# Robotic Reach, Grasp, and Pick-and-Place using Combined Reinforcement Learning and Traditional Controls

by

Andrew Lobbezoo

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Applied Science

in

Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2022

## Author’s Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

# Abstract

Electrically actuated robotic arms have been implemented to complete tasks which are repetitive, strenuous, and/or dangerous since they were first developed in the 1970s. More than 50 years have passed since initial development; however, robots in factories today are still operated with the conventional control strategies requiring individual programming on a task-by-task basis, with no margin for error. The implementation of conventional controls relies on experienced technicians and skilled robotic engineers sending commands on graphical or text-based programming interfaces to perform simple actions. Although automation has been shown to drastically increase productivity and reduce workplace injuries, the initial time and R&D cost for setting up robotic agents with traditional methods is presently too large for many firms.

As an alternative to traditional operation planning and task programming, machine learning has shown significant promise with the development of reinforcement learning (RL) based control strategies. With RL, robotic agents can be presented with a task which they learn to solve through the exploration of various action sequences in the real world, or on internal simulated models of the environment. There are some existing RL applications; however, most examples are based on relatively simple video games and basic robotic tasks (inverted pendulum, vector-based reach, and so on). Additionally, the documentation for much of this research is limited, there is little real-world testing, and there is room for significant improvement in performance. The objective of this project is to implement RL based control strategies in simulated and real environments to validate the RL approach for standard industrial tasks such as reach, grasp, and pick-and-place. The goal for this approach is to bring intelligence to robotic control so that tasks can be completed without precisely defining the environment, target object positions, and action plan.

To achieve the primary objective of this research, the following sub-objectives were pursued: 1) develop a custom simulation task environment, 2) create an RL pipeline for tuning and training a robotic RL agent, 3) develop a methodology for a novel semi-supervised RL system for improving image-based RL, 4) setup the Panda robot and establish a communication, control, and path planning system, and 5) tune, train, and test simulated and real-world RL based control.

After developing the environments, creating the training and tuning framework, and establishing the real-world robotic control in objectives 1 to 4, extensive training and testing was conducted in objective

5. Results from testing showed that model performance was highly dependent on task difficulty. A high task completion rate was the outcome from training an RL network in simulation with coordinate-based positional feedback. For this simulation set, the robotic agents were able to independently learn to complete tasks with a high precision and repeatability. The outcome from training a network in simulation with image-based positional feedback was respectively poor. For the image-based tasks, the agent converged on sub-optimal solutions and underperformed expectations due to difficulties training the CNN positional location extractor. To overcome the issues with image-based RL training, the novel semi-supervised RL approach was implemented and tested. The results from this testing indicate that RL training performed well with image-based inputs given a pre-trained feature extractor. The semi-supervised methodology shows potential; however, this approach has the downside of requiring additional data collection for supervised training.

After training in simulation, real-world reach, grasp, and pick-and-place testing was completed with coordinate-based positional inputs. The real-world testing validated the communication framework between the simulated and real environments and indicated that real-world policy transference was possible. Accuracy of coordinate-based reach, grasp, and pick-and-place was reduced by 10-20% compared to the simulation environment, which indicates that additional model calibration is required.

The results from this research provide optimistic preliminary data on the application of RL to robotics. Further research, which is required to bridge the gaps on image-based learning, should include network generalization, domain adaptation, and imitation learning.

# Acknowledgements

I would like to thank everyone involved in making this experience possible. First, I would like to thank Prof. HJ Kwon for his kind and heartfelt support, consistent guidance, and mentorship throughout my studies. Second, I would like to thank Dr. Yanjun Qian for lending his technical acumen to the various challenges I experienced throughout this project. Additionally, I would like to thank my reviewers: Prof. Arash Arami and Prof. Soo Jeon for their comments and feedback on my thesis.

I know that this work was only possible due to the consistent support and love shown to me by my family and friends. My siblings, and my support networks from McMaster and Waterloo have always provided me with the motivation to push through whenever I was overwhelmed. For that I will always be in your debt.

Thank you.

# Table of Contents

# List of Figures

# List of Tables

# Nomenclature

| | |
|---|---|
| A2C | Actor-Critic |
| BC | Behavior Cloning |
| CNN | Convolutional Neural Network |
| DDPG | Deep Deterministic Policy Gradient |
| DH | Denavit-Hartenberg |
| DP | Dynamic Programming |
| FCI | Franka Control Interface |
| FFNN | Feed Forward Neural Network |
| HER | Hindsight Experience Replay |
| IRL | Inverse Reinforcement Learning |
| MC | Monte-Carlo |
| MDP | Markov Decision Process |
| NN | Neural Network |
| PPO | Proximal policy optimization |
| PS | Policy Search |
| RL | Reinforcement Learning |
| ROS | Robot Operating System |
| SARSA | State-Action-State-Reward |
| TD | Temporal Difference |
| TRPO | Trust Region Policy Optimization |

# Chapter 1: Introduction

## 1.1 Project Motivation

Over 50 years ago, the first electrically powered robot, “The Stanford Arm”, was built [1], [2]. The resemblance between the Standford arm and the modern Panda research robot can be viewed in Figure 1. Surprisingly, the similarities between these two robots extend beyond the mechanics. After many years of application, most robots in factories are operated with the conventional control strategies developed decades ago.

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 and skilled robotics engineers sending commands on graphical or text-based programming interfaces to perform a sequence of simple tasks [3]–[6]. Due to the high level of expertise required to operate robots, the application of robotics in small and medium sized firms is ~50% less than seen in large and mega sized firms [7]. Robotic automation has been shown to drastically increase labor productivity and reduce workplace injuries [7]; however, the initial time and research and development (R&D) cost for setting up robotic agents with traditional methods is too large for most firms.

Machine learning has shown significant promise for replacing traditional robotic control with intelligent reinforcement learning (RL) based control strategies [8]–[16]. In RL, agents are presented with a task, which they learn to solve by exploring various action sequences in the real world, or on internal simulated models of the environment [17]. Compared to other RL applications such as self-driving cars

**Figure 1: The Stanford Arm [2] (left) and the Franka Emika Panda Arm (right)**

and video games [17]–[19], robotic control is difficult due to the high dimensionality of the problem and the continuous space of actions [20]–[22].

To date, RL has been successfully applied to robotics for basic tasks such as target object grasping, placement, and basic manipulation [9], [10], [15], [23]–[25] which gives some indication of its potential as a method of controlling robotic agents [26]. However, there is room for significant improvement for tasks with long action sequences (traditionally done by humans) and tasks with image-based positional control. All existing methods of RL in robotics still require extensive manual programming and/or are only capable of completing tasks with short action sequences, and most are based on known gripper and target object coordinates. Before the idea of a functional multipurpose robot can be instantiated, more research is required in the field of RL in robotics.

## 1.2 Objectives

The principal objective of this thesis 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 sub-objectives to achieve the higher-level goal:

1. Develop a pipeline for training robotic agents in simulation. This task involves simulating the Panda, building environments, implementing an RL framework, and developing a hyperparameter tuning system.
2. Install the robotic arm and an RGBD sensor, establish communication and control with a real-time Robot Operating System (ROS) network, test various kinematic packages, and establish communication between the digital twin and the real robot.
3. Train various models and compare performance of each, develop modifications to the existing system to get improved performance over baseline, and test the RL control system in the real-world.

## 1.3 Contributions

The main contributions of this research to the RL for robotics field can be summarized as follows:

1. Implemented a previously untested RL approach for high-level robotic arm task planning and trajectory generation.
2. Developed improved simulation tasks for training and testing RL on the Panda robot.

3. Modified and tuned the existing framework networks and their associated hyperparameters for the application of RL to robotics.
4. Tested image-based RL with various networks to determine feasibility for realistic high complexity problems.
5. Developed an improved methodology for feature extraction through the development of a semi-supervised RL approach.

## 1.4 Thesis Outline

Chapter 1 reviews the project motivation, objectives of the research, and contributions to the field.

Chapter 2 provides a comprehensive literature review on the background required to understand the framework for RL as applied to robotics. Due to the novel nature of this research for the lab, special attention is given in Sections 2.3-2.5 to the theory of reinforcement learning, and the various strategies currently used in the research. Additionally, section 2.9 performs an extensive audit of the available publications related to RL grasp and pick-and-place, with the goal of providing direction for this project.

Chapter 3 presents the simulation study pipeline implemented for this project. Sections 3.2-3.5 focus on the development of the custom environment and tasks. Section 0 reviews the RL codebase and hyperparameter tuning frameworks. Section 3.7 reviews the semi-supervised RL implementation.

Chapter 4 describes the real-world control interface established for this project. Section 4.1-4.2 introduces the Panda robot and the applicable control strategies. Section 4.3 examines the control framework. Section 4.4 reviews the communication framework tested during data collection.

Chapter 5 examines the results from the simulation study and the real-world testing, and discusses the implications. Section 5.1 completes a deep dive on the hyperparameter tuning, and reviews the convergence of the vector, and image-based tasks with various feedback strategies. Section 0 reviews the results from real-world testing.

Chapter 6 presents the conclusion of the research and the recommendations for future work.

# Chapter 2: Literature Review

Some parts of the thesis are published in the journal: A. Lobbezoo, Y. Qian, and HJ. Kwon, "Reinforcement Learning for Pick and Place Operations in Robotics: A Survey." Robotics 2021, 10, 105. https://doi.org/10.3390/robotics10030105

## 2.1 Scope of Review

This survey provides a holistic review of reinforcement learning (RL) in robotics as it relates to the grasping and pick-and-place tasks. The fundamental formulation of robotic control and RL is reviewed, and the elements unique to the robotics application are analyzed in detail. A comprehensive list of papers published on the topic are presented, and a critical analysis on the state of research is conducted.

## 2.2 Traditional Robotic Control

Two standard methods exist for controlling robotic arms. The first method is joint based control, and the second method is kinematics based.

The first method involves controlling each of the seven individual joints of the robotic arm by setting positional, or torque and velocity values. Generally, this approach is ineffective for complex tasks, and serves only as a method for testing the robotic agent's range and functionality. With this approach, the controlling agent (human or AI) must specify seven parameters when implementing angular control, and 14 parameters when implementing torque and velocity control.

The second approach has been prevalent in robotics since the 1970s and involves implementing traditional kinematic equations for path planning. This approach removes some of the complexity for the controlling agent, as only 6 parameters must be specified for the end effector position (x, y, and z) and orientation (pitch, yaw, and roll). The second approach is founded on the framework of forward and inverse kinematics.

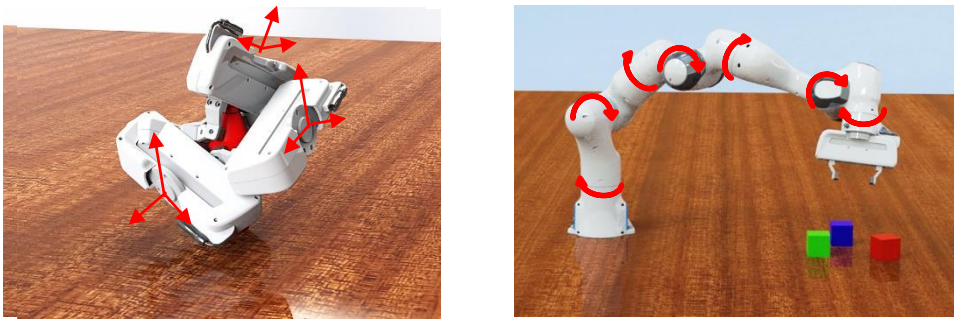


**Figure 2: Control Methods shown with the Panda (Franka Emika)**

### 2.2.1 Forward Kinematics

Forward kinematics is a series of calculations which can be completed to determine the end effector position based on the current joint angles of the robot, and the parameters for the system (reference frame, link lengths, and angle constraints).

Before forward kinematics can be calculated, the reference frames and parameters of the robotic arm must be defined. The standard way to describe a serial manipulator such as a robotic arm, is with a Denavit–Hartenberg (DH) table. The DH table is used to define the four parameters required to fully describe any linkage. The parameters include: the offset along previous z, the angle about previous z,

| Joint | $a(m)$ | $d(m)$ | $\alpha(rad)$ | $\theta(rad)$ |
|---|---|---|---|---|
| *Joint 1* | *0* | *0.333* | *0* | $\theta 1$ |
| *Joint 2* | *0* | *0* | $-\pi/2$ | $\theta 2$ |
| *Joint 3* | *0* | *0.316* | $\pi/2$ | $\theta 3$ |
| *Joint 4* | *0.0825* | *0* | $\pi/2$ | $\theta 4$ |
| *Joint 5* | *-0.0825* | *0.384* | $-\pi/2$ | $\theta 5$ |
| *Joint 6* | *0* | *0* | $\pi/2$ | $\theta 6$ |
| *Joint 7* | *0.088* | *0* | $\pi/2$ | $\theta 7$ |
| *Flange* | *0* | *0.107* | *0* | *0* |

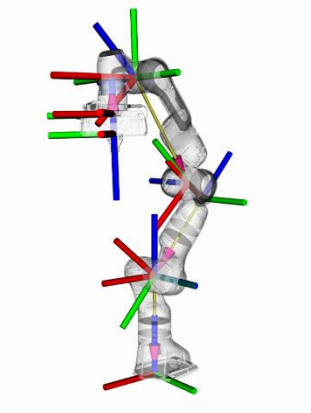


**Figure 3: Panda Denavit–Hartenberg (DH) table for the Panda robot [31,32]**

$$T = \left[\begin{array}{ccc|c} & R & & T \\ & & & \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} \cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & r\cos\theta \\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & r\sin\theta \\ 0 & \sin\alpha & \cos\alpha & d \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \qquad {}^{0}\boldsymbol{T}_{6} = {}^{0}\boldsymbol{T}_{1}\,{}^{1}\boldsymbol{T}_{2}\,{}^{2}\boldsymbol{T}_{3}\,{}^{3}\boldsymbol{T}_{4}\,{}^{4}\boldsymbol{T}_{5}\,{}^{5}\boldsymbol{T}_{6}\,.$$

**Figure 4: Transformation matrix (left), calculation of end effector position (right) [30]**

the length of common normal, and the angle about the common normal [6], [27]. The DH table for the Panda robot is shown in Figure 3 [28], [29].

The DH table parameters are implemented to create the transformation matrix between any two joints. The transformation matrix for any joint relative to the prior joint can be related with the transformation as shown in Figure 4. The resulting end effector position can be calculated with the combination of all these transformation matrixes, for each joint. The calculation shown in Figure 4 outputs the resulting x, y, and z position of the end effector, and serves as the process for calculating the forward kinematics [30].

### 2.2.2 Inverse Kinematics

Inverse kinematics are implemented to calculate the joint angles required to reach a target end effector pose. Inverse kinematics can be understood as the reversed process of forward kinematics, as shown in Figure 5.

The calculation of inverse kinematics is more complicated than forward kinematics, as it involves solving nonlinear differential equations in which multiple or no solutions may exist. In inverse kinematics, the goal is to find the joint positions required, given the relative positions of the end effector current position, target position, and the base. In cases for which algebraic methods are not possible, numerical methods are required [6]. Due to the difficulty in solving the inverse kinematics, pre-built packages as MoveIt [28] or RoboticToolbox[31] are traditionally applied.

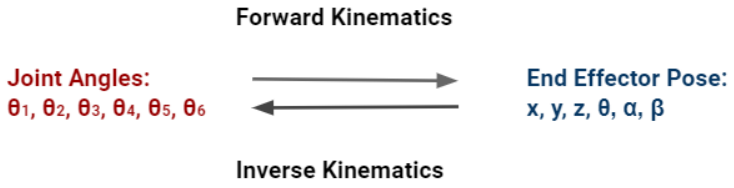



**Figure 5: Forward and inverse kinematics**

### 2.2.3 Real World Application

An engineer does not directly code the forward and inverse kinematics to complete each robotic task. Instead, graphical or text-based programming environments implement interpreters and compilers to translate basic commands or position controls to the robot in its native language [4], [32]. The programming environment is determined by the interfaces provided by the supplier and the choice between online and offline programming [5].

Online programming is traditionally used in industry to allow skilled operators to teach basic motion trajectories and operations to the robot with teach-pendant control interfaces or manual robotic arm positioning. There are many difficulties associated with online programming, such as the robot downtime, the range in pendant interfaces, and the lack of dynamic control to handle changes in the environment. Online programming is only usable for basic tasks with high repeatability. Reprogramming is required for any minor variances in the workspace, limiting the applicability for this type of programming [5], [33]. In such cases, offline programming can be an alternative control technique.

Offline programming involves skilled programmers interacting with graphical program environments and lower-level programming languages to test control strategies on CAD representations of the robotic arm and its environment. Offline programming has the advantages of reducing downtime (and the associated downtime cost) and allowing the implementation of advanced control strategies [4], [5], [33].

## 2.3 Reinforcement Learning Formulation

Reinforcement learning (RL) is the optimal machine learning approach for this problem. Reinforcement learning has many branches and implementations, which are discussed below.

### 2.3.1 Reinforcement Learning Formulation

The goal for completing robotic operations without task-specific programming is to allow the agent to independently perform actions in the environment, and then learn the optimal strategy for its future tasks based on its experiences [17]. The learning process is analogous to a child learning to walk. A child must first attempt the motions of moving limbs, rolling over, and crawling before it can complete the target action of walking. The child's behavior is driven by rewards from its environment after the

completion of each task. Positive affirmation from the child's parents is an example of an environmental reward which would drive the walking behavior.

### 2.3.2 Markov Decision Process

The Markov decision process (MDP) is the underlying formulation for learning with these reinforcement iterations [17]. The MDP encourages specific behaviors and discourages others by feeding rewards or punishments to the agent based on the agent's actions in the environment. In the MDP, the agent selects an action based on its policy function for available actions in each state $s \in \mathcal{S}$. In the deterministic case, the same action $a$ is always taken in state $s$, and the policy is denoted as $\pi(s)$. For an agent operating in a stochastic environment, the policy defines the probabilities of selecting each possible action given a state. The policy for the stochastic case is denoted as $\pi(a|s)$ [17]. The primary goal for RL is to find the optimal policy $\pi^*(a|s)$. The optimal policy can be found through the implementation of the value function approach [10] or directly through a policy search (both discussed further in Section 2.5) [34].

The goal of the policy function is to optimize the expected return $G$, the (weighted) sum of all discounted returns before the final time step. At time step $t$, the expected return is expressed as [17]

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{1}$$

where $T$ is the total time step, $R_k$ is the reward for each step, and $\gamma$ is the discounting parameter, a number between 0 and 1, which discounts future rewards. Figure 6 depicts the standard MDP for a robotic arm.

