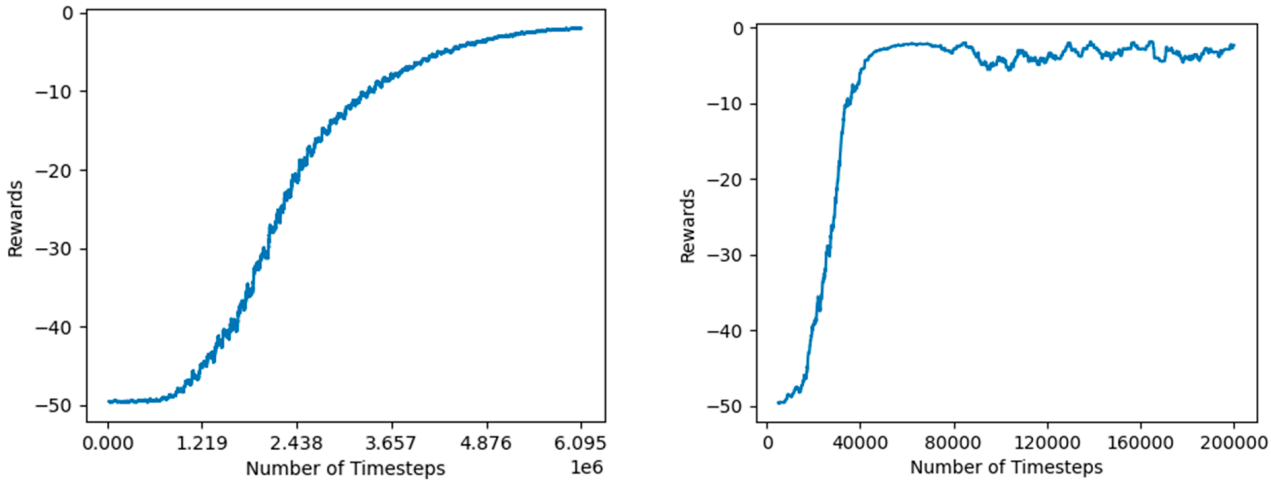


Figure 13. Left: PPO convergence for Panda grasp with sparse rewards. **Right**: SAC convergence for Panda grasp with sparse rewards.

The contrast between sparse and the dense rewards can be understood by comparing this section with Section 3.1.1. With sparse reward functions, the agent required approximately twice as many training steps, because additional exploration was required to find the states resulting in positive rewards.

The agent consistently received a high negative reward with the dense scheme (range of −1.5 to −3), because the reward is −1 per step rather than being based on position. For the reach task, both sparse and dense rewards schemes resulted in the agent completing the task with a 100% success rate. For the grasp task, agents trained with sparse rewards outperformed agents trained with dense rewards by an average of 2%.

These results indicate that for problems with minimal difficulty, a simple, sparse, deterministic reward signal is most effective. The main disadvantage of the sparse reward

scheme is that significant exploration and clock-time are required for the policy to obtain convergence.

3.1.3. Panda Pick-and-Place with Dense Rewards

This section reviews the results for the Panda pick-and-place task with dense rewards. For the Panda pick-and-place problem, the agent had to learn the relationship between the reward and the end effector XYZ coordinates, the target XYZ coordinates, and the placement location XYZ position. Of all the tasks tested, pick-and-place had the highest complexity. A dense reward function was implemented, because sparse rewards make the task intractable.

Several sets of hyperparameter studies were completed for this task. The results from hyperparameter tuning can be seen in the PCP's shown in Figure 14. Due to the problem complexity, the training algorithm required high-entropy coefficients and a high action noise to increase the exploration of the solution space. Deeper networks with 5-7 layers were required to solve this problem optimally.