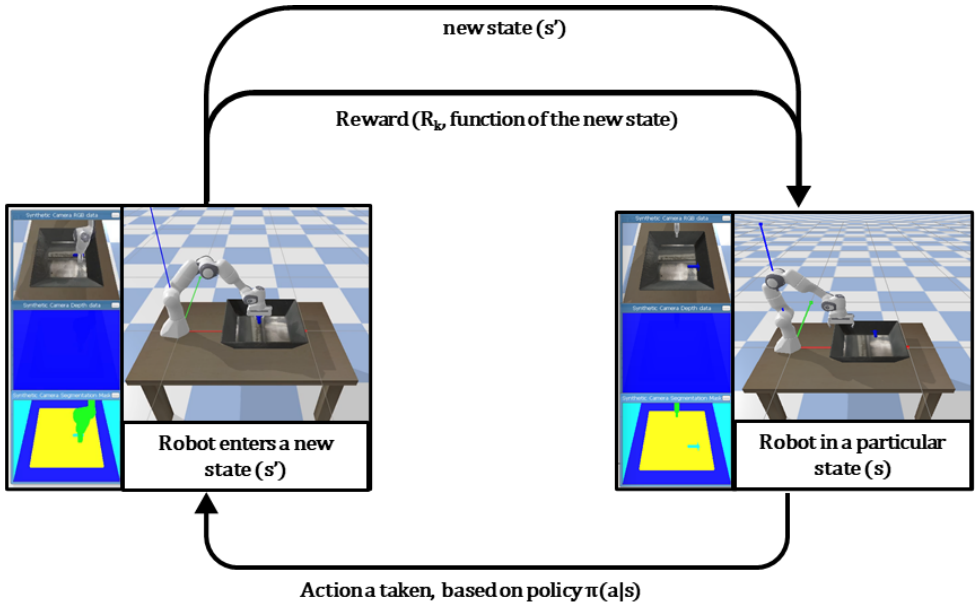



**Figure 6: Agent-environment interaction in the MDP.**

Again, the intuition behind the MDP can be understood with the analogy of a child learning to crawl. The policy function is analogous to the child's thought process before attempting an action. The child's policy is dependent on the reward it receives for completing the action and is the most significant factor which drives learning.

### 2.3.3 RL for Robotics

RL formulated around the basic MDP appears to be intuitive and straightforward; however, robotic training has specific nuances. The primary impediments include the high dimensionality, continuous action space, and extensive training time.

Most robotic arms implement 6 degrees of freedom as this is the minimum freedom required to reach any position in space at any angle [35]. The freedom of movement makes the task complex, as all actions require the control of the torque and velocity of each joint of the agent. Additionally, the robot operates in a complex environment with a continuous operational space. Rather than having discrete state choices, the agent in this problem has infinitely many states to explore. The method of limiting the number of actions and states to a tractable set becomes a choice that affects the training time, test accuracy, and repeatability [21].

Many RL algorithms have been developed for the application with simple agents and environments. OpenAI (https://gym.openai.com/, accessed on 15 August 2021) has attempted to standardize the benchmarking environments for RL by creating open access simulations for testing control strategies. Cartpole is an example of a classic control problem found on OpenAI, in which the agent can make a command (left or right) and receive a low dimensional feedback signal (position and angular velocity) [36]. Compared to this simple benchmark, robotic arm manipulation is foundationally more difficult. The high dimensionality of robotic arms drives the need for complex reward functions to promote smooth motions and minimize training time [20]. Even with complex reward functions, the type of object picked, and the pose (orientation and position) of the target object affects the probability of task completion [37].

Due to the high dimensionality and continuous action space, the agent requires extensive training to learn the optimal policy. Online training is costly because of the required human supervision, robotic wear, danger of damaging the robot, and lack of available training robots. The costs associated with online training drives the need for simulation training [10]; however, simulation training often has a low test accuracy. The fundamental problem with simulation training is that the knowledge trained in the simulation must be transferred to the real world. Any inaccuracies in modeling physical parameters such as shapes and weights of objects handled, lighting, or friction coefficients will cause test accuracies to be lower than training accuracies [15].

## 2.4 Policy Optimization

Finding the optimal policy for a RL problem involves value iteration or policy search. With both policy optimization techniques, the solution process changes depending on if the approach is model-based or model-free. Additionally, the exploration-exploitation tradeoff significantly influences the speed of solution convergence.

The model-based technique implements an understanding of the environment through prior learning or through state-space search to create an approximation of the transition probabilities between states given actions $p(s'|\, s, a)$) (also commonly formulated as $T(s'|\, a, s)$). The model-free (direct) design has no explicit representation of the system dynamics. Instead, the model-free technique learns the value function directly from the rewards received during training [17], [38], [39]. For RL in robotics, most approaches are model-free, since creating a perfect model of all transition probabilities in a continuous space is intractable.

Finding the policy for the agent always involves a tradeoff between searching random actions to find the optimal policy and greedily selecting the optimal action to improve rewards. In RL, $\varepsilon$ is the variable used to represent the probability of not taking the reward-maximizing action during exploration [17]. To ensure that the whole action space is explored, the agent traditionally implements some form of on- or off-policy learning. On-policy learning improves the current policy marginally by selecting the optimal action for the majority $(1-\varepsilon)$ of actions and selecting a random action with a probability $(\varepsilon)$ for the rest. Off-policy learning applies two policies; the target policy, which is the policy being learned about, and the behavior policy, which is the policy that drives behavior for learning [17].

In the following sections, the value function and policy search approaches are explained, and different search techniques inside each methodology are reviewed [39].

### 2.4.1 Value Function Approach

The value function approach is based on dynamic programming (DP). DP is a strategy of breaking up a complex problem into a sequence of successive sub-problems which can be iteratively solved. Implementing DP for RL requires an understanding of value functions.

### 2.4.2 Value Functions

Value functions are founded on the understanding that each state has an associated value dependent on the reward achieved in that state and that state's potential to be a step in achieving future rewards with the agent's current policy. A reward is often only received in a specific goal state, so positions proximal to the goal state have the highest value. The value function is found by breaking down the problem into successive sub-problems following the sequence of states. The Bellman equations in Equations (2) and (3) are established to solve for action-value function $q_\pi$, and state value function $v_\pi$ respectively [17].

$$q_\pi(s,a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{2}$$

$$v_\pi(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{3}$$

In stochastic reinforcement learning, randomness in the system implies that taking an action $a$ from state $s$ does not always result in the same final state $s'$. The transition dynamics (or transition probability) of the environment $p(s', r \mid s, a)$ is used to represent the probability of reaching state $s'$

and receiving a reward $r$ given that action $a$ is taken in state $s$ [17]. Using this formulation, the action-value function $q_\pi(s,a)$ given in Equation (2) can be understood as being equal to the summation of the transition probabilities of reaching all possible states ($s'$) multiplied by the value ($r + \gamma v_\pi(s')$) of each possible state ($s'$).

The state-value function given in Equation (3) can be understood similarly. The primary difference between the two formulations is that the state-value function (Equation (3)) incorporates action selection. The probability of selecting action $a$ is dependent on the stochastic policy $\pi(a|s)$. Based on this understanding, the value of a given state is equal to the summation over all actions of the probability of selecting a particular action multiplied by the state value function $q_\pi(s,a)$.

In reinforcement learning, state- and action-value functions are implemented for finding the optimal policy with Bellman's optimality equations. The optimal state value function $v_*$ and optimal action-value function $q_*$ are found by maximizing the probability of choosing the actions which lead to states yielding values larger or equal to those from all other policies [17].

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{4}$$

$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \tag{5}$$

For finite discrete actions and states, lookup tables can be used to pick appropriate actions for each state to form the optimal policy. For pick-and-place operations in robotics, determining Bellman's optimality equations is difficult because the space of all states is continuous. Typically the Monte-Carlo (MC) or temporal difference (TD) approaches [10] are used. Both approaches are founded on dynamic programming (DP).

### 2.4.3 Dynamic Programming

DP is a model-based strategy that creates internal models for transition probabilities and rewards to calculate the value function. A primary assumption of DP is that the models perfectly represent reality, which is unrealistic for continuous spaces. Even though it cannot be directly applied for RL in robotics, DP is the framework for understanding the MC and TD techniques [17]. DP consists of two fundamental approaches: policy iteration and value iteration.

Policy iteration is a DP approach that consists of policy evaluation and policy improvement subtasks [18]. Policy evaluation is used to update the value of each state based on the current policy, transition probabilities, and the value of the following states [17].

$$v_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \tag{6}$$

Policy evaluation is completed by calculating the $v_k(s)$ (Equation (6)) for all states with an arbitrarily initiated policy. After an infinite number of exploratory steps ($k \to \infty$), the transition probabilities between states given actions $p(s', r \mid s, a)$ and the state-values $v_k(s)$ can be determined. Policy improvement then greedily selects the best action that can be taken in each state based on these newly found transition probabilities and state values [10], [17].

$$\pi'(s) \doteq \underset{a}{\operatorname{argmax}}\, q_\pi(s, a) \tag{7}$$

Policy improvement is solved by implementing $q_\pi(s, a)$ as shown [17] where argmax over actions involves taking the action which maximizes the value for each state. The new choice of action indicates the agent's policy in the next round of policy iteration. The process of policy iteration in DP consists of iterating between policy evaluation and policy improvement until the policy stops changing. At this point, the policy is assumed to be optimal [18].

Value iteration is an alternative to policy iteration, which can be used effectively by combining policy evaluation and improvement into one step. Value iteration is formulated as [17]

$$v_{k+1}(s) \doteq \max_{a} \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \tag{8}$$

With this formulation, the policy evaluation step is stopped after one iteration, and the policy for each state can be updated dynamically.

### 2.4.4 Model-Free Techniques

Techniques that are not model based can incorporate the DP strategy by using sample transitions and rewards to learn approximate value functions. Learning the value functions without complete probability distributions for all transitions requires exploration to ensure all states are visited, and non-greedy policies are investigated.

The Monte-Carlo (MC) averaging returns approach is a model-free technique that requires no prior knowledge. The model only requires experience through interaction with the environment. MC completes various actions in each state, and the average returns for each action are used as the expected action reward. After many iterations, the solution will converge to the actual underlying value for each action [10].

MC approaches implement a combined on- and off-policy technique. The state values for MC are determined during off-policy environmental interaction where sample trajectories are explored. After the state values are determined, the policy is found during the on-policy policy improvement step of policy iteration. During this step, the greedy policy is found by taking the value-maximizing actions for the values learned during off-policy exploration. [17]

The temporal difference (TD) approach is equivalent to an incremental implementation of MC. TD updates the estimates of the state-value function based on individual experiences without waiting for the outcome of the entire sample trajectory as opposed to MC. TD uses the difference between the old estimate of the value function and the experienced reward and new state to update the value function after one step, as shown in [15,25,26]:

$$V(s_t) \leftarrow V(s_t) + \alpha\,[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{9}$$

Here, $\alpha$ is the learning rate, a real number between 0 and 1 (usually 0.1 or smaller). This form of TD is referred to as the TD(0) algorithm.

State Action State Reward (SARSA) is a TD model which is used to update the action-value function ($Q(s,a)$) [10]. SARSA implements on-policy TD control, implying that the action value is updated after each exploration step. [17], [39]

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha * [R_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \tag{10}$$

Operating on-policy means that the action space is only explored due to the $\varepsilon - greedy$ aspect of on-policy search [17]. Due to its lack of off-policy exploration, SARSA does not have the same convergence guarantees as to the MC approach.

Q-learning is a method which is very popular in reinforcement learning research [17], [18], [39]. Q-learning is the off-policy equivalent of SARSA, formulated as: [17]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * [R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (11)$$

The use of off-policy TD allows Q-learning to perform updates to the value function regardless of the action selected [39]. This choice guarantees the convergence of $Q(s_t, a_t)$ if the learning rate $\alpha$ is well defined [17].

Value function approaches are intuitive; however, they often struggle to converge due to the high dimensionality of robotic state space, the continuous nature of the problem, and the bootstrapped approaches used to estimate the value function quickly. Another issue with the value function approach is that random exploration of the agent can result in mechanical damage due to range and torque limitations [40].

## 2.5 Policy Search Approach

Policy search (PS) is an approach to policy optimization which has recently shown potential in robotics as an alternative to the value function approach [10]. PS is a valuable tool in robotics due to the scalability with dimensionality, a vital issue with the value function approach [41]. Research has shown that PS is better than value function approaches for tasks that are finite horizon and are not repeated continuously [40].

PS does not approximate a policy as the value function does, i.e., by learning values of actions, then acting to optimize rewards based on these expected values. Instead, policy search implements a parameterized policy that can select the optimal action without reviewing the value function [17]. $\theta$, the variable representing the policy parameters, could be understood intuitively as weights of the deep neural network that influences the policy for a value function [17], [40] The formulation for the linear case is [40]

$$\pi_\theta(s) = \theta^T \Phi(s) \quad (12)$$

where $\Phi(x)$ is the basis function representing the state, and the policy parameters $\theta$ determine the choice of action in each state [40].

PS has the goal of learning the policy parameters that optimizes the performance measure $J(\theta)$, the expected cumulated discounted reward from a given state [40]. This reward is formulated as [17]

$$J(\theta) \doteq v_{\pi_\theta}(s) \quad (13)$$

wherein $v_{\pi_\theta}$ is the true value function for policy $\pi_\theta$.

To maximize the cumulative reward, the optimal policy can be found using the policy gradient method [17], which implements gradient ascent by

$$\nabla \mathrm{J}(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_\pi(s,a)\, \nabla_\theta \pi(a \mid s, \theta) \tag{14}$$

where $\mu(s)$ is the on-policy probability distributions for policy $\pi$.

Like the value function approach, the policy search approach can be implemented with model-free and model-based techniques [10]. The model-free technique uses online agent actions to create trajectories from which to learn. This technique has the advantage of not needing to learn an accurate forward model, which is often more challenging to learn than the policy itself. The model-based technique is more sample conservative, as the agent creates internal simulations of the dynamics of the model based on a few observations from the online model. The agent then learns the policy based on these internal simulations [40]. The online model-free technique has been implemented more in research due to its ease of use; however, model-based approaches have shown better ability to generalize to unforeseen samples [40].

Inside of policy search, several methods have been developed and implemented in robotics, including REINFORCE, Actor-Critic, Deterministic Policy Gradient, and Proximal Policy Optimization.

### 2.5.1 REINFORCE

REINFORCE is a MC based policy gradient method that uses vector samples (episodes) filled with state, action, reward triples found with the policy $\pi_\theta$ to approximate G, the return as defined in Equation (1). The algorithm updates the policy parameters incrementally with [17]

$$\theta \leftarrow \theta + \alpha\, \gamma^t G\, \nabla \ln \pi_\theta\, (a_t \mid s_t, \theta) \tag{15}$$

where $\nabla \ln \pi_\theta$ is equal to [17]

$$\nabla \ln \pi_\theta = \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)} \tag{16}$$

The intuition behind this formulation is that the policy parameter vector $\theta$ is shifted in the direction of the steepest ascent with positive return *G*, to improve the policy $\pi_\theta$ after each step.

REINFORCE is a valuable policy search method that creates a state value function and has guaranteed convergence; however, it tends to converge slowly compared to other bootstrapped methods.

### 2.5.2 Deterministic Policy Gradient

Another common implementation is the deterministic policy gradient (DPG) technique [42]. DPG implements Q-learning updates to determine the action value function as shown [42]

$$\delta_t = R_t + \gamma Q^w\big(s_{t+1}, \mu_\theta(s_{t+1})\big) - Q^w(s_t, a_t) \tag{20}$$

In this approach, DPG moves the policy in the direction of the gradient of $Q$ for continuous spaces, rather than selecting the globally optimal $Q$ value in each step [42].

$$w \leftarrow w + \alpha^w \delta \, \nabla_w Q^w(s_t, a_t) \tag{21}$$

$$\theta \leftarrow \theta + \alpha^\theta \nabla_\theta \mu_\theta(s_t) \, \nabla_a Q^w(s_t, a_t) \, |_{a=\mu_\theta(s)} \tag{22}$$

Here $Q^w(s_t, a_t)$ is a differentiable action value function which is used to replace the true action value function.

### 2.5.3 Actor-Critic Methods

The one-step advantage actor-critic (A2C) is analogous to the TD methods introduced above, which do not involve offline policy improvement steps. A2C implements the notion of $\delta_t$, the difference between the expected value and actual value for a given state and state-value weighting $w$ [17].

$$\delta_t = R_{t+1} + \gamma \hat{v}(s_{t+1}, w) - \hat{v}(s_t, w) \tag{17}$$

The difference parameter can be used each step to update the state-value weighting $w$ and the policy parameter $\theta$ with the equation [17]

$$w \leftarrow w + \alpha^w \delta \, \nabla \hat{v}(s \mid w) \tag{18}$$

$$\theta \leftarrow \theta + \alpha^\theta \delta \, \nabla \ln \pi_\theta \, (a \mid s, \theta) \tag{19}$$

The calculation for the policy and difference parameter is completed after each incremental exploration step in the search space.

A common actor-critic approach seen in the literature is asynchronous advantage actor-critic (A3C) [43]. A3C has the same formulation as A2C, with the slight difference of allowing multiple agents to

asynchronously explore different policies in parallel. This approach reduces the need for experience replay to stabilize learning in time and improve stability.

Another approach based on A2C which is seen in literature is Soft Actor-Critic (SAC) [44]. The SAC approach implements a novel entropy regularization approach to optimize the tradeoff between entropy and reward. The goal of this approach is to accelerate learning while preventing convergence on sub-optimal solutions.

### 2.5.4 Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) optimizes the return of the policy in the infinite-horizon MDP by implementing the loss function shown below:

$$Maximize_{\theta} E_t[\frac{\pi_\theta(at|st)}{\pi_{\theta old}(at|st)}\widehat{At} \tag{8}$$

TRPO introduces the KL convergence to constrain the difference between the new policy and the old policy. The trust region constraint can be expressed in the equation (9) below [45], [46]:

$$E_t[KL[\pi_{\theta old}(\cdot\,|st), \pi_\theta(\cdot\,|st)]] \leq \delta \tag{9}$$

where $\delta$ is the size of the region. TRPO is overly complicated to solve, as it requires a conjugating gradient method.

### 2.5.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is also a policy gradient method that implements a surrogate objective function. PPO has an advantage over TRPO, as it is much simpler to solve. PPO combines the region constraint as a penalty to the loss function TRPO. The loss function can be expressed as shown in equation (10) [46].

$$Maximize_{\theta} E_t[\frac{\pi_\theta(at|st)}{\pi_{\theta old}(at|st)}\widehat{A_t} - \beta KL[\pi_{\theta old}(\cdot\,|st), \pi_\theta(\cdot\,|st)]] \tag{10}$$

PPO introduces a clipped surrogate objective function, to penalize any changes that move the probability ratio between the new and old policy away from 1. The object function can be depicted as follows[46]

$$L^{clip}(\theta) = \hat{E}_t[\min\big(r_t(\theta)\widehat{A_t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_t}\big)] \tag{11}$$

By using this objective function, the action will be clipped in the interval $[1-\epsilon, 1+\epsilon]$ [47].

PPO is a policy gradient technique which was designed to provide faster policy updates then A2C or DPG. PPO applies the DPG structure, but updates the policy parameter $\theta$ based on a simple surrogate objective function [45]. The policy update includes a step of minimizing the penalty term associated with the difference between the surrogate and the original function.

### 2.5.6 Summary

The methods reviewed are fundamental to the different implementations of RL in robotics. The choice in policy optimization has a significant impact on the convergence guarantees and training time required.