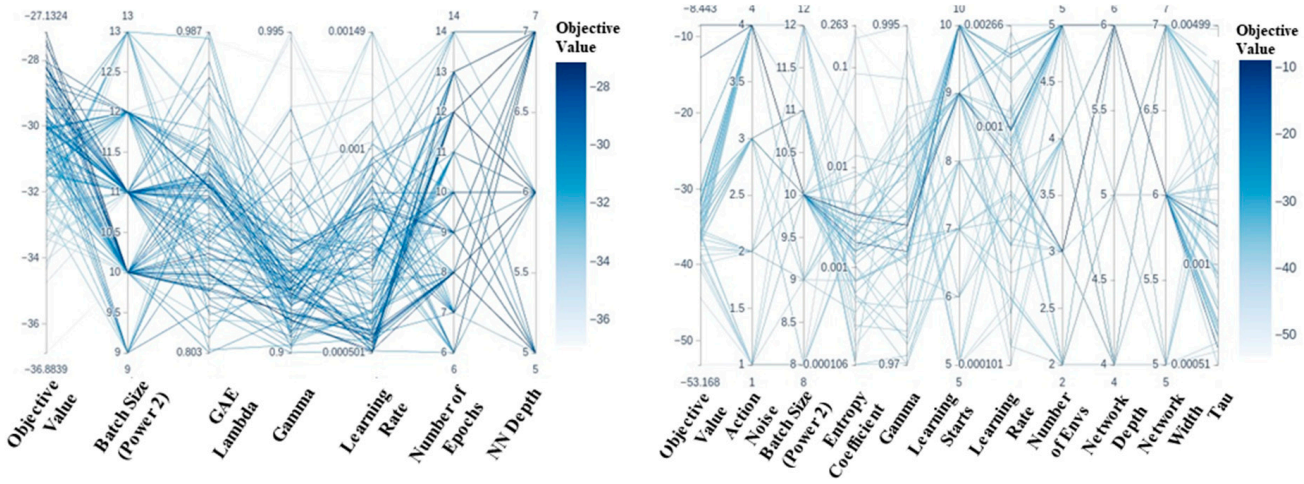


Figure 14. Left: Parallel coordinate plot (PCP) for PPO Panda pick-and-place with dense rewards. Agent receives the highest objective value with batch sizes of 2^{10} or 2^{11}, learning rate of ~0.00075, and a network depth of seven layers. The batch size is shown. **Right**: Parallel coordinate plot (PCP) for SAC panda pick-and-place with dense rewards. Agent receives the highest objective value with batch sizes of 2^{10}, 10 learning starts, and five environments.

As can be seen in Figure 15, Panda pick-and-place was trained efficiently with both PPO and SAC. The PPO model took 8.3 million-time steps to converge to a reach reward greater than -7.0 and the SAC model took 0.47 million-time steps to converge to a reach reward greater than -7.0. With this reward range, the PPO and SAC agents could complete the task with 85% and 71% success rates, respectively.

The use of sparse rewards means that there are many suboptimal solutions for this problem. One suboptimal solution involves pushing the target object toward the target position while keeping the block on the table. This solution increases the reward, because the distance between the target object and the target position is decreased; however, this solution does not involve grasping and lifting the block to complete the task. Both the SAC and PPO solutions have a rolling reward convergence, due to the agent first learning these suboptimal solutions.