## 2.6 Reward Shaping

In RL, the allocation of rewards drives the training process because the agent aims to maximize the reward signal [48]. Reward shaping is the technique that involves modifying the reward function to help the agent learn faster [49]. An efficient reward function can accelerate the learning process by introducing background knowledge to RL agents [50].

Traditional RL applications used monolithic goals [51]. Monolithic goals behave well with simple domains with clear reward objectives. However, as RL problems started to become more complex in the 1990s, this approach was found to be inadequate for modeling real physical world situations (such as robotics). The difficulties in recognizing the state information, nondeterministic and noisy environment, variances in the learning trails required, timeliness of the rewards, and the requirement of achieving multiple goals can contribute to the failure of monolithic goals for these applications [51].

To address these challenges, as an improvement to a naïve monolithic single goal reward function, Mataric proposed a heterogeneous reinforcement function [51]. The method broke the rewards down into several goals representing the knowledge of the domains, then grouped these goals with progress estimators. Goal refinement enables the system to learn through immediate transitionary rewards which can speed up learning and convergence. The issue with this approach is that the agent can achieve a net positive gain in reward by running cycles rather than finding the end state [20], [48].

To address this, Ng et al. [20] introduced a potential-based shaping algorithm. The algorithm incorporated a new reward function: $R' = R + F$, where $R$ is the traditional reward and $F$ is the shaping

reward function. The shaping reward function $F$ is a function over states, which should be chosen or constructed [50] based on the domain knowledge between two states. For example, $F$ could be a function of distance to the goal state, wherein $F$ increases as the agent selects actions closer to the goal state. Ng et al. proved that the optimal policy would not change given this new reward function [20].

Adding rewards is advantageous when the rewards represent the actual effect of the actions, however, adding rewards can also cause errors in the learning process when noisy reward inputs are given. To find balance between efficiency and effectiveness in RL reward shaping, Luo et al. proposed a method called Dense2Sparse[52]. By letting the system receive rewards continuously in the first several learning episodes, the policy is expected to converge quickly despite the noisy information. The suboptimal policy learned from the dense reward can be optimized under a noise-free sparse rewarding scheme, where a positive reward is given only when the task or sub-task is completed. Jang et al. found the concept of combining sparse and dense reward schemes to be effective with their research on training a Walker robot with high dimensional continuous action and state spaces [53].

Besides using a predefined change in the reward function, reward shaping can also be performed dynamically throughout the learning process. Tenorio-González et al. realized a dynamic reward shaping technique by implementing verbal feedback from a human observer [54]. The inclusion of human intervention can help the system converge faster but requires active human participation in the learning process. Konidaris and Barto [55] developed a method to improve the reward function autonomously. Their work focused on generalizing the knowledge learned by the agents for small tasks to produce a shaping function to apply to all rewards. Their experiment showed good performance of the shaped rewards in a rod positioning scenario.

Reward shaping is a tradeoff between accuracy and speed. A monolithic goal reward can be a noise-free input, but it is sometimes not practical for complex tasks. Breaking down the task and applying rewards frequently allows the system to learn faster, but it can also distract the agent [48], [56].

## 2.7 Pose Estimation for Grasp Selection

Pose estimation is a critical factor for pick and handle operations. Incorrect estimation of pose could result in poor grasp choice due to the robotic arm pose or unstable placement due to incorrect estimation of object pose. Gualtieri and Platt [21] noted the value in pose estimation when comparing the accuracy of cup placement against blocks and bottles. Blocks are symmetric in $x$, $y$, and $z$, bottles are symmetric

in *x* and *y*, and cups are symmetric only in *x*. The complexity of the cup pose made the pick-and-place task more difficult, which resulted in lower overall accuracy compared to blocks and bottles (Figure 4). Pose estimation is critical, as the task should be completable regardless of the complexity of the target object.

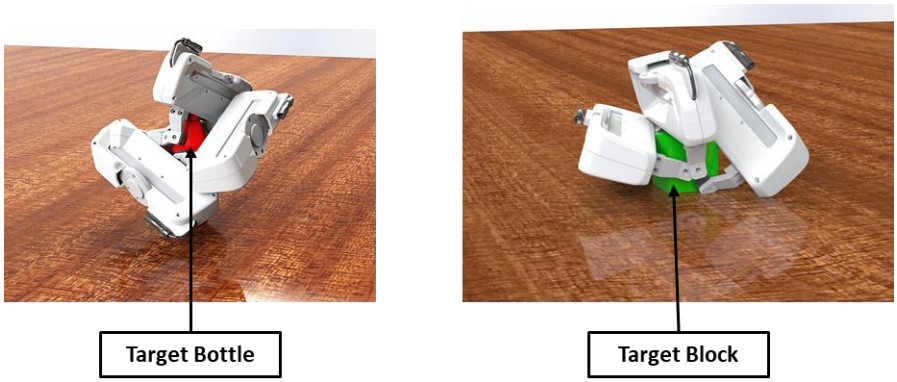



**Figure 7: Various grasp pose configurations for bottle and block targets. Bottle targets are symmetric in x and y compared to blocks which are symmetric in x, y, and z.**

The literature refers to two pose types, the object pose and the gripper pose (also referred to as the grasps [57]). The object pose is described by the position and orientation of the object being manipulated [58], [59]. Each object pose exists inside the set of all possible poses for that object, called the pose space [59]. The gripper pose is used to describe the different gripping positions, orientations, and jaw openings that could be used to grasp a target object [15]. Each gripper pose exists inside the set of all possible grasps for an object, called the grasp space. Pose estimation can be completed using analytic (geometric) or data-driven methods.

Analytic methods fit the point cloud of sensor input to known models (often CAD). Model fitting limits the agent to detecting the pose of items which have models available for comparison [60]. The analytic method first determines the target object pose, then selects an optimal gripper grasp based on the target object pose. The analytical method is usually set up as an optimization problem. The grasp is chosen from the grasp space based on maximizing the values for resistance to disturbance, dexterity for further manipulation, equilibrium to reduce object forces, and stability in the case of external forces or grasp errors [61], [62]. The analytic method is useful when precise placement in a particular orientation is required [15].

Data-driven methods are free from the constraint of requiring models, as this approach implements machine learning to estimate optimal gripper poses directly [15]. There are two approaches inside of the data-driven method: the target-model-based approach and the model-free approach.

The target model-based approach implements the point cloud data from the sensors to estimate a model of the target object. The target model can be used to eliminate many of the available grasp pose options. Supervised learning (SL) can be used to label objects based on the training data [15], [62]. To save time on data labeling for the SL problem, simulation environments can be implemented to create a database of randomly oriented and mixed samples for training the model. The data-driven target model-based approach is optimal when accurate placement and correct target orientation is critical to receive the reward or when the target objects must be selected from a cluttered scene [15].

The model-free approach removes the model estimation step and instead focuses on optimizing the grasp pose based on the entire range of available grasps. The model-free approach is composed of discriminative and generative techniques. The discriminative technique reviews all the available grasp pose candidates and trains a convolutional neural network (CNN) to select the optimal grasp based on these [21]. The generative approach recommends a grasp configuration based on the orientation of the target. This approach is fundamentally the same as object detection, where the gripper pose is selected based on the orientation of the detected object. An example of this comes from Jiang et al. [63], who implemented orientation rectangles to provide an estimate for the ideal position and orientation of the gripper head. Since multiple grasp candidates are often available, a neural network (NN) is used to train the model to select the best grasp. The model-free approach is most successful when dealing with new targets and soft placement restrictions [15].

The approaches listed work well in combination with other strategies. For example, the agent could be allowed to implement the temporary placement of the object, free from clutter, to allow a better estimation of pose. This approach would be useful when accurate placement is required [21]. An alternative method would be to implement multiple viewing angles by hoisting the item in front of external cameras. Multiple viewing angles could be used to identify the object or determine if a more optimal grasp pose is available [64]. Another practical approach for estimating pose is by implementing focus. Attention focus has been successfully implemented by Gualtieri and Platt [37] via iterative zooming to regions of interest. The use of focus would limit the number of available grasp poses to those seen in the focus region.

## 2.8 Simulation Environment

In the sections above, training in simulation was mentioned as a requirement for some approaches, and a useful tool for others. The benefit of training in a simulated environment is that the agent can perform thousands of training iterations in a short period, without any wear effects on the online device. Online training requires the availability of the robot and human supervision for safety reasons. Further discussion on the selection of simulation environments can be viewed in Section 2.8.

Examples of the difficulties with training in real life can be seen from Levine et al. [41] (2018), who implemented 14 robotic manipulators over two months to collect 800,000 grasp attempts. The accuracy of the grasp operation after this training was 82.5 percent, a lower accuracy than has been achieved for comparable pick-and-place actions when training was completed in simulation.

## 2.9 State of Research

The above sections review the primary choices that need to be made during problem formulation. This analysis section focuses on reviewing and critically discussing the current state of research.

### 2.9.1 State of Research—Complete Pick-and-Place Task

Benchmarks for pick-and-place operations include simulation accuracy, the number of online tests required to train the model, and most importantly, the accuracy of testing. The table below reviews papers published by leading RL robotics researchers to illustrate the strengths and weaknesses of the different implementations. Success rates in each paper are difficult to compare due to the differences in the tasks. The discussion outside of the table gives clarity for each implementation and explains the results.

Fu et al.[17] completed several tasks, with comparable difficulty to pick-and-place, including inserting a peg in a hole, stacking toy blocks, placing a ring on a peg, and more. The approach involved the model learning system dynamics from prior examples, then the completing various manipulation tasks in one shot after goal is defined. The approach did not directly use RL for learning individual tasks by searching through the state space, rather used RL to learn the system dynamics for robotic control. The accuracy of these tasks was high; however, the approach did not incorporate a cluttered scene.

Gualtieri et al.[19,22,55] published several papers on completing the pick-and-place action in a cluttered scene with an assortment of objects. Before this work, Gualtieri et al. [52] first published a

manuscript on pose estimation. The pose estimation work was used in combination with other strategies to complete the entire pick-and-place action. The group completed testing in simulation and online. The accuracy listed represents the online accuracy of grasping a novel object after being trained on objects of similar type. The group found that an items picked from a clutter of objects had significantly lower accuracy than selecting an object in isolation. Additionally, the results indicate that items with complex geometry (cups, bottles) are significantly more difficult to grasp compared to items with simple geometry (square blocks). It should be noted that most manuscripts published on pick-and-place performed grasping on simple target objects in isolation. Training accuracies should always be noted as relative to task complexity.

Popov et al. [62] complete the action of precision stacking of Lego blocks using deep deterministic policy gradient (DDPG) with a robotic arm. The blocks were initially positioned in isolation in the same orientation, which significantly reduced task difficulty. The approach compared a simple single reward and composite, multi-step reward function and proved that single reward had the fastest convergence. The groups asynchronous implementation of DDPG proved that distributed training with asynchronous updates can improve convergence speed.

Mahler and Goldberg [63] applied imitation learning to complete the task of selecting novel objects from a cluttered environment and placing them aside in an organized manner. The learning approach implemented the Dex-Net 2.0 database of point clouds and grasp poses to generate training samples for learning effective Grasp Quality Convolutional Neural Networks (GQ-CNN). The policy learned appropriate NN weights based on replicating the results from the demonstrated samples. Transfer learning was effectively used to apply synthetic training to real world pick-and-place.

Sehgal et al [64] simulated a simple, single block pick-and-place task with the use of DDPG HER whose parameters were tuned with the genetic algorithm. The main contribution to this paper was to show that the GA could be used to minimize the number of training iterations required to achieve comparable convergence to that of DDPG HER without the GA.

Zuo et al. [65] implemented deterministic GAIL approach to complete a simple, single block pick-and-place task. DGAIL implemented an Inverse Reinforcement Learning (IRL) based NN discriminator to develop a reward function based of examples, and then a policy gradient method for learning ideal actions based on the discriminator. During testing, DGAIL was compared to stochastic GAIL and

DDPG to prove the relevance of this approach. The approach yielded much better results than all other methods except DDPG, a more computationally expensive approach.

Chen et al. [74] completed a picking operation where the target object was located in a cluttered environment. The robotic arm could rearrange the cluttered scene if the target was invisible from the camera vantage. The approach was limited due to the model-based nature (model of target always needed), however, it showed excellent performance in finding the target object in cluttered scenario. Placement was not specified but could be implemented since models for the target object are available.

Xiao et al. [66] achieved a high accuracy for pick-and-place task in a cluttered environment using the fetch robot with Parameterized Action Partially Observable Monte-Carlo Planning (PA-POMCP). The system approximated the utility of available actions based on the current belief of the agent about the environment. The approach allowed the agent to change perspective or maneuver items in the scene to find an accurate estimation of the location of the target object. The approach showed significant performance for known items but was not tested on unknown items in new environments.

Liu et al. [67] designed several different robotic pick-and-place tasks, including stacking, sorting, and pin-in-hole placement. A major contribution of this paper was the work around reward shaping to solve complex problems. The approach assigned rewards based on the Euclidean distance between the Objects Target Configuration (OTC) and the Objects Current Configuration (OCM). The closer the configuration of blocks, pins, etc., to the optimal configuration, the more reward allocated to the sample. Additional rewards were allocated for final task completion. Comparison between this approach and a basic linear reward approach shows significant convergence improvement. In addition to reward shaping, this paper compared the MAPPO learning technique to traditional PPO and Actor to Critic (A3C) techniques. MAPPO showed significant performance increase to the other two techniques by directly outputting manipulation signal to the agent.

Mohammed et al. [68] presented the task of organizing a training set consisting of 10 blocks dropped into the workspace using Q-learning. The geometries were not all the same, but the shapes were simple. The accuracies for the pick-and-place operations were only tested in simulation, and not applied to an online agent. Considering the complexity of the task, the accuracy of the pick-and-place and the computational efficiencies are promising.

Li et al. [69] applied a robotic arm to attempt challenging tasks including reach, push, multi-step push and pick and place using the Augmented Curiosity-Driven Experience Replay (ACDER) approach. The pick-and-place operation was free from clutter and only dealt with an individual items at a time. The novelty of the research includes a method of exploration in which rewards are allocated for "curiosity", the exploration of unfamiliar states. Additionally, this approach initiated the robot in states not stored in the replay buffer to improve exploration efficiency. Through testing this approach was proven to be several times more sample efficient than hindsight experience replay (HER).

Pore and Aragon-Camarasa [70] proposed an algorithm for robotic pick-and-place on a simple block placed in isolation on a surface. The solution method involved decomposing the task into *approach*, *grasp* and *retract* segments. Each of these actions was trained independently, then after perfect accuracy was achieved a high-level choreographer (actor-critic network) was used to learn the policy for ordering the behaviors. An end-to-end approach with DDPG + HER resulted in a maximum of 60% accuracy after 10,000 episodes compared to a 100% accuracy after 6000 training episodes with this subsumption architecture. The model was never tested online with a real robot.

Al-Selwi et al. [71] simulated the pick-and-place task on a simple block placed in isolation on a surface using a DDPG approach combined with HER. The approach validated what has been shown in other papers [55] that target object complexity changes task effectiveness. Their works presents little novelty in terms of policy optimization.

Marzari et al.[72] presented simulations and online testing for pick-and-place task on a simple block placed in isolation on a surface by using a DDPG approach combined with HER. By implementing task decomposition, the model was able to achieve 100% accurate performance with simulation and online testing. The major contribution of this approach compared to [70] (see above) was that behavior cloning was not used for training the *approach* subtask, so sample operations were not required. This approach indicates excellent generalizability for novel objects for online robots and much higher sample efficiency than seen in all other approaches.

Anca and Studley [73] completed a simulated pick-and-place operation in which a simple block was picked up by manipulating a robot in 2D by using a twin delayed actor critic network. The result demonstrated operational accuracy without requiring example simulations (unlike [70]), however, showed little advantage over approaches based on DDPG HER [72].

**Table 1: Summary of RL in Robotic Pick-and-Place.**

| Paper | Policy Optimization | Pose Estimation | Sim Package | Success Rate | Strengths (S) and Weaknesses (W) |
|---|---|---|---|---|---|
| Fu et al., 2016 [34] | Value function approach with DP using iterative LQR. Basic two-layer NN for updating priors | Model-based approach which combined prior knowledge and online adaptation | MuJoCo | Simulation Testing: 0.80–1.00<br>Robot Testing: ~0.80 | S: Implemented prior knowledge to reduce the number of samples required<br>W: To create model of system dynamics the NN must be trained on prior examples |
| Gualtieri et al., 2018 [37],<br>Gualtieri et al., 2018 [21] | Value function approach with SARSA. Caffe (NN) used to update the SARSA weights | Model-free data-driven approach which implemented focus. The algorithm allowed for temporary and final placements. Model was trained using 3DNET CAD models | OpenRave (ODE base) | Robot Testing:<br>Picking: 0.88–0.96<br>Placement: 0.80–0.89 | S: Model-free approach with high picking accuracy<br>W: Low placement accuracy and low overall accuracy in testing |
| Popov et al., 2017 [65] | Distributed Deep DPG (DDPG) method with efficient schedule updates and composite reward functions | Model-free data-driven approach. Reward achieved for appropriate grasp selections. Blocks always oriented in the same manner. | MuJoCo | Simulation Testing: 0.955 | S: Proved that asychronous DDPG has higher data-efficiency then other approaches<br>W: No online testing to validate simulation accuracy |
| Mahler and Goldberg, 2017 [66] | Imitation learning approach in which the robot was trained on data from synthesized grasps. Sample grasps were found by using force and torque space | Model based approach used for demonstration synthesis.<br>Model free data driven approach for pose selection during training | Pybullet | Simulated Testing: 0.96<br>Robot Testing: 0.94–0.78<br>(Testing accuracy depends on the number of objects in the cluttered environment) | S: High picking accuracy on cluttered enviroment of unfamilier objects<br>W: Requires models for the wrench space analysis to develop demonstration samples. Intensive programming effort |

| | | | | | |
|---|---|---|---|---|---|
| | analysis and knowledge of object shapes and poses | | | | |
| Sehgal et al., 2018 [67] | DDPG with hindsight experience replay (HER). Parameters tuned using the genetic algorithm | Model-free data-driven approach | MuJoCo | Simulated Testing: ~0.90 | S: Effectively implemented HER for faster convergence<br>W: No improvement in overall task completion performance. No online testing to validate simulation accuracy |
| Zuo et al., 2019 [68] | Deterministic generative adversarial learning (DGAIL) which implemented a policy gradient generator and discriminator trained with IRL (an actor-critic approach) | Data driven approach in which action selection was based on the difference between the demonstrated and generated policy | MuJoCo | Simulated Testing: 0.90 | S: DGAIL had faster convergence then several other RL techniques<br>W: DGAIL was less stable, and less accurate then other modified DDPG techniques |
| Chen et al., 2019 [69] | Two deep Q-Networks (DQN) for pushing and grasping. Mask Region convolutional NN (R-CNN) for object detection | Model based approach where the target object was mixed in clutter. RBGD image was used for target object detection. The scene was rearranged if the target is invisible from the sensor's perspective | V-REP simulation platform | Simulated Testing: 1.0<br>3 different scene options were tested. Each seen had the target in a more/less hidden location | S: High target location accuracy. Approach proves that rearranging the enviroment can improve results<br>W: Approach was only applied for known target objects. No online testing to validate the simulation accuracy |