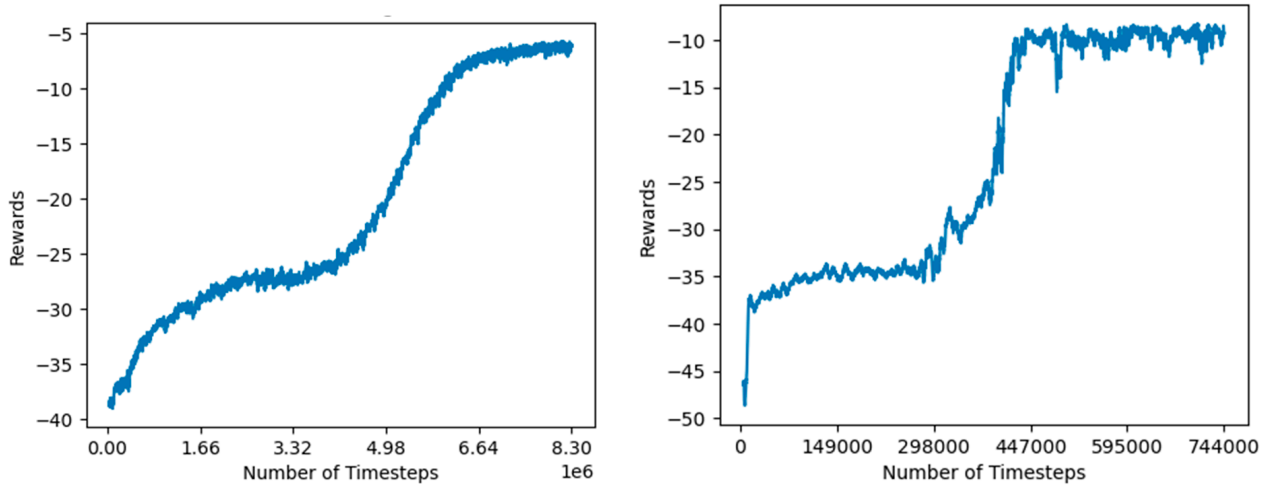


Figure 15. Left: PPO convergence for Panda pick-and-place with dense rewards. **Right**: SAC convergence for Panda pick-and-place with dense rewards.

3.1.4. Summary

Given sufficient hyperparameter tuning and reward shaping, both PPO and SAC agents were able to effectively learn optimal control policies for reach, grasp, and pick-and-place. For relatively simple reach and grasp tasks, sparse and dense rewards performed well. The main consequence of using sparse rewards is the extensive hyperparameter tuning required and the increase in training time. Due to the task complexity for pick-and-place, the dense reward scheme was implemented. After extensive training, the agent was able to complete the task with an average accuracy of 78%. Table 3 shows a summary of all the simulation results. The results for pick-and-place were noticeably lower than some of the results from the literature. The primary rationale for this difference is the obstacle avoidance which must be learned in this environment. The pick-and-place task simulated here does not only require the agent to move the target block to a position in space, but also requires the agent to avoid the placement block while completing this motion. Most of the failures noticed in simulation were due to interference between the gripper and the large target block. Although this change makes the task significantly more difficult, the results are more realistic for application in the real world.

Table 3. RL performance on simulated tasks.

Simulated Problem	Reward	Positional Feedback Method	RL Implementation	Task Success Rate (%)
Panda reach	Dense	Vector	PPO	100
Panda reach	Dense	Vector	SAC	100
Panda reach	Sparse	Vector	PPO	100
Panda reach	Sparse	Vector	SAC	100
Panda grasp	Dense	Vector	PPO	89
Panda grasp	Dense	Vector	SAC	92
Panda grasp	Sparse	Vector	PPO	90
Panda grasp	Sparse	Vector	SAC	95
Panda pick-and-place	Dense	Vector	PPO	85
Panda pick-and-place	Dense	Vector	SAC	71

The comparison of PPO and SAC reveals several patterns in training time, performance, and convergence. For all problems, SAC required a minimum of one order of magnitude fewer steps than PPO to obtain convergence. The SAC advantage in training time stands in stark contrast to SACs convergence difficulties. SAC was highly hyperparameter-sensitive compared to PPO, which resulted in additional time being spent to determine the

ideal hyperparameters for each task. For the simple reach and grasp problems SAC converged to a more optimal solution than PPO; however, for the more difficult pick-and-place problem, PPO significantly outperformed SAC.

The results from the comparison of SAC and PPO are intuitive. SAC implements entropy maximization and off-policy training to reduce training time. The consequence of these training principles is that SAC is sample-efficient but tends to converge to a local optimum. PPO maintained slow, consistent convergence and SAC tended to diverge when overtrained.