| | | | | | |
|---|---|---|---|---|---|
| Xiao et al., 2019 [16] | Parameterized Action Partially Observable Monte-Carlo Planning (PA-POMCP) | Model based approach which implemented known models from a benchmark model set | OpenRAVE & Gazebo | Robot Testing: 1.0 | S: High pick-and-place accuracy<br>W: Model based approach. Grasp poses are pre-defined so the primary task is model recognition. This technique significantly simplifies the problem |
| Liu et al., 2020 [11] | PPO with actor output as manipulation signal (MAPPO) | Data driven approach which extracted target object pose from RGBD sensor | Gazebo | Robotic Testing:<br>Relative improvement of MAPPO to PPO and A3C shown to be >30%.<br>Final accuracy not given | S: Compared various learning and reward shaping approaches<br>W: No final pick-and-place accuracy given. Difficult to guage the performance improvement |
| Mohammed et al., 2020 [13] | Value Function approach with Q-Learning. CNN used to update Q-learning weights. Model pre-trained on Densnet-121. | Data-driven approach. Grasps were generated by using a CNN to identify available poses from a RGB image | Vrep (ODE base) | Simulated Testing:<br>Picking: 0.8–1<br>Placement: 0.9–1 | S: High placement accuracy considering the absense of model for target object. Short training time<br>W: No online testing to validate the simulation accuracy |
| Li et al., 2020 [70] | DDPG approach with goal-oriented and curiosity driven exploration and dynamic initial states | Data driven approach which used RGBD images to determine object and goal positions | MuJoCo | Simulated Testing: 0.95<br>Robotic Testing: 0.85 | S: Comparison to several other sampling and learning techniques validated the sample efficiency for this approach<br>W: Deployment on robot showed a reduced overall effectiveness |
| Pore and Aragon-Camarasa, 2020 [71] | Hierarchical RL in which the task was broken into simplified multi-step behaviors with the subsumption architecture (SA). Behavior cloning was used for low-level behavior | Data driven approach in which grasp poses were trained with behavior cloning. Target objects were consistent which means that advanced pose estimation was not required | OpenAI Fetch environment with MuJoCo | Simulated Testing: 1.00 | S: Validated that the subsumption architecture improves entire task performance significantly compared to end-to-end approaches<br>W: Technique required behavior cloning samples, and a human for problem breakdown. No online testing to validate simulation accuracy |

| | | | | | |
|---|---|---|---|---|---|
| | training. Actor-Critic technique was applied for high level task completion | | | | |
| Al-Selwi et al., 2021 [8] | DDPG with HER | Model based approach in which RGB image was used to determine bounding box and rotation angle | MuJoCo | Simulated Testing: 0.502–0.98 (depending on target geometry) | S: Validated that HER DDPG can be used with vision feedback to improve accuracy<br>W: CAD models required for pose selection. No online testing to validate simulation accuracy |
| Marzari et al., 2021 [72] | DDPG with HER and task decomposition | Data driven approach in which grasp poses were learned from a single target object with simple geometry | MuJoCo | Simulation and Robotic Testing: 100% | S: Proved that the DDPG approach with HER can perform excellent task completion accuracy by using task decomposition rather then end-to-end training<br>W: Approach assumed human involvement for task decomposition |
| Anca and Studley, 2021 [73] | Twin delayed hierarchical actor critic (TDHAC) which broke task into high- and low-level goals | Data driven approach which only implemented a 2D motion for picking the action | Pybullet | Simulated Testing: 100% | S: Confirmed that the hierarchical approach improves convergence<br>W: Approach showed no significant improvement over DDPG with HER. The motion was limited to 2D. No online testing to validate simulation accuracy |

### 2.9.2 State of Research—Pick-and-Place Subtasks

Although there are many publications available presenting different layers of contribution to the study of pick-and-place operations with RL in robotics, many do not include key elements of the action sequence. Examples include the literature on grasp selection [74,75] or precision placement [76]. To keep the current paper as a comprehensive survey, the key examples for the pick-and-place subtasks drawn from the literature are listed in Table 2.

Finn et al. [76] developed an IRL algorithm that could be applied to various tasks in different environments to learn the policy and reward function. The tasks completed in simulation included peg insertion in a hole to a depth of 0.1m. The robotic model could complete this task with high precision after training with ~30 examples. Of the real robotic tasks completed, the task of placing dishes in a rack was the closest task completed to pick in place. After 25-30 samples of human teaching, the dish placement action could be completed with perfect accuracy. This manuscript gives an excellent indication of the power of IRL, however, to apply this algorithm to the full pick-and-place action sequence, further extension is required to include the pose estimation step for "picking".

Kalashnikov et al. [75] developed a custom Q-learning approach focused on generalizability and scalability for unfamiliar environments. The task required the picking of random objects from a dense clutter and raising the object above the clutter. The training was online to incorporate real-world physics and all properties which were unmodeled in the physics simulation. To apply this algorithm to the full pick-and-place action sequence, the research requires additional task programming to include precision placement.

Wu et al. [77] designed a custom robot which implements a 3 finger gripper to pick objects out of a cluttered containing 2-30 cluttered objects. The study proved that attention improved the picking action in cluttered scenes by ~20% and implementing a 2-finger gripper improved object selection by ~45%. To apply this algorithm to the full pick-and-place action sequence, the research requires additional task programming to include precision placement.

Deng et al. [78] designed a custom robotic hand to be used for picking objects out of a cluttered scene using a series of fingers and a suction rod. The mode of selection allowed for a simple convolutional NN structure to be used for selecting ideal locations for grasp. The grasp failure occurred if 3 consecutive grasp attempts were not completed sequentially. Low-test accuracy limits this approach for real world application.

Beltran-Hernandez et al. [79] performed the task of picking simple geometry items in simulation training (cylinder, cube, and sphere). Accuracy for simulation testing on random objects in simulation (duck, nut, mechanical part) was high, especially considering the major differences between the trained models and the test samples. The approach was model based, so it could be easily extended to incorporating high precision placement.

Berscheid et al. [80] completed the task of manipulating and picking a variety of objects in a cluttered scene by using Q-Learning. Results indicate high picking effectiveness and generalizability. Training was completed online, without the use of a simulation model. The task definition did not include placement, but the picking action included a high level of difficulty. To apply this algorithm to the full pick-and-place action sequence, the research requires additional task programming to include precision placement.

Kim et al. [81], complete the task of picking up unknown targe objects in a highly cluttered scene by using disentanglement of image input. The goal of this approach was to reduce the computational difficulty of performing a pick action compared to value-based approaches. A key contribution of this paper is showing the value in creating a low dimensional state representation of the target object with the use of autoencoders. Additionally, this paper investigated the value of implementing disentanglement to allow simple scene recognition to guide behavior. The disentanglement approaches in focus include attention, separation of internal (robot) and external (scene) information, and separation of position and appearance in the scene.

Many other papers exist, which can be applicable for subtasks of pick and place, however most of them could not be easily extended to the entire operation. For more samples of individual grasping techniques for the pick action, it would be helpful to review [12].

**Table 2: Robotic Pick and Place Subtasks.**

| Paper | Policy Optimization | Pose Estimation | Sim Package | Success Rate | Strengths (S) and Weaknesses (W) |
|---|---|---|---|---|---|
| Finn et al., 2016 [25] | Combined relative entropy IRL and path integral IRL with sampling using policy optimization | Algorithm did not incorporate the picking operation so pose selection was not required | MuJoCo | Robotic Testing: Placement: 1 | S: Perfect placement accuracy<br>W: Technique did not incorporate picking action. Online training samples are required |
| Kalashnikov et al., 2018 [12] | QT-Opt, a modified Q-learning algorithm that implemented off-policy training and on-policy fine-tuning | Data-driven approach. Strategy implemented dynamic closed loop control with RL to solve the grasping task. | Bullet Physics Simulator | Robotic Testing: Picking: 0.96 (with object shifting) | S: High picking accuracy in a cluttered environment with unknown objects<br>W: Time-consuming online training required |
| Wu et al., 2019 [75] | Pixel attentive PPO | Data driven approach which learned optimal grasps through trial and error. Pixel attentive method cropped image to focus on local region containing ideal grasp candidates | Pybullet | Simulation and Robotic Testing: 0.911–0.967.<br>Accuracy changed based on the density of the clutter | S: Excellent alignment between simulation and real enviroments. Useful data presented which compared camera orientation and grasp accuracy<br>W:Three finger gripper may contribute to high picking success rate. Most standard grippers are planar (2 fingers) |
| Deng et al., 2019 [76] | Deep Q-Network (DQN) implemented to select actions based on affordance map | Data driven model-free approach in which lifting point candidates were selected based on affordance map showing “hot spots” or ideal grasp | V-REP | Robotic Testing: Picking: ~0.71 | S: Novel robotic arm design effective for selection of randomly oriented objects in a clutter.<br>W: Picking operation success rate was only 11% better then random grasp actions (very poor) |

| | | | | | |
|---|---|---|---|---|---|
| | | candidate locations in the RGBD image. | | | |
| Beltran-Hernandez et al., 2019 [23] | Guided policy search, with image input and grasp pose as output. | Model-based approach. The agent was trained on basic shapes and then tested on complex geometries | Gazebo (ODE Base) | Simulation Testing:<br>Picking: 0.8–1 | S: Approach shows significant controbution to the space by showing effectiveness of using model based techniques for grasping unfamiliar objects<br>W: No online testing to validate simulation accuracy |
| Berscheid et al., 2019 [24] | Modified Q-learning. Upper confidence bound for off-policy exploration. | Model-free approach which implemented NN to generate action values | Online Testing. 25,000 grasps trained in 100 h | Robotic Testing:<br>Picking: 0.92 (with object shifting) | S: High picking accuracy and good recognition of required shifts<br>W: Time-consuming online training required |
| Kim et al., 2020 [77] | A2C with state representation learning (SLR). Involves learning a compressed state representation | Data driven approach which implemented raw image disentanglement | Pybullet | Simulation Testing:<br>Picking: 0.72 | S: Computationally affordable grasping action<br>W: Low picking accuracy. No online testing to validate simulation accuracy |

### 2.9.3 Summary

The literature discussed above shows that significant progress has been made towards developing a RL approach which can be applied to robotics, however there are several issues with each approach. Additional research is required for validating simulations, reducing the loss in accuracy from crossing the reality gap, reducing the amount of human intervention, and training the positional feature extractor.

Only 40% of the literature analyzed above includes results for online testing. Most of the research only included data on testing the models in simulation, wherein the dynamics are completely dependent on accurate modeling. To validate that the various methods are accurate and implementable in real life, further testing is required.

Of the manuscripts analyzed, several had robotic testing accuracies which were significantly lower than simulated accuracies [34], [70]. Deviation between simulated and testing accuracies indicate that issues on bridging the reality gap require further investigation. As previously noted, training robotic agents online requires extensive time and equipment availability [41]. For RL to be applicable for robotic pick-and-place, simulations must be developed which can be accurately transfer to the real world.

The approaches which achieved robotic test accuracies high enough to be considered for application in industry, such as [66], [72] required extensive human intervention. Task decomposition significantly speeds up learning and improves overall accuracy, however this method requires human intervention. The research must be extended to include methods for automatic task decomposition, or to develop strategies for task decomposition which do not require significant effort.

Of the strategies listed, most implemented vector based positional inputs. For these problems, the agent is given positional coordinates for the target positions rather then images, which significantly reduces problem complexity. Of the strategies listed which implemented images, Yolo, or similar pre-weighted networks were used to extract positional information. For the application for robotics to RL, image-based control must be further trained and tested.

# Chapter 3: The Simulation Study

## 3.1 Introduction

The section below presents the simulation environments developed, the codebases applied, and the hyperparameter tuning framework implemented to solve the robotic RL problem. Additionally, the methodology behind the custom *Semi-Supervised RL* approach is reviewed in detail.

Some parts of the thesis are published in the journal: A. Lobbezoo, Y. Qian, and HJ. Kwon, "Reinforcement Learning for Pick and Place Operations in Robotics: A Survey." Robotics 2021, 10, 105. https://doi.org/10.3390/robotics10030105

## 3.2 Gym – The Learning Framework

Gym is a standard API, designed by OpenAI Inc., which was created to serve as a template for solving RL problems. The Gym [78] toolkit provides open-source environments for developing and comparing reinforcement learning algorithms (A2C, PPO, etc) and testing these algorithms for various tasks (cartpole, pendulum, etc). Gym enables the implementation of several physics engines, including Pybullet, DART and MuJoCo. Examples of research which has implemented Gym for simulations of robotic reach, grasping, and/or pick and place include: [67], [68], [70]–[72].

### 3.2.1 Learning Methodology

Gym API has many functions which have been developed to enable efficient communication between the RL algorithm and the simulated environment. The methodology for implementing Gym can be seen in detail in Appendix A. At its foundation, Gym follows the MDP learning cycle depicted in Figure 6. The primary difference between the Gym framework and the standard MDP is that Gym returns a Boolean describing whether the target position is reached, along with the state and reward information after each step.

The Gym based 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. Each timestep in the environment, the agent can move for 1/240s in the simulation. For the robotic RL framework, each episode begins with the agent initialized in a standardized home position (Appendix A, Figure 36), with the target object initialized 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.

The goal of the gym framework for this project, is to enable the robotic agent to learn an action plan and an optimal robotic end-effector motion path for various tasks. The agent must learn to relate the input state information from the environment to the ideal action commands for a series of timesteps. For the pick-and-place task, the agent must learn the following action series. The agent must learn to follow a short path to the target object, to actuate the gripper to grasp the target block, to follow a path to lift the end-effector and the target block above the placement block level and to transport the target block to the target position. The short paths the agent follows for each trajectory are often based on the difference between the end effector and the target position, multiplied by some gain to speed up the action per step. The robotic agent must learn to execute these paths quickly to maximize reward, while not colliding with the target block.

The RL learning framework is designed this way, so the agent can learn to perform any task, provided the proper inputs. For example, a task which the RL agent could learn in the future, would be to tighten a bolt. For this problem, the RL agent could be provided with the state information (location and orientation of the bolt, nut, and EE). Through exploration in the environment and by learning how to relate the state information to rewards, the agent could learn how to thread the nut onto the bolt.

The key difference between this framework and existing traditional control approaches, is that the RL agent only requires human input for determining goal states. The RL agent does not require human support for action sequence planning or motion path creation, which makes this implementation very generalizable.

### 3.2.2 Physics Engine

The three environments which are commonly used for robotic representations are Gazebo, Pybullet and Gazebo. Renders from the three environments can be viewed in Figure 8. Each environment requires a physics engine for simulating gravity, friction, and dynamics. Of the packages listed, each comes with strengths and weaknesses.

Gazebo [79] is a highly functional package which is designed to interact with Robot Operating System (ROS). Gazebo is useful when creating a model to serve as a digital twin to a real-world robotic agent, and to preview path plans and tasks. As shown in Appendix B, Figure 37, the Gazebo-ODE environment

renders realistically, and the kinetic sensor produces a clear image. Since Gazebo works over ROS, the Gazebo control communication can be developed to perfectly replicate the real-world robot. Gazebo allows for the integration of various physics engines (ODE, Bullet, and DART). Traditionally Gazebo implements ODE [79]. A few examples of research which has implemented Gazebo for physics simulations of robotic reach, grasp and/or pick-and-place include: [11], [16], [23].

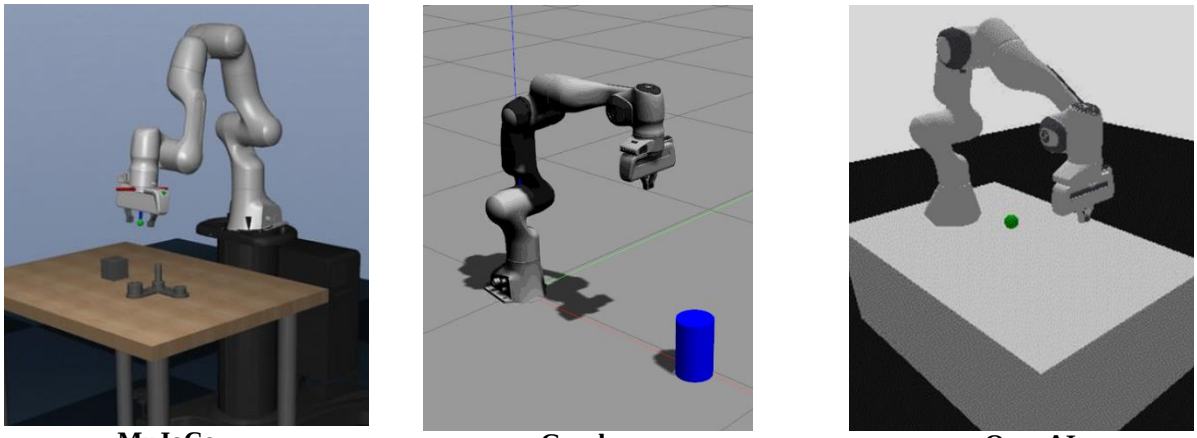
The main difficulty implementing Gazebo for simulating the physics environment, is that it requires the integration of custom controllers, camera nodes, and URDF robotic agents inside the Linux-Ubuntu ROS interface. While running the Gazebo environment, it was noticed that compared to MuJoCo and Pybullet, Gazebo was computationally expensive, and it required a significant portion of GPU usage to render the environment. During RL training, several robotic arms must be rendered in parallel to speed up exploration, which would cause significant lag on the GPU. GPU usage should be preserved for network updates during backpropagation, so Gazebo was ruled out.

MuJoCo is an intensive physics engine with the highest solver stability, consistency of results, accuracy of calculations, and energy conservation during actions compared to other physics environments [80]. The physics engine at the backbone of simulation packages has a large influence on the effectiveness of the environment, so MuJoCo's success here is crucial. Examples of research which has implemented MuJoCo for physics simulations of robotic reach, grasp and/or pick-and-place include: [8], [34], [65], [67], [68], [70]–[72].

Unlike Pybullet or ODE, MuJoCo does not accept regular URDF robotic files, but instead requires modification and compilation into a native MJCF robotic definition. MuJoCo's usage was limited to users with purchased licenses until October 18, 2021[81], when Deepmind Acquired MuJoCo and opensourced it for the RL community. Due to the licensing issues (until recently), difficulties with implementation, and the poor community supporting the MuJoCo environment, MuJoCo was not implemented for this project.

PyBullet [82] 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 [93]. PyBullet does not have pre-built ROS communication;, however, custom ROS nodes can be written to allow for ROS integration. The package is simple to modify and has excellent documentation. Examples of researchers that have implemented PyBullet for physics simulations of robotic reach, grasp, and/or pick-and-place include: [66], [73], [75], [77].

Due to the ease of modification and implementation of the PyBullet environment, and simplicity of parallelization for training [82], PyBullet was selected as the primary environment for training the RL agent. The base PyBullet panda model implemented for this project was cloned from the Github repository created by Gallouedec, et al. [83], and modified to suit the tasks for which this RL algorithm was tested.



**MuJoCo** **Gazebo** **OpenAI**

**Figure 8: Simulation environments**

## 3.3 Custom PyBullet Tasks

To make custom PyBullet environments, several interacting PyBullet classes had to be modified. The details on how each class had to be modified for the custom tasks are explained in Appendix C. The main modifications of the base environment included the addition of depth image rendering, reward shaping, target object block instantiation (for pick-and-place), episodic termination, and modifications to friction coefficients and maximum joint forces.

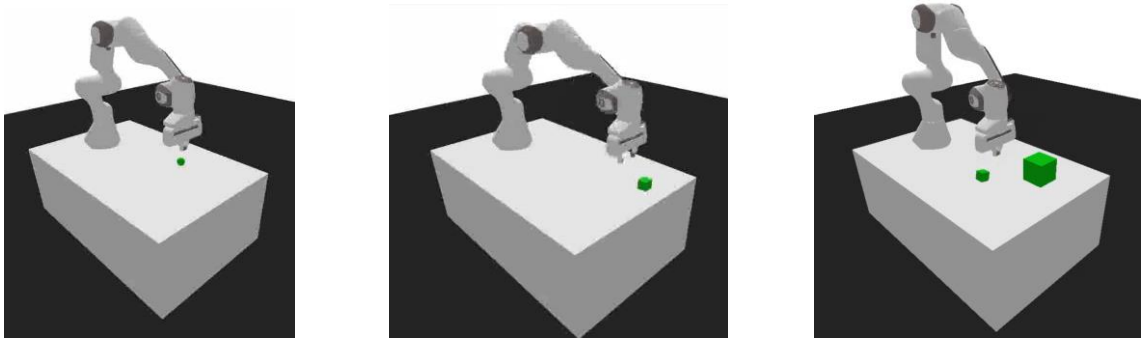
The three custom tasks created for testing the RL robotic system are panda-reach, panda-grasp, and panda-pick-and-place. The task environments can be viewed in Figure 8. For each task, end effector-based control strategies with pre-built IK packages were implemented. Joint base control was considered; however the goal of this project was to learn high-level task planning strategies, so to reduce the difficulty in training, end effector-based control was applied.

For the panda-reach problem, the agent must learn to provide XYZ input action commands to the robotic arm. The reach target object is stationary and penetrable, so the gripper cannot cause changes

in the target object position. The task ends when the center of the end effector is located within 1mm of the center of the target object.

The panda grasp problem requires that the agent actuates the end effector coordinates and the gripper width. The reach target object is not stationary and is impenetrable, so any collisions between the end effector and the target block will cause the block to move and/or slide off the table. The task ends when the center of the end effector is located within 1mm of the center of the target object with the gripper open.

Like the grasp problem, the panda-pick-and-place problem requires the agent to actuate the end effector coordinates and the gripper width, without colliding with the target object or the placement block. The task is completed when the panda grasps the target object, lifts it up, and places the target at rest on top of the target block. The mass of the placement block was increased to 50kg, so that the agent would not push the target block off the table during exploration.



**Closed gripper, end effector control [x,y,z] Target stationary and penetrable**

**Open gripper (gripper width W), end effector control [x,y,z, W] Target object is dynamic and impenetrable**

**Open gripper (gripper width W), end effector control [x,y,z, W] Target object is dynamic and impenetrable. Placement block on right**

**Figure 9: Various panda environment configurations**

## 3.4 Reward Structure

As discussed in Section 2.6, reward shaping plays a critical role in solving the RL problem. The two variations on rewards implemented for this project are dense (heterogeneous) and sparse (homogeneous).

With the standard sparse reward scheme, the agent receives a reward of -1 for all states except the final placement state. The agent has difficulty solving complex RL problems with this reward scheme due to the Monte-Carlo (random) nature of this approach. For example, the pick-and-place task requires the agent to pick up the target block, transport the block, and release the block at the correct location. This sequence of actions is not realistic for a random search, as the agent may never learn to solve the problem.

For such a complex task, the agent requires a reward function with a dense reward structure. A common dense reward function implemented for RL, is the heterogeneous reinforcement function [51]. In this approach, the reward improves as the agent executes the task correctly. With the incremental reward implementations for reach and grasp, the reward increases proportionally to the distance between the gripper and the target block. For the pick-and-place task, the reward is further increased as the target block is lifted, and the distance between the target block and the placement position is decreased.

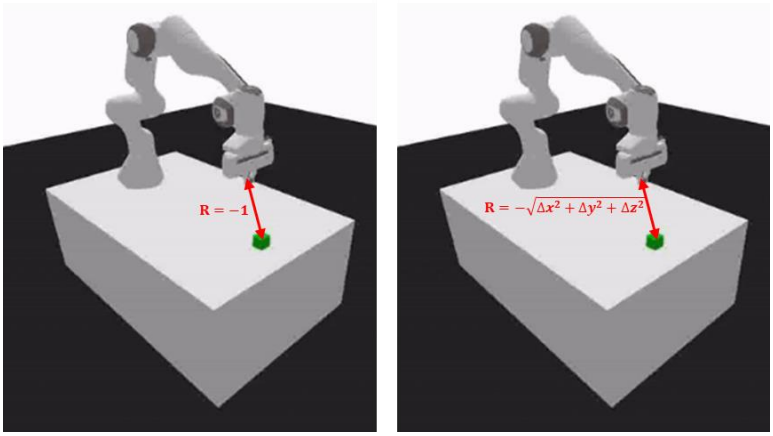



**Figure 10: Sparse (left) vs dense (right) reward shaping**

The difference between the sparse and dense reward functions can be seen in Figure 10. As shown, the rewards for both functions are usually negative. The only positive reward which the agent receives is provided in the final state in which the task is terminated. For the reach, grasp, and pick-and-place task several steps are required, so the cumulative reward over each episode will be negative.

For this project, both dense and sparse rewards are explored for the vector-based reach and grasp tasks. Due to the complexity of pick-and-place and image-based reach and grasp tasks, only sparse rewards are implemented.

## 3.5 Positional Inputs

Two forms of RL can be tested with the custom Gym environment: vector-based position feedback or image-based position feedback. Most papers published on the topic implement vector-based position feedback, where the agent is fed the state position of the gripper (x, y, z, Vx, Vy, Vz, pitch, roll, yaw, etc) and the target object/objects (x, y, z, Vx, Vy, Vz, pitch, roll, yaw, etc).

The issue with this type of problem is that the agent is essentially "cheating" as it knows the target position of the object, which in the real world would require some form of measurement or calculation. The alternative to the vector-based positional feedback task is a task with an image input. The image-based task requires the RL agent to train a CNN feature extractor to return valuable positional information from a compressed image.

The feedback given to the robotic agent during training significantly affects problem difficulty. The difference between the vector-based and image-based positional feedback problems are reviewed below.

### 3.5.1 Vector-based Tasks

For the panda-reach problem, the agent is provided the end effector and target XYZ coordinates after each step. The agent must learn to provide $\Delta x \Delta y \Delta z$ input action commands to the robot. The short 6-value input vector and 3-value output action command is relatively simple for any RL agent to learn.

Like the reach problem, the agent in the panda-grasp problem is provided the end effector coordinate, and the target coordinates. Additionally, the agent receives the gripper width, gripper orientation, target orientation, and target velocity. The difficulty of the grasp problem is significantly higher than reach, as the agent must learn the relationship between the 13-value input vector, 4-value output action command, and the reward. The agent must learn to approach the block by following specific paths which requires extensive exploration and training.

Like the grasp problem, panda-pick-and-place requires the agent to actuate the end effector coordinates and the gripper width. The agent is provided the same values about the end effector and target as well as the coordinates and orientation of the placement block. The difficulty of the pick-and-place problem is significantly higher than both reach and grasp, as the agent must learn the relationship between the 19-value input vector, 4-value output action command, and the reward. The agent must learn a similar

approach strategy as for the panda-grasp task, while also learning grasp, lift and transportation action sequences.

### 3.5.2 Image-based Tasks

As an alternative to the vector-based task, image-based feedback can be implemented to make the problem realistic for real world applications. The image-based approach to solving this problem

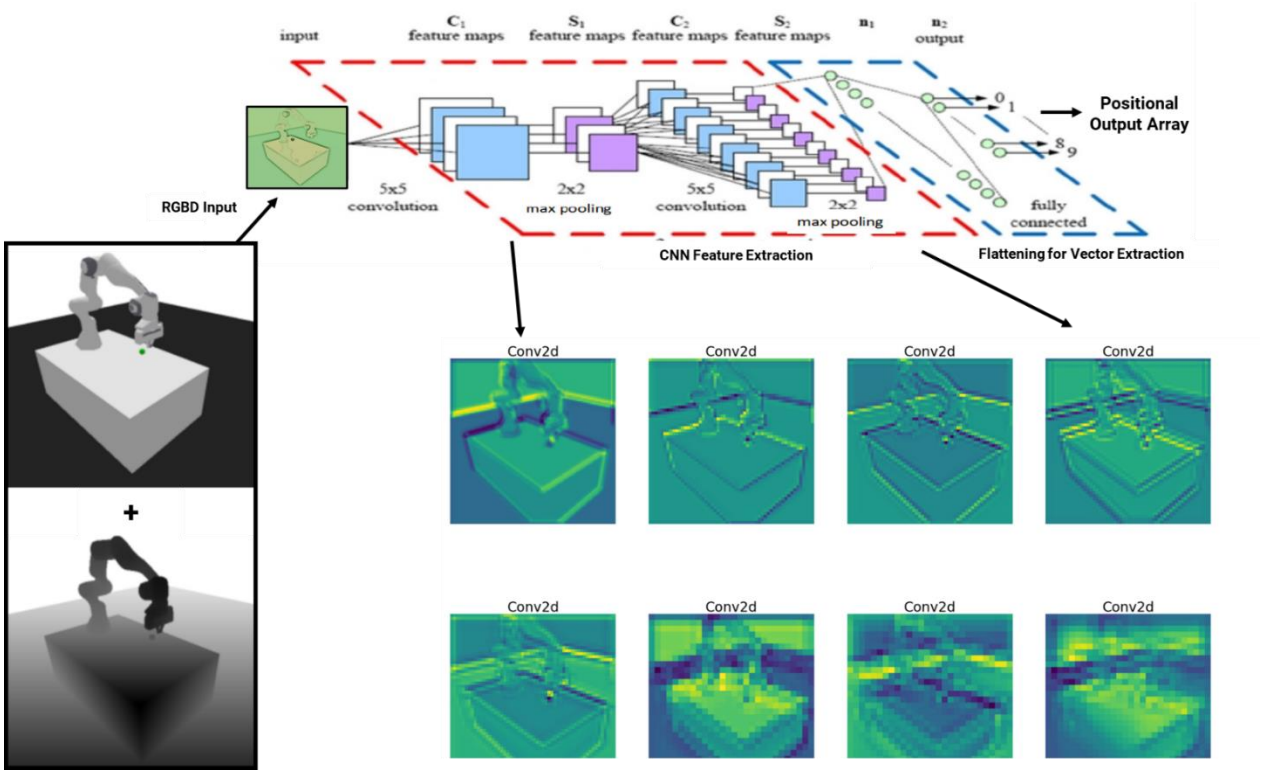



**Figure 11: Image-based feature extraction**

involves feeding the agent an image of the environment, then allowing the agent to learn the relationship between images, actions, and rewards directly.

The image-based approach can be implemented without creating a new environment, by applying a custom gym wrapper which modifies the state feedback. Each time the step and reset methods are called in the gym environment, the custom wrapper renders depth and RBG images. The RBG image is compressed to greyscale to reduce the size of the file and combined with the depth image to form a 100x100x2-channel image for training. The images implemented for training are rendered from a front

view to improve training performance. The pipeline for rendering, compressing, and feeding into the CNN can be seen in Figure 11. Samples of images fed into the network can be viewed in Figure 12.

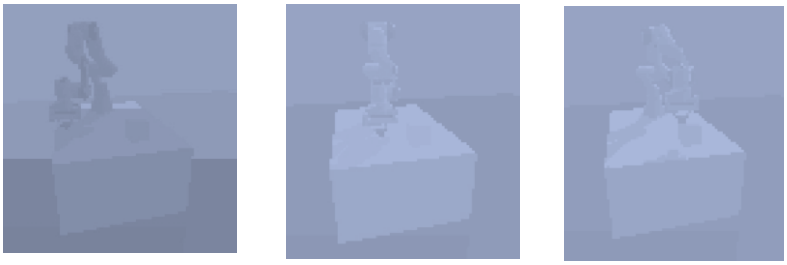


**Figure 12: Greyscale combined with depth image**

### 3.5.3 Combined Image and Vector-based Tasks

For the semi-supervised approach (3.7) a combined image and vector-based solution was tested. In this approach, the compressed image rendered from the environment is combined with the end effector position vector calculated with forward kinematics. The combination approach is intuitive, since the more information fed into the network about robotic state, the less information the feature extractor must learn from the image.

## 3.6 The RL Training Codebase

RL codebases are notoriously difficult to code from scratch; however, there is a strong open-source machine learning community. To implement RL for robotics, the efficient approach involved leveraging and modifying existing codebases. The section below summarizes the investigated codebases, and the hyperparameter tuning framework applied to improve training. For this project, the primary codebases implemented include Stable-Baselines3 (SB3), Optuna, and PytorchCV.

### 3.6.1 Stable Baselines3

The primary RL codebase implemented for this project was SB3. Custom PPO and DDPG algorithms were written and tested; however, this code underperformed compared to benchmark SB3 algorithms which have been optimized for performance.

SB3 was selected as the primary codebase for testing PPO and SAC, because it was easily modified to accommodate the custom Panda environment, and had pre-built parallelization, visualization, and GPU

integration features. Additionally, SB3 had excellent supporting documentation, many functioning examples, and is built on PyTorch.

Table 3 compares the primary open-source codebases implemented by the RL community. The table clearly indicates that SB3 has a slight advantage over RLlib due to the additional tutorials, pretrained models, and PyTorch backbone.

**Table 3: Comparison of RL Codebases [84]**

| | ***SB3*** | ***RLlib*** | ***Tianshou*** |
|---|---|---|---|
| ***Backbone*** | PyTorch | PyTorch/TF | PyTorch |
| ***Documentation*** | Excellent | Excellent | Good |
| ***Tutorials and Examples*** | Excellent | Good | Poor |
| ***Last Commit*** | <1 week | <1 week | <2 weeks |
| ***Pretrained Models*** | Yes | No | No |

### 3.6.2 Optuna

All ML methods rely on a series of hyperparameters to be set to allow for accurate results and fast convergence. For all ML problems, network hyperparameters, such as number of epochs, batch size, choice of activation function, network hidden layers and width, learning rate, and optimization algorithm, all play a role in determining update speed, stability, and convergence.

Due to the complex nature of RL and robotic control, hyperparameter tuning is critical. RL problems have a range of additional hyperparameters not required for supervised or unsupervised learning, such as number of steps (time between updates), gamma (discount factor), entropy coefficient (confidence parameter, encourages exploration), and many other method dependent hyperparameters.

For RL problems, on average there are 8 hyperparameters each with a minimum of 6 values. With this many parameters, grid or random search are unreasonable. Grid search would require ~1.7 million hyperparameter combinations; random search, although more sample efficient grid search, would not guarantee optimality.

An alternative to these traditional searches is to implement an intelligent search to ensure the entire range of hyperparameters is surveyed and the optimal solution is found. Optuna is one of the leading packages for intelligent hyperparameter tuning. Optuna is compatible with the current machine learning libraries including scikit-learn, PyTorch, Tensorflow, Keras and others [85], [86]. For this research, Optuna was implemented due to its usability, and compatibility with SB3.

#### 3.6.2.1 Optuna Search

Optuna implements complex methods for improving training. Examples include the Tree-structured Parzen Estimator search algorithm and the median pruning strategy.

The search algorithm implemented for this project is the pre-built Gaussian-Process-based Tree-Structured Parzen Estimator (TSPE). This algorithm initially randomly samples hyperparameter combinations to explore the space of hyperparameters. After the boundaries of the space are explored, the model creates a probability distribution to represent the probability for each hyperparameter given an objective score. Hyperparameters are selected based on which values are expected to return the highest expected objective score [87].

The pruning strategy involves checking the progression of the optimizer to allow for early termination if unsatisfactory results are achieved from a particular combination of hyperparameters. Pruning samples reduces the number of timesteps wasted on poor-performing hyperparameter combinations. The median tuner was implemented for this problem. This pruner exits trials if the intermediate results perform worse than the results of the median of the other trials at the corresponding timestep. The pruner was set to prune after 1/3 of the steps were completed for each hyperparameter trial.

#### 3.6.2.2 Implementation

The process of creating an Optuna study involves developing an objective function, then iterating through different trials (combinations of parameters) until the optimal hyperparameters are found. For the panda robot, the objective function contained the "reward" metric which was to be maximized while training [85], [86]. The two types of hyperparameters which affected the reward metric during the hyperparameter search were the algorithm and network parameters.

One of the primary factors which affected policy training was the algorithm hyperparameters. The training of both PPO and SAC were improved by modifying the learning rate, entropy, batch size, activation function, number of episodes, and gamma. Additionally, each algorithm had its own hyperparameters which affected training. PPO specific hyperparameters included max gradient norm and value function coefficients. TD3 hyperparameters included gradient steps and action noise.

The other major factor which affected policy training was the network parameters. The network parameters had to be changed based on problem complexity, positional input, etc. During exploration,

network parameters were improved by modifying the number of layers, neurons, convolutions, and so on.[88], [89].

### 3.6.2.3 Optuna Outputs

After each Optuna trial is completed, the results of the trial can be viewed in the Parallel Coordinate Plot (PCP) and the Hyperparameter Importance Plot (HIP). PCPs are implemented as a tool for comparing hyperparameters, to learn how specific hyperparameter ranges affect training accuracy. Alternatively, HIPs are implemented to determine which hyperparameters cause the most significant impact on training results. Figure 13 depicts example PCP and HIP plots from an Optuna trial.

Each line on the PCP must be understood as a trial completed during hyperparameter tuning. The first column on the PCP, objective value, indicates the average cumulative reward experienced in the test environment for the policy after 100,000 training time steps. Higher objective values indicate better performance for a particular combination of hyperparameters.

All columns to the right of objective value represent specific hyperparameters which have been tuned. Each hyperparameter has a range which is explored during the trial. Some ranges are integers (i.e. number of epochs), some are floats (gamma, learning rate, etc.) and some are integers representing powers (batch size ranges from 128 ($2^7$) to 512 ($2^9$)).