3.2. Real-World RL

After all simulated RL tasks were developed, tuned, and trained, the networks were tested in the real world. For each implementation, 10 tests were completed to approximate the real-world testing accuracy. Due to the stochastic nature of the PPO and SAC policies, the planned path and final grasp position were different for each attempted grasp. For each grasp position, two grasp attempts were made. This testing serves to validate the accuracy of the PyBullet digital twin, and the potential for this methodology for real-world RL implementation. Some of the sample real and simulated test grasps and pick-and-place actions can be viewed in the Supplementary Material.

During testing, the Panda agent was instantiated in PyBullet, and each incremental step the agent took in the simulation space was replicated in the real world. After each action step was executed in PyBullet, the agent waited to take a new step until the real-world arm moved to match its digital twin. During each step, the Franka-ROS feedback loop ensured the positional accuracy of the end effector. The incremental stepping approach was implemented to slow down the real-world testing to prevent damage to the robot during collisions with the mounting table or any objects in the robot's vicinity.

Table 4 depicts the real-world performance of the RL agent for each task. As shown, the reach and grasp tasks were completed with relatively high accuracy. Two minor issues that caused a reduction in task completion rate during testing were (1) minor difference in geometry of the tested object vs. the simulated object, and (2) the calibration of the physics environment.

Table 4. Real-world task performance.

Real Problem	Reward	Positional Feedback Method	RL Implementation	Task Success Rate
Panda reach	Dense	Vector	PPO	90
Panda reach	Dense	Vector	SAC	90
Panda grasp	Dense	Vector	PPO	70
Panda grasp	Dense	Vector	SAC	80
Panda pick-and-place	Dense	Vector	PPO	70
Panda pick-and-place	Dense	Vector	SAC	60

Future work that could improve real world testing includes the following. First, the simulated or real-world target could be modified so both target objects match. This change is relatively small but will prevent geometry differences from causing failure. Second, the simulation environment can be upgraded to the recently open-sourced simulation package MuJoCo. Due to the higher accuracy of this environment, and the low computational cost, this change would improve sim-to-real transfer while not increasing training time. Finally, positional sensing could be applied to the real-world target block to ensure that the real and simulated target positions match during each trained task.

4. Conclusions

Considerable progress has been made in this project toward the goal of creating simulated and real-world autonomous robotic agents capable of performing tasks such as reach,

grasp, and pick-and-place. To achieve this goal, custom representative simulation environments were created, a combined RL and traditional control methodology was developed, a custom tuning pipeline was implemented, and real-world testing was completed.

Through the extensive tuning of the RL algorithms; SAC and PPO, optimal hyperparameter combinations and network designs were found. The results of this training were implemented to complete a comparison between PPO and SAC for robotic control. The comparison indicates that PPO performs best when the task is complex (involves object avoidance) and time is readily available. SAC performs best when the task is simple and time is limited.

After optimal SAC and PPO hyperparameters were found, SAC and PPO algorithms were connected with the Libfranka, Franka-ROS, and MoveIt control packages to test the connection between the simulated PyBullet agent and the real-world Panda robot. Real-world testing was conducted to validate the novel communication framework developed for the simulated and real environments, and to verify that real-world policy transference is possible.

During real-world testing, the accuracy of the reach, grasp, and pick-and-place tasks were reduced by 10–20% compared to the simulation environment. This result provides and optimistic indication of the future applicability of this method, and also indicates that further calibration of the simulation environment and modifications to the target object are required.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/robotics12010012/s1, Video S1: RL GRASP. Real and simulated grasp and pick-and-place testing.