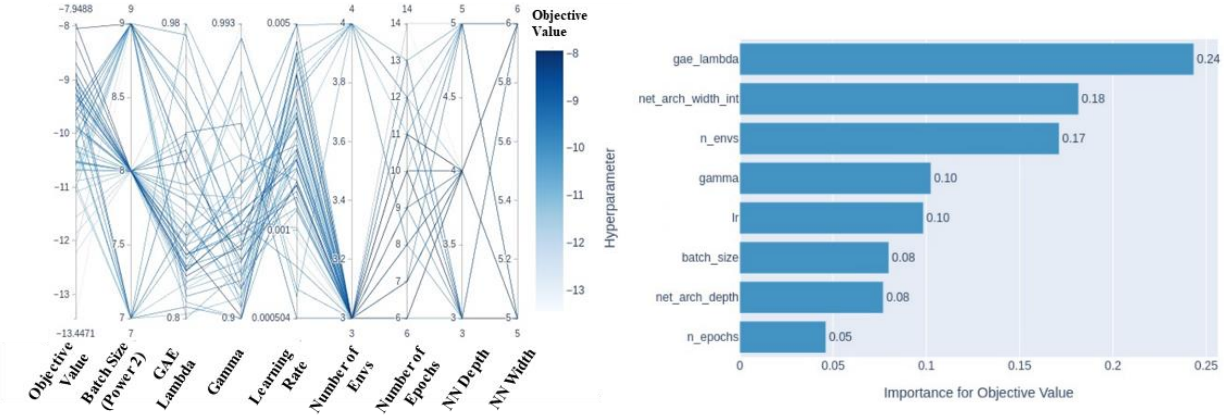



**Figure 13: Example Parallel Coordinate Plot (PCP) and Hyperparameter Importance Plot (HIP) from the training of PPO for vector-based panda-grasp with dense rewards**

The PCP is a useful tool when comparing the performance of specific ranges of hyperparameters, and the relationships between hyperparameters. From the PCP shown in Figure 13, 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. Additionally, the PCP indicates that 3 environments solving in parallel perform better than 4.

For some trials, the relationship between the objective value and specific hyperparameters shown in the PCP are not clear. For such cases, the HIP should be implemented to determine if the hyperparameters of concern have a significant impact on training. Hyperparameters with little impact on training can be fixed once a realistic value for the hyperparameter is found from literature or from hyperparameter tuning. The results section (Chapter 5) presents PCPs and not HIPs, since values which have little effect on training performance are removed during early rounds of hyperparameter tuning.

## 3.7 Semi Supervised RL

One of the major difficulties with RL is the inadequate performance which traditionally results from tasks with image-based feedback. One of the major issues with image-based RL is the extensive training time required for the CNN feature extractor, and the inadequate performance with results after CNN feature extraction training.

The CNN network implemented for the image-based tasks is a shallow 3-layer network, similar to the networks tested by Mnih et al. [90], [91] for deep Q-network training. Shallow networks are traditionally implemented for image-based tasks due to the design of the RL network updates. In RL, the agent must learn to implement the reward signal to tune the Feed Forward Neural Network (FFNN) policy which relates states and actions. For image-based tasks, the reward signal is also implemented to update the weightings of the CNN feature extractor which extracts a vector of states from a BW/D image. Implementing the same reward signal to train the FFNN and CNN in series is an intensive task.

For deep networks with high accuracy in classification challenges (SqueezeNet, DenseNet, ResNet, etc.), large amounts of training data are required. Training networks with hundreds of thousands of nodes is not a realistic task for any existing RL algorithms as it would require the FFNN to be tuned in series. The CNN would not have deterministic classes or vector positions to learn from, but rather would be updated based on the backpropagation error beginning at the FFNN.

To improve the process of extracting vectors of positional information from a CNN network, a novel method *Semi-Supervised Feature Extractor Based Reinforcement Learning* was tested for this problem. This method involved pre-training a highly optimized network (SqueezeNet) to extract positional

information from the image, before training the RL task-specific policy. The theory with this approach is that the network can be pre-trained to extract positional information from the image. After the network is trained, the layers can be frozen, so no network updates are required during PPO or SAC policy training.

Pre-training the position vector extractor is relatively simple. Target blocks are placed in the environment at random positions. BW/D images are captured and stored, in parallel with the vector positions. The network is trained based on these pre-sampled images and position vectors in a fully supervised manner. The process for this training operation can be viewed in Figure 14.

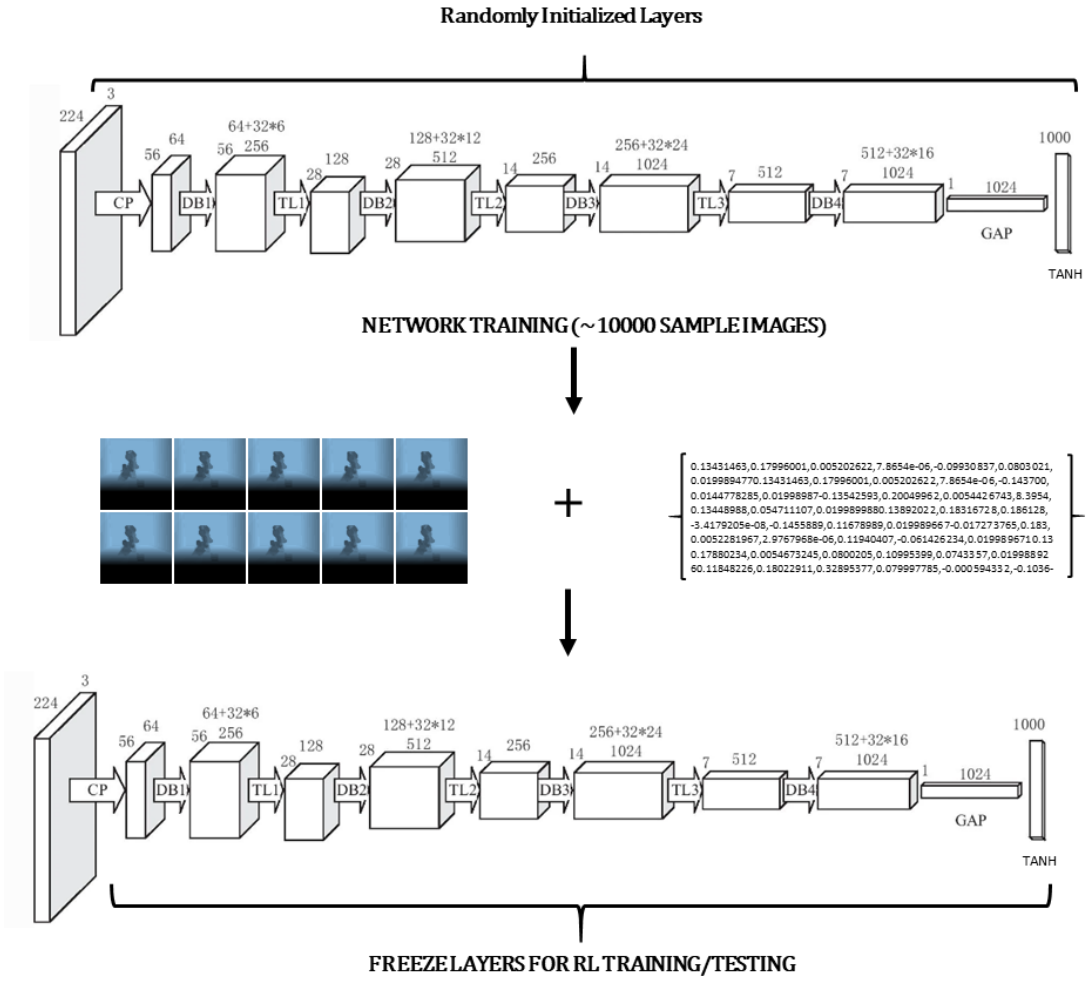



**Figure 14: Pre-Training DenseNet with input BW/D images and target positional vectors**

The pre-trained network can be added to the RL algorithm, so that each time an image is sampled during RL training, the image is fed through the pre-trained SqueezeNet. The network outputs the target position vector (x, y, z), which is then implemented for action selection. The design of the RL learning algorithm which implements this method can be seen in Figure 15.

Two methods for implementing this process can be tested. The pre-trained vector-based policy and the vector position extractor (DenseNet) can be trained separately, or the vector position extractor can be trained first, then the RL policy can be trained directly from the vectors extracted by DenseNet from the sampled images. Due to the time constraints on the project, the first approach has been implemented thus far; however, in the future the second approach could be tested as it would lead to a policy which accounts for stochastic error in the positional inputs.

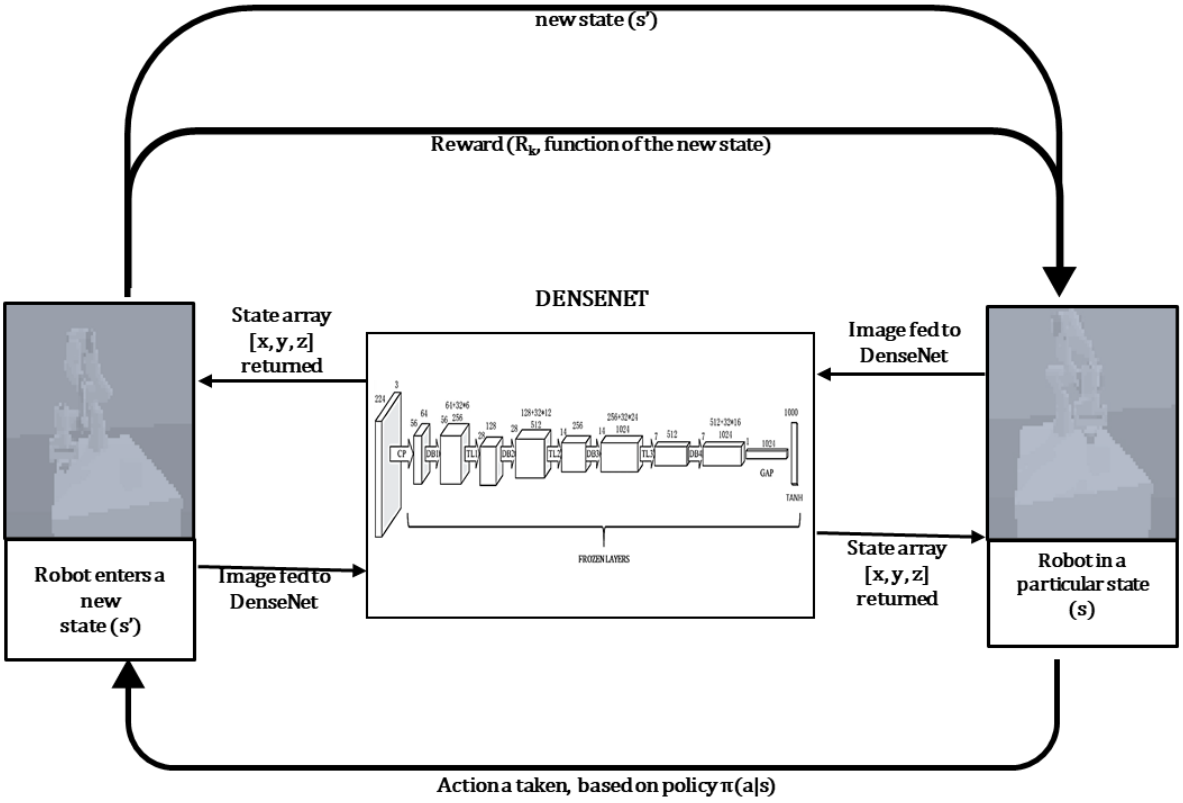



**Figure 15: RL policy exploration with a pre-trained feature extractor**

## 3.8 Training Limitations

The Panda robot simulations were run on the lab workstation with a Nvidia RTX 3070 GPU and an AMD Ryzen 9 3900 CPU. Some limitations for the hyperparameter tuning included RAM capacity (the computer was upgraded from 32Gb-128Gb of RAM), GPU memory (the NVIDIA 3070 has 8 GB of storage) and clock time. Maxing out RAM and GPU capacity had major implications for explorations and training. Checkpoints were implemented to prevent losses in model training during exploration.

# Chapter 4: 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, and a kill switch, which all can be viewed in Figure 16. 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 [92].

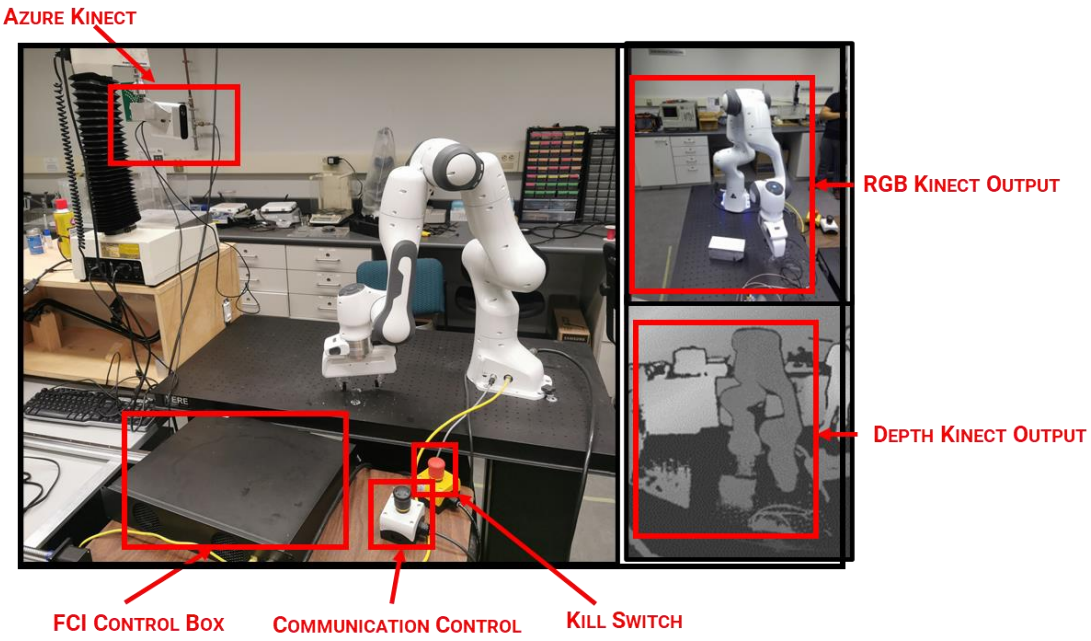



**Figure 16: Panda Robot lab setup**

## 4.1 Panda Robot

The Panda robot consists of the arm: a kinematic chain with 7 articulated joints, and a gripper end effector. Each joint implements high precision encoders and torque sensors which enable the robot to have a pose repeatability of 0.1mm and a force resolution of 0.05N. With a maximum gripping force of 140N, the robot can support a payload of 3kg. The arm has a maximum reach of 0.855m [92].

The Panda was procured alongside the control unit for the device. The control unit is designed as a part of the Franka Control Interface (FCI). The FCI is the interface design for controlling the motion of the robotic arm from a local workstation via an ethernet connection. The FCI interface allows for bi-directional communication between the agent and the workstation to communicate 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 17.

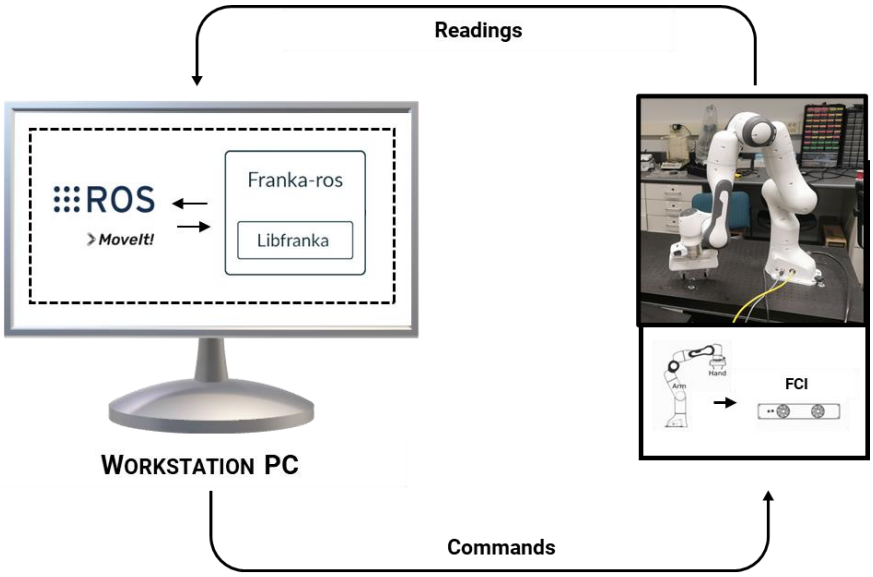



**Figure 17: The FCI Communication framework.**

The FCI allows for 1kH 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).

## 4.2 Control Interfaces

The FCI allows for two forms of control: Franka World and Robot Operating System (ROS) [93]. Franka World is a proprietary pre-built control interface, while ROS is an open-source software library that allows for customizable control communication.

The Franka World UX includes software packages and modules which can be easily applied to the robot. The Franka World interface can be viewed in Figure 18. The Franka World modules can be procured to allow for basic operations with the Franka robot. The high cost and limitations with the Franka World interface make it inapplicable for this project.

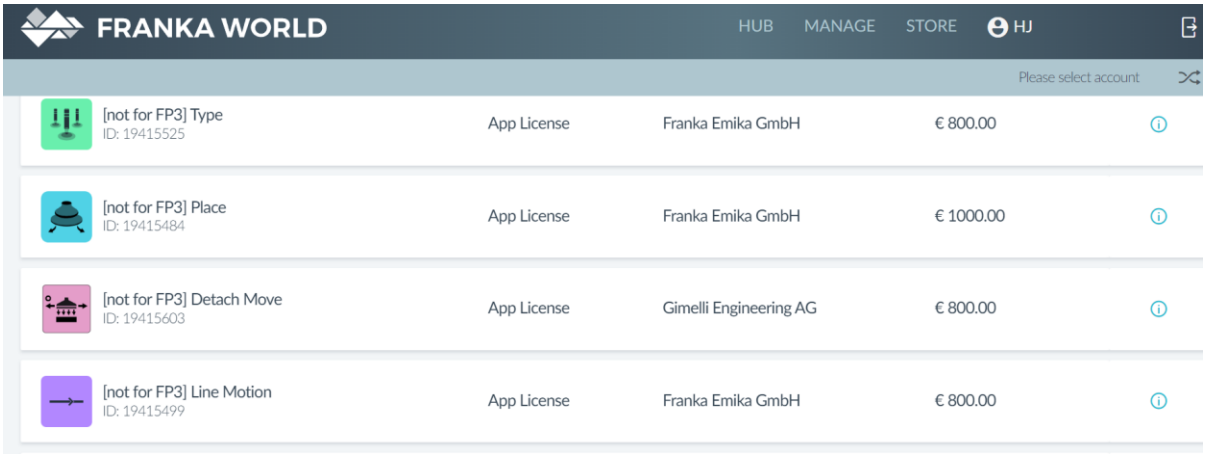



**Figure 18: Franka World UX for Package Selection**

ROS is the open-source alternative to Franka World. ROS is a set of software libraries and pre-built packages which can be applied for integrating advanced control software with real world robotic hardware. Details on the Panda ROS interface (Franka-ROS) and the high-level ROS based motion planner (MoveIt) are discussed in detail in Section 4.3.

## 4.3 Advanced Robotic Control

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

Libfranka can be used for basic non-real-time and real-time functions such as setting controller parameters, processing feedback signals, reading robot states, generating motion paths, and sending torque commands. Libfranka is a useful low-level tool; however, applying libfranka directly requires the user to write custom 1kHz control loops with error catching, joint limit monitoring, and collision behavior setting.

Due to the limited pre-built functionality of libfranka, Franka-ROS was implemented to wrap the environment and integrate advanced ROS features (Figure 17). Details on the implementation of Franka-ROS, and specifics on the integration of MoveIt, the visualization and path planning package, are reviewed in the section below.

### 4.3.1 Franka-ROS

ROS is an open-source customizable OS that allows for the integration of low-level controllers and sensors with higher-level information processors and action planners. ROS is designed based on a graph network wherein individual packages (nodes) communicate information (messages) over channels (topics), to allow for high-level control [94].

The ROS package implemented for this project is Franka-ROS. The Franka-ROS package was pre-built by Franka Emika to allow for the integration of the Panda into the ROS ecosystem. Franka-ROS contains description (franka_description), hardware and control (franka_hw, franka_control), Gazebo (franka_gazebo), and visualization (franka_visualization) packages. The relationships between the Franka-ROS control packages are shown in **Error! Reference source not found.**.

The Franka description contains the Unified Robot Description Format (URDF) files, and the

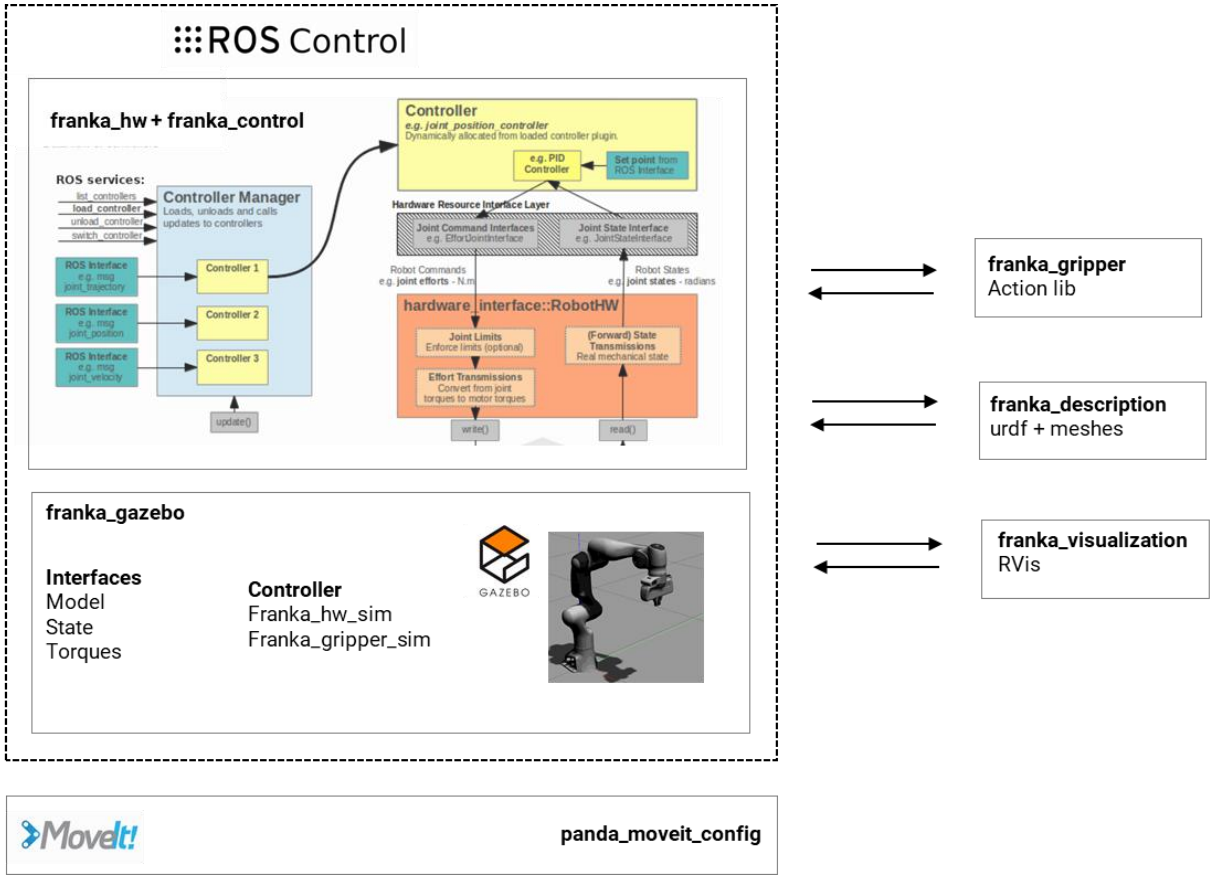



**Figure 19: Franka ROS Control [29]**

visualization meshes required to instantiate and visualize the Panda. The URDF file contains the list of links, joints, connections, etc., required to represent the robotic arm. This list contains all the values required for creating the DH-table for the Panda. The URDF file implemented here is the same as implemented in PyBullet for creating the simulated robot.

The Franka hardware and Franka control packages serve as a bridge between the libfranka API and the ROS system. Specifically, the Franka hardware package contains the abstraction of the hardware for ROS, and the Franka control package creates a hardware node which specifies the control parameters (PID values, constraints, etc) for the Panda.

Franka Gazebo is a package which allows for the instantiation of a Gazebo model for the robot for testing various control schemes and learning approaches. As discussed in 3.2.2, Gazebo was tested, and ruled out as a primary environment for simulating the Panda.

Franka visualization is a useful package which can be implemented to create a UI for interacting with the Panda. The package can be used to render a digital twin of the Panda using RViz. The digital twin RViz positions are set based on current positional readouts from the Panda (simulated Gazebo or real-world) as shown in Figure 20.

The Franka-ROS package allows for the integration of the FCI into the ROS environment. Once inside the ROS environment, other ROS packages such as MoveIt may be applied.

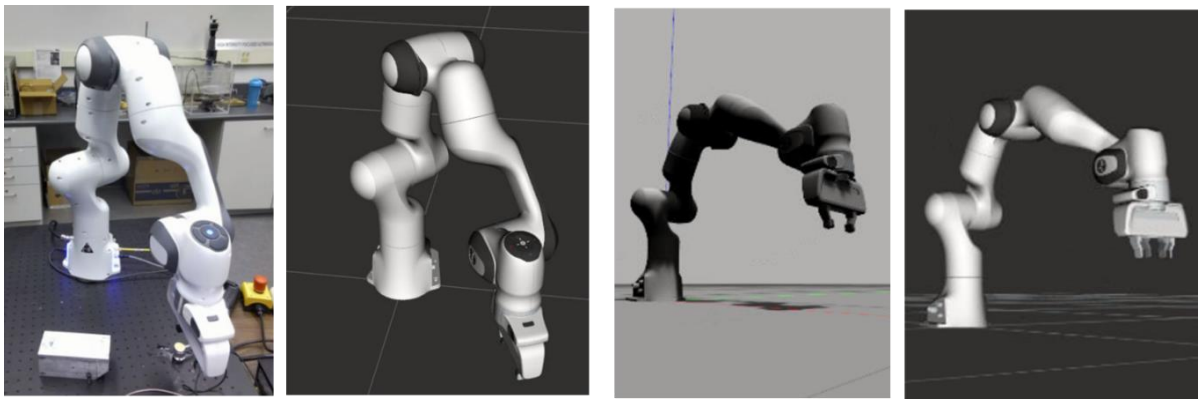


**Figure 20: RVis digital twin of the Real (left) and Simulated (Gazebo) Panda (right)**

### 4.3.2 MoveIt

MoveIt is an open-source library designed for implementing advanced motion planning, kinematics, control, and navigation strategies [95]. MoveIt has some prebuilt functions for the Franka robot, which allows for integration with RViz and the Franka ROS control package.

The Moveit control interface is more advanced than most other planners, as it incorporates a collision detection pipeline prior to allowing action execution. The ROS node graph describing the communication between the MoveIt action server, and the Franka-ROS controller can be viewed in Figure 39 (Appendix D).

## 4.4 Control Implementation

The framework for controlling the Panda robot can be seen in Figure 21. 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 (x,y,z). 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) [96]. 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 inside of the accuracy threshold, the process repeats.

The agent iteratively steps through this control cycle, until the task is completed in the simulated world. The control cycle is simple; however, due to the iterative nature of the cycle, the real-world motion is incremental.

Future implementations may involve completing the entire action sequence in simulation, then feeding the full action sequence into the MoveIt package. For now, the iterative approach is applied since it allows for consistent monitoring during action execution.

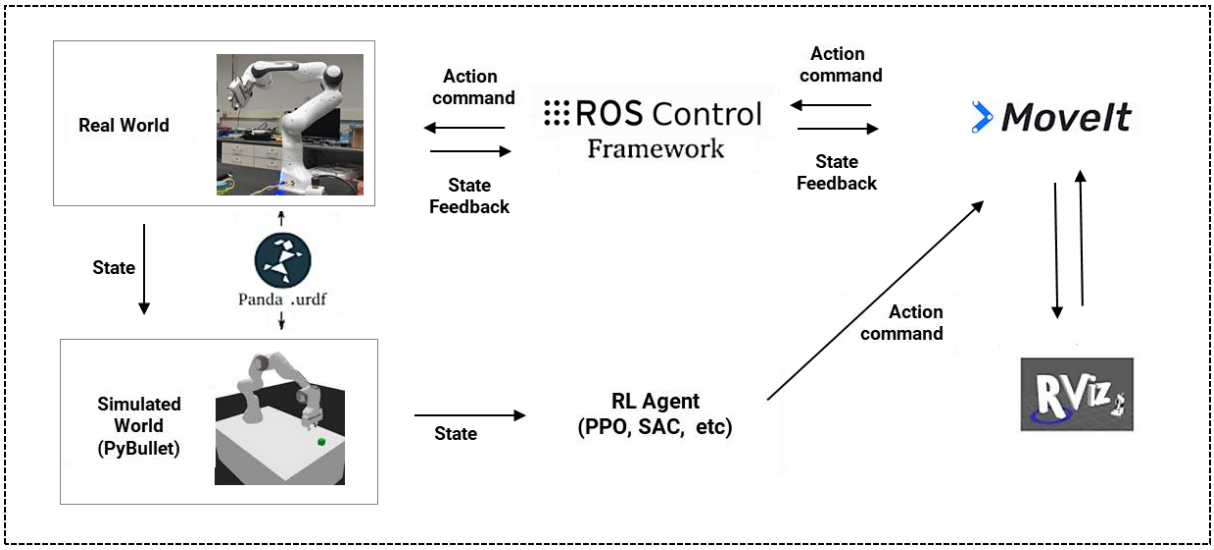



**Figure 21: Control Communication Interface**

# Chapter 5: Results and Discussion

## 5.1 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 and are shown 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 vector-based reach and grasp tasks. For complex tasks such as pick-and-place or for problems with image-based positional feedback, extensive tuning (200+ trials) was required.

A top-down approach was implemented for solving the various RL problems. Vector-based RL training was completed first, which compared PPO and SAC models for reach, grasp, and pick-and-place. For the vector-based problems, sparce and dense reward models were tested. After the relatively simple vector-based problems were completed, PPO and SAC models were tested for image-based grasp and reach problems. The results for the image-based implementation drove the development of a novel Semi-Supervised RL method.

### 5.1.1 Vector-Based RL

The first RL problem explored for this project was vector-based policy training. For this type of problem, image-based feature extractors were not required, and inputs to the network included all necessary positional coordinates.

#### 5.1.1.1 Panda Reach and Panda Grasp with Dense Rewards

This section reviews the results for the Panda reach and grasp tasks with dense reward functions. For the reach problem, the Panda agent had to learn the relationship between the reward, the end effector XYZ coordinates, and the target XYZ coordinates. Similarly, the Panda grasp agent had to learn the relationship between the end effector and the target, while also actuating the gripper. The grasp problem was more complex than the reach problem since the end effector 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. The difference between dense and sparce rewards were discussed in detail in Section 3.4.

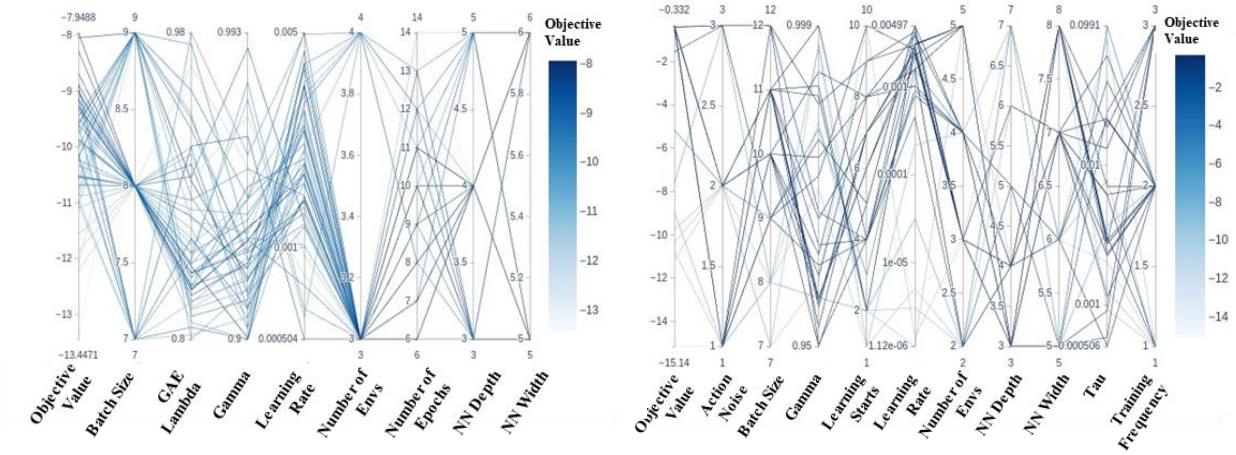



**Figure 22: Left: PCP for PPO Panda-grasp with dense rewards. Right: PCP for SAC Panda-grasp with dense rewards**

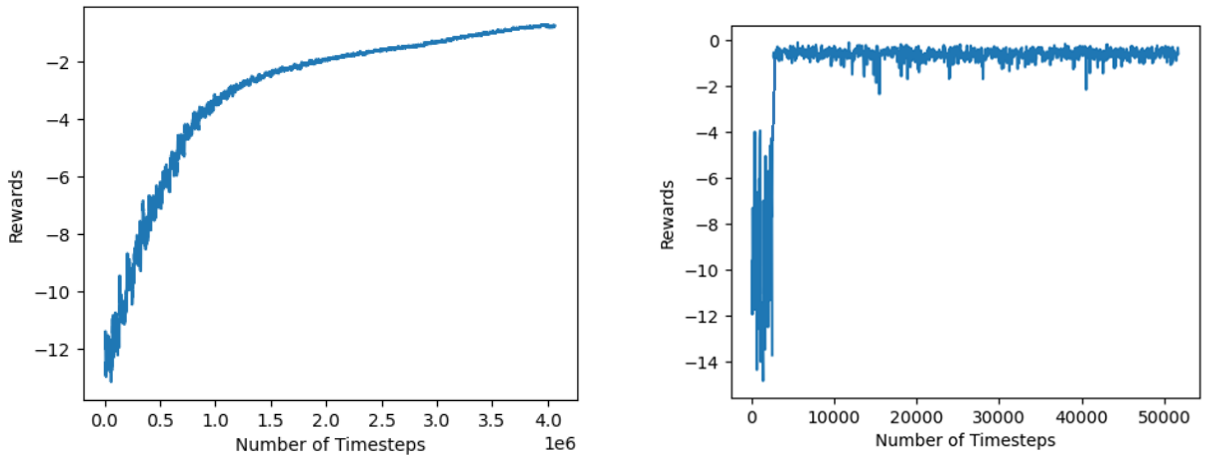



**Figure 23: Left: PPO Panda-reach with dense rewards. Right: SAC Panda-reach with dense rewards**

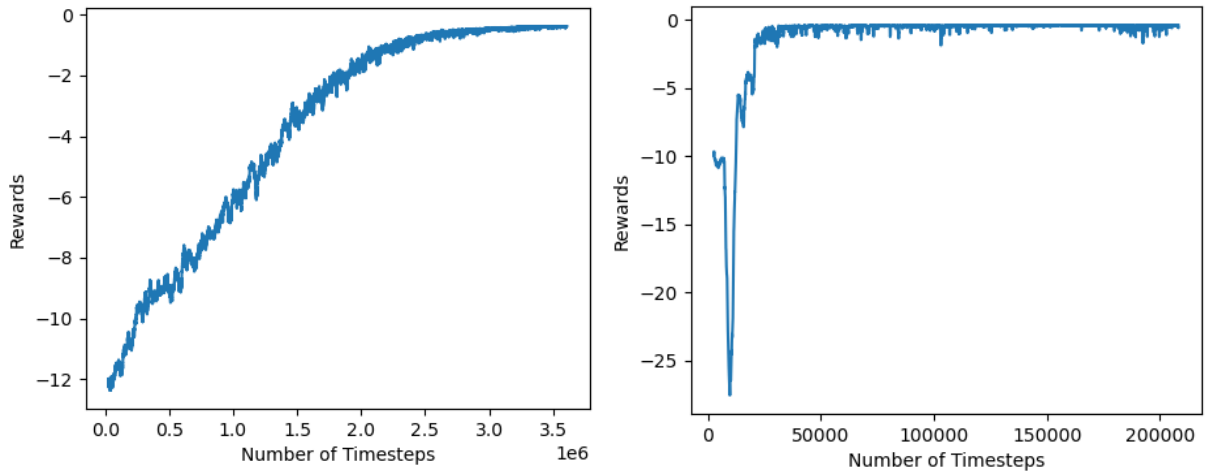



**Figure 24: Left: PPO Panda-grasp with dense rewards. Right: SAC Panda-grasp with dense rewards**

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. As could be expected, the grasp hyperparameters performed well when applied to the reach task so only the PCP for the grasping task is presented in Figure 22.

As shown in Figure 23, Panda Reach was trained efficiently for both PPO and SAC. The PPO model took 3.9-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 task.

As shown in Figure 24**,** Panda grasp was trained efficiently for both PPO and SAC. The PPO model took 2.0-million-time steps to converge to a reach 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.

### 5.1.1.2 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 the section above (5.1.1.1), this problem required the Panda Reach agent to learn the relationship between the reward and the end effector XYZ coordinates, and the target XYZ coordinates. The difference between these studies is in the reward scheme.