Author Contributions: Conceptualization, A.L.; methodology, A.L.; writing—original draft preparation, A.L.; writing—review and editing, H.-J.K.; supervision, H.-J.K.; project administration, H.-J.K.; funding acquisition, H.-J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Korea-Canada Artificial Intelligence Joint Research Center at the Korea Electrotechnology Research Institute (Operation Project: No. 22A03009), which is funded by Changwon City, Korea.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to other continuing research on this topic.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Massa, D.; Callegari, M.; Cristalli, C. *Manual Guidance for Industrial Robot Programming*; Emerald Group Publishing Limited: Bingley, UK, 2015; pp. 457–465. [CrossRef]
2. Biggs, G.; Macdonald, B. *A Survey of Robot Programming Systems*; Society of Robots: Brisbane, Australia, 2003; p. 27.
3. Saha, S.K. *Introduction to Robotics*, 2nd ed.; McGraw Hill Education: New Delhi, India, 2014.
4. Craig, J. *Introduction to Robotics Mechanics and Control*; Pearson Education International: Upper Saddle River, NJ, USA, 2005.
5. Al-Selwi, H.F.; Aziz, A.A.; Abas, F.S.; Zyada, Z. Reinforcement Learning for Robotic Applications with Vision Feedback. In Proceedings of the 2021 IEEE 17th International Colloquium on Signal Processing & Its Applications (CSPA), Langkawi, Malaysia, 5–6 March 2021.
6. Tai, L.; Zhang, J.; Liu, M.; Boedecker, J.; Burgard, W. A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation. *arXiv* **2016**, arXiv:1612.07139.
7. Kober, J.; Bagnell, A.; Peters, J. Reinforcement Learning in Robotics: A Survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
8. Liu, D.; Wang, Z.; Lu, B.; Cong, M.; Yu, H.; Zou, Q. A Reinforcement Learning-Based Framework for Robot Manipulation Skill Acquisition. *IEEE Access* **2020**, *8*, 108429–108437. [CrossRef]
9. Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In Proceedings of the 2nd Conference on Robot Learning, Zürich, Switzerland, 29 October 2018.
10. Mohammed, M.Q.; Chung, K.L.; Chyi, C.S. Pick and Place Objects in a Cluttered Scene Using Deep Reinforcement Learning. *Int. J. Mech. Mechatron. Eng.* **2020**, *20*, 50–57.

11. Liu, R.; Nageotte, F.; Zanne, P.; de Mathelin, M.; Drespp-Langley, B. Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review. *arXiv* **2021**, arXiv:2102.04148.
12. Kleeberger, K.; Bormann, R.; Kraus, W.; Huber, M. A Survey on Learning-Based Robotic Grasping. *Curr. Robot. Rep.* **2020**, *1*, 239–249. [CrossRef]
13. Xiao, Y.; Katt, S.; ten Pas, A.; Chen, S.; Amato, C. Online Planning for Target Object Search in Clutter under Partial Observability. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.
14. Sutton, R.; Barto, A. *Reinforcement Learning: An Introduction*; The MIT Press: Cambridge, MA, USA; London, UK, 2018.
15. Russell, S.; Norvig, P. *Artificial Intelligence A Modern Approach*, 4th ed.; Pearson Education, Inc.: Hoboken, NJ, USA; ISBN 978-0-13-461099-3.
16. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Magazine* **2017**, *34*, 26–38. [CrossRef]
17. Ng, A.; Harada, D.; Russell, S. Policy invariance under reward transformations theory and application to reward shaping. In Proceedings of the Sixteenth International Conference on Machine Learning, San Francisco, CA, USA, 27 June 1999; pp. 278–287.
18. Gualtieri, M.; Pas, A.; Platt, R. *Pick and Place Without Geometric Object Models*; IEEE: Brisbane, QLD, Australia, 2018; pp. 7433–7440.
19. Gualtieri, M.; Platt, R. Learning 6-DoF Grasping and Pick-Place Using Attention Focus. *arXiv* **2018**, arXiv:1806.06134.
20. Pore, A.; Aragon-Camarasa, G. On Simple Reactive Neural Networks for Behaviour-Based Reinforcement Learning. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020.
21. Li, B.; Lu, T.; Li, J.; Lu, N.; Cai, Y.; Wang, S. ACDER: Augmented Curiosity-Driven Experience Replay. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 21 August 2020; pp. 4218–4224.
22. Marzari, L.; Pore, A.; Dall'Alba, D.; Aragon-Camarasa, G.; Farinelli, A.; Fiorini, P. Towards Hierarchical Task Decomposition Using Deep Reinforcement Learning for Pick and Place Subtasks. *arXiv* **2021**, arXiv:2102.04022.
23. Pedersen, M.; Nalpantidis, L.; Andersen, R.; Schou, C.; Bøgh, S.; Krüger, V.; Madsen, O. Robot skills for manufacturing: From concept to industrial deployment. *Robot. Comput.-Integr. Manuf.* **2016**, *37*, 282–291. [CrossRef]
24. Lobbezoo, A.; Qian, Y.; Kwon, H.-J. Reinforcement Learning for Pick and Place Operations in Robotics: A Survey. *Robotics* **2021**, *10*, 105. [CrossRef]
25. Mohammed, M.; Kwek, L.; Chua, S. Review of Deep Reinforcement Learning-Based Object Grasping: Techniques, Open Challenges, and Recommendations. *IEEE Access* **2020**, *8*, 178450–178481. [CrossRef]
26. Howard, A. Gazebo. Available online: http://gazebosim.org/ (accessed on 20 September 2022).
27. Erez, T.; Tassa, Y.; Todorov, E. Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 4397–4404.
28. DeepMind Opening Up a Physics Simulator for Robotics. Available online: https://www.deepmind.com/blog/opening-up-a-physics-simulator-for-robotics (accessed on 11 July 2022).
29. Coumans, E. Tiny Differentiable Simulator. Available online: https://pybullet.org/wordpress/ (accessed on 10 June 2022).
30. Gallouédec, Q.; Cazin, N.; Dellandréa, E.; Chen, L. Multi-Goal Reinforcement Learning Enviroments for Simulated Franka Emika Panda Robot. *arXiv* **2021**, arXiv:2106.13687.
31. Shahid, A.A.; Piga, D.; Braghin, F.; Roveda, L. Continuous Control Actions Learning and Adaptation for Robotic Manipulation through Reinforcement Learning. *Autonomous Robots* **2022**, *46*, 483–498. [CrossRef]
32. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.
33. Karagiannakos, S. Trust Region and Proximal Policy Optimization (TRPO and PPO). Available online: https://theaisummer.com/TRPO_PPO/ (accessed on 13 December 2021).
34. Schulman, J.; Levine, S.; Moritz, P.; Jordan, M.I.; Abbeel, P. Trust Region Policy Optimization. *arXiv* **2015**, arXiv:1502.05477.
35. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018.
36. Tuomas, H.; Zhou, A.; Hartikainen, K.; Tucker, G. Soft Actor-Critic Algorithms and Applications. *arXiv* **2019**, arXiv:1812.05905v2.
37. Haarnoja, T.; Ha, S.; Zhou, A.; Tan, J.; Tucker, G.; Levine, S. Learning To Walk via Deep Reinforcement Learning. *arXiv* **2019**, arXiv:1812.11103.
38. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; Dormann, N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *J. Mach. Learn. Res.* **2021**, *22*, 1–8.
39. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kegl, B. *Algorithms for Hyper-Parameter Optimization*; Curran Associates Inc.: Granada, Spain, 2011; pp. 2546–2554.
40. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework. In Proceedings of the Applied Data Science Track Paper, Anchorage, AK, USA, 4 August 2019.
41. Mataric, M.J. Reward functions for accelerated learning. In *Machine Learning Proceedings 1994*; Elsevier: Amsterdam, The Netherlands, 1994; pp. 181–189.
42. Anca, M.; Studley, M. Twin Delayed Hierarchical Actor-Critic. In Proceedings of the 2021 7th International Conference on Automation, Robotics and Applications (ICARA), Prague, Czech Republic, 4–6 February 2021.

43. Franka Emika. Data Sheet Robot—Arm & Control. Available online: https://pkj-robotics.dk/wp-content/uploads/2020/09/Franka-Emika_Brochure_EN_April20_PKJ.pdf (accessed on 13 July 2021).
44. Görner, M.; Haschk, R.; Ritter, H.; Zhang, J. MoveIt! Task Constructor for Task-Level Motion Planning. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.
45. Coumans, E.; Bai, Y. PyBullet Quickstart Guide. Available online: https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3 (accessed on 12 March 2022).