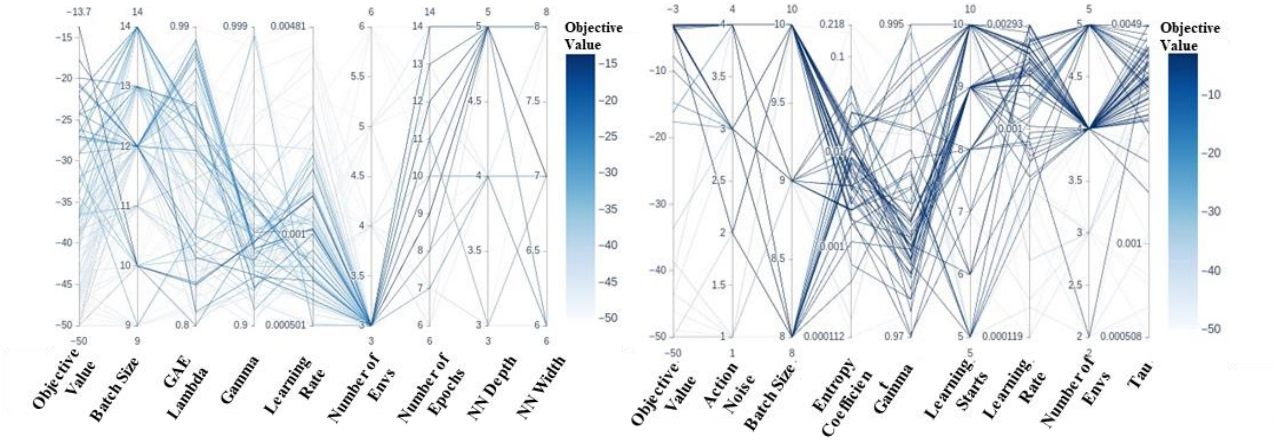



**Figure 25: Left: PCP for PPO Panda-grasp with sparse rewards. Right: PCP for SAC Panda-grasp with sparse rewards**

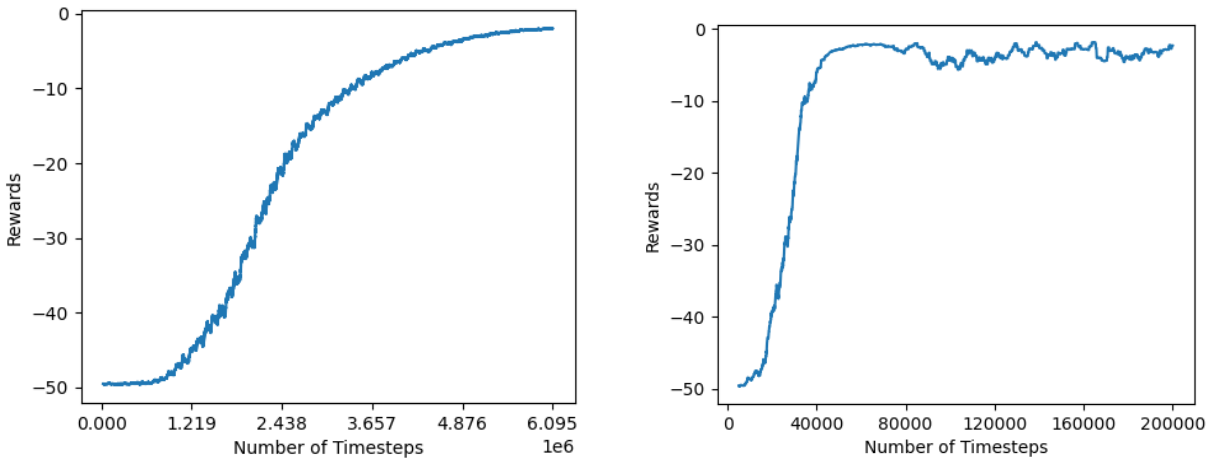



**Figure 26: Left: PPO Panda-reach with sparse rewards. Right: SAC Panda-reach with sparse rewards**

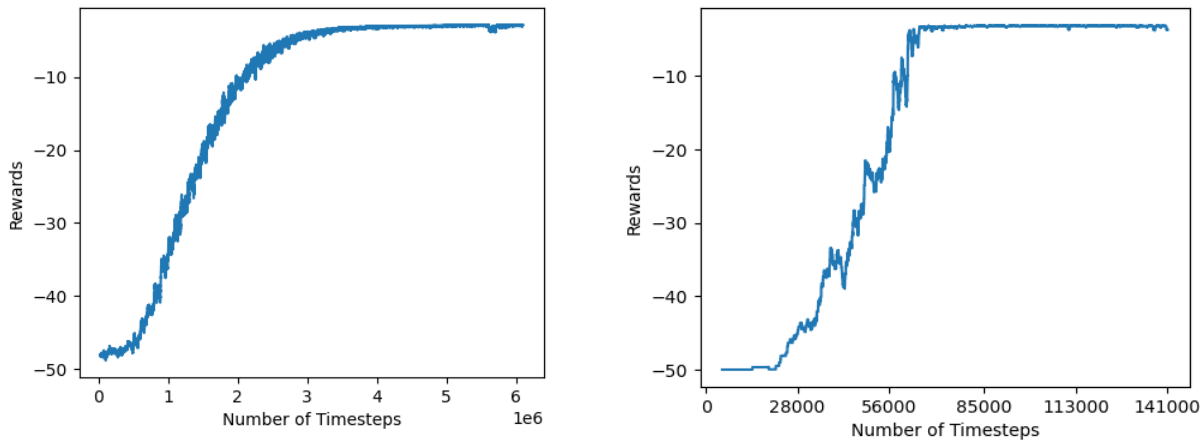



**Figure 27: Left: PPO Panda-grasp with sparse rewards. Right: SAC Panda-grasp with sparse rewards**

For the sparse reward scheme, the agent received a reward of -1 after each step, unless the agent had reached the target position and correctly completed the task. The difference between dense and sparce rewards were discussed in detail in Section 3.4.

Several sets of hyperparameter studies were completed for these tasks (Figure 25). 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.

As shown in Figure 26, Panda Reach was trained efficiently for both PPO and SAC. The PPO model took 5.6-million-time steps to converge to a reach 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.

As can be seen in Figure 27**,** Panda grasp was trained efficiently for both PPO and SAC. The PPO model took 5.5-million-time steps to converge to a reach average accuracy greater than -2.75. The SAC model converged very quickly, breaking the average reward of -3.1 after only 0.14 million steps. With this reward range, the PPO and SAC agents could complete the task with 90% and 95% success rates respectively.

The contrast between sparse and the dense rewards can be understood by comparing this 5.1.1.2 Section, and Section 5.1.1.1. With sparse reward functions, the agent required approximately twice as

many training steps, since additional exploration was required to find the states resulting in more 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, sparce rewards outperformed the dense rewards by ~2%. These results indicate that for problems with minimal difficulty, a simple, sparse, deterministic reward signal is effective. The main disadvantage of the sparse rewards scheme is that significant exploration and clock-time are required for the policy to obtain convergence.

For problems with greater complexity (pick-and-place, grasp with image-based feedback, etc.), the final reward state may rarely or never be reached with pure Monte-Carlo type exploration. This makes reward shaping for these tasks crucial to enable solution convergence. As an alternative to reward shaping, work by Wolski et al.[97], and Fang et al.[98], have shown how hindsight experience replay (HER) can reduce training time and improve convergence for sparse reward schemes. To simplify the problem and maintain focus on learning to solve higher level tasks, only tasks with sparse rewards were explored beyond this point.

#### 5.1.1.3 Panda Pick-and-Place with Dense Rewards

This section reviews the results for the Panda pick-and-place task with sparse 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 has the highest complexity. The agent must learn to find the object, and grasp, lift, and transport it to the target position. A dense reward function is implemented, so there are many sub-optimal solutions for this problem. One sub-optimal solution involves pushing the target object towards the target position while keeping the block on the table. This solution increases the reward, since 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.

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 28. 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. Networks with 5-7-layers were required to solve this problem optimally.

As can be seen in Figure 27, Panda pick-and-place was trained efficiently with both PPO and SAC.

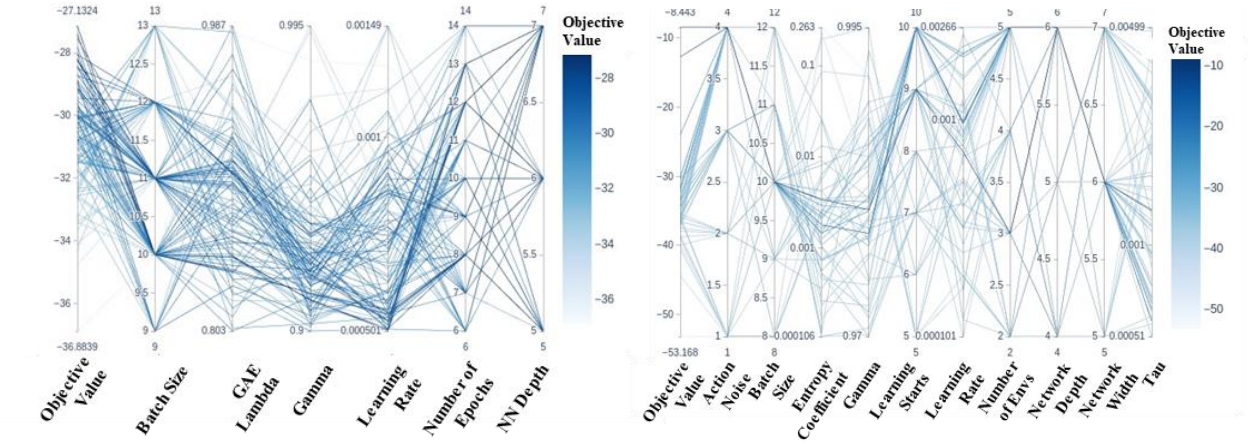



**Figure 28: Left: PCP for PPO Panda pick-and-place with dense rewards. Right: PCP for SAC panda pick-and-place with dense rewards**

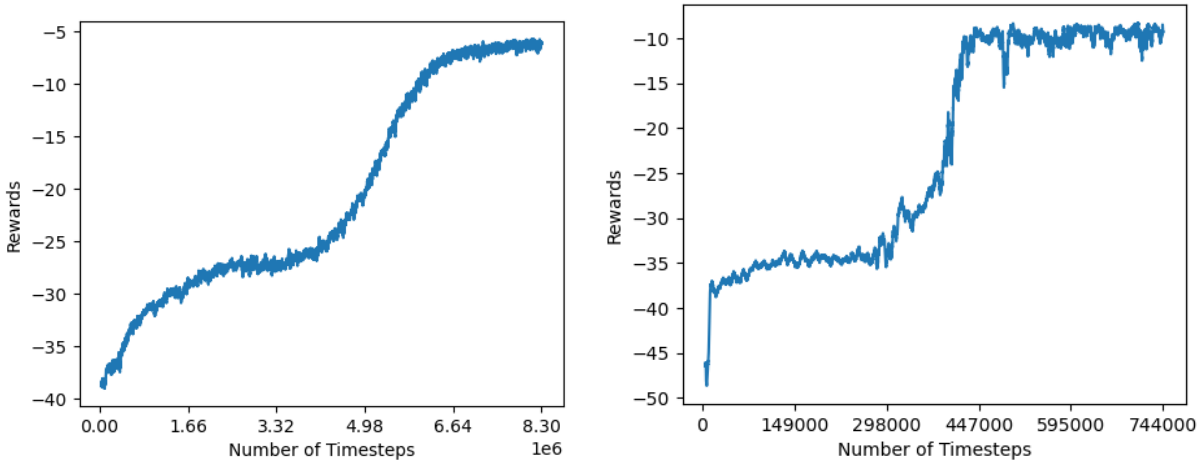



**Figure 29: Left: PPO Panda pick-and-place with dense rewards. Right: SAC Panda pick-and-place with dense rewards**

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 71% and 85% success rates respectively.

Both the SAC and PPO solutions have a rolling reward convergence. The agent first learned the simple suboptimal solution of pushing the target block toward the target position, before learning the truly optimal solution of lifting the block up to the target position.

#### 5.1.1.4 Summary

For the vector-based tasks, 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 ~80% accuracy.

### 5.1.2 Image-Based RL

The second RL problem investigated in this project was image-based policy training. For this problem, a CNN network was trained to extract positional information from a compressed 100x100x2 channel image. The extracted positional information was fed into a Feed Forward Neural Network (FFNN) to find the relationships between positions in rewards, in the same manner as the vector-based problems discussed in Section 5.1.1. Training the sequential FFNN and CNN is a difficult problem, as it requires gradient updates for two networks in series.

Due to the problem complexity, this approach was only implemented for the Panda-reach and Panda-grasp tasks. The pick-and-place task will rely on approaches discussed later (5.1.3). The image-based training was completed with dense rewards.

### 5.1.2.1 Panda-Reach with Dense Rewards

This section reviews the results for the Panda reach task with sparse rewards. Due to the problem complexity, image-based reach required many hyperparameter tuning studies. The PCPs for the final hyperparameter tuning studies run for this problem are shown in Figure 30. For these trials, the learning

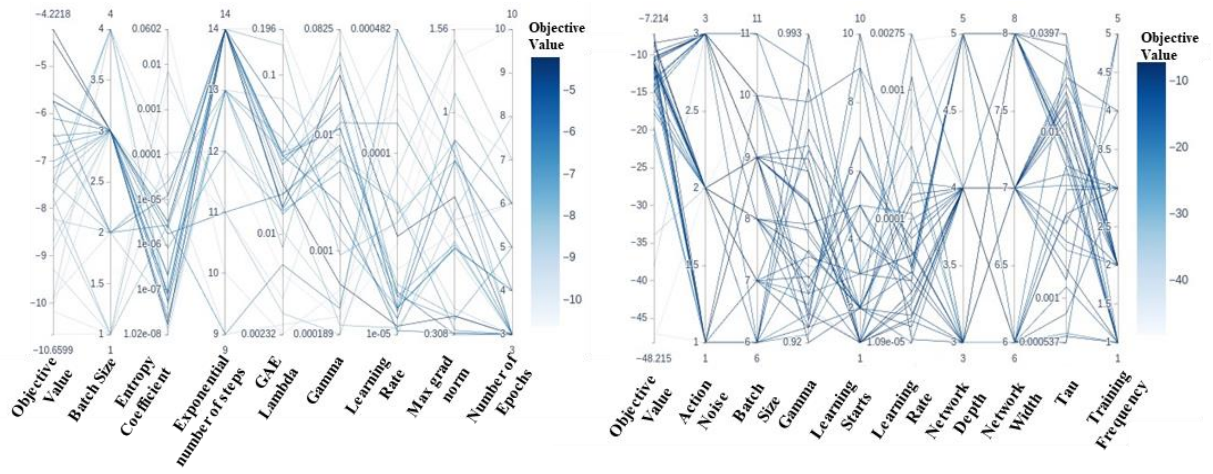



**Figure 30: Left: PCP for PPO Panda-reach with dense rewards. Right: PCP for SAC Panda reach with dense rewards**

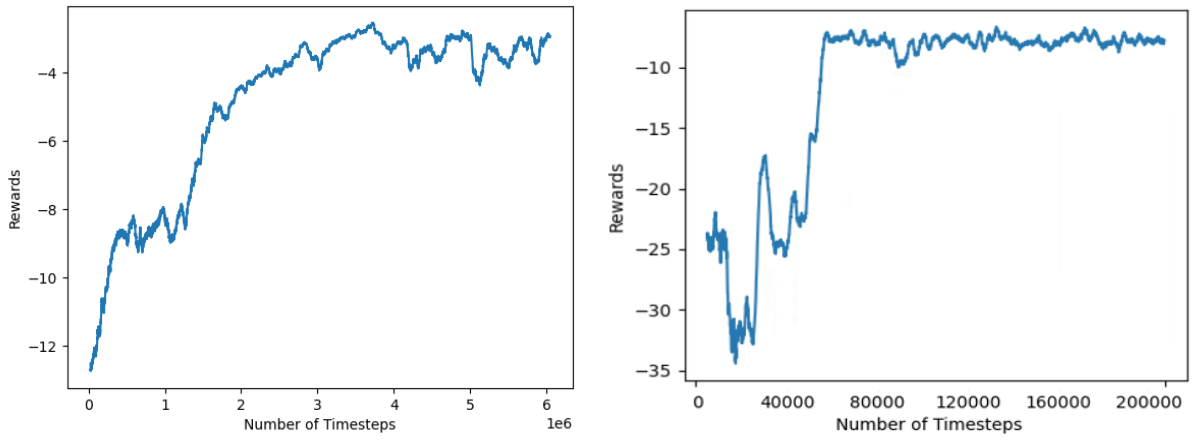



**Figure 31: Left: PPO Panda reach with dense rewards. Right: SAC Panda reach with dense rewards**

rate was low relative to the vector-based problems, since the RL network had many more nodes and required many small updates to learn the pixel-reward relationships. Similarly, the agent required a low

entropy relative to vector-based problems, as the policy had to consistently and slowly learn the optimal policy without drastically modifying weights with each update.

As can be seen in Figure 31, the agent had a difficult time solving the image-based reach task. The PPO model took 3.1-million-time steps to converge to a reach reward greater than -2.9 and the SAC model took 0.08-million-time steps to converge to a reach reward greater than -7.3. With these reward ranges, the PPO and SAC agents could complete the tasks with 32% and 13% success rates respectively. Compared to the vector-based problems, the image-based reach training is unstable, and the final training rewards are poor.

#### 5.1.2.2 Panda-Grasp with Dense Rewards

This section reviews the results for the Panda-grasp task with sparse rewards. The image-based grasp problem is the most difficult of the tasks listed. Due to the task complexity, image-based grasp required many hyperparameter tuning studies. The PCP for the final hyperparameter tuning study run for this

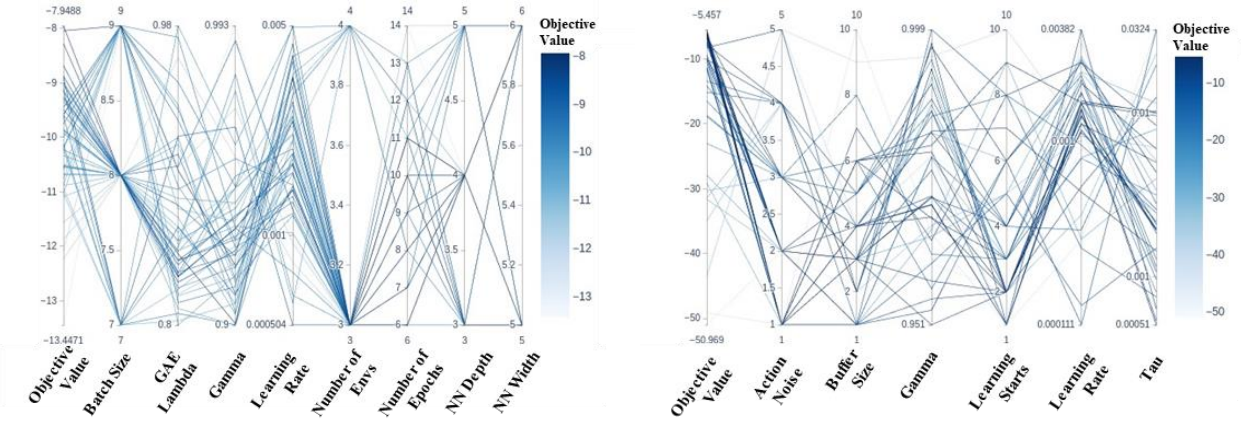



**Figure 32: Left: PCP for PPO Panda-grasp with dense rewards. Right: PCP for SAC Panda-grasp with dense rewards**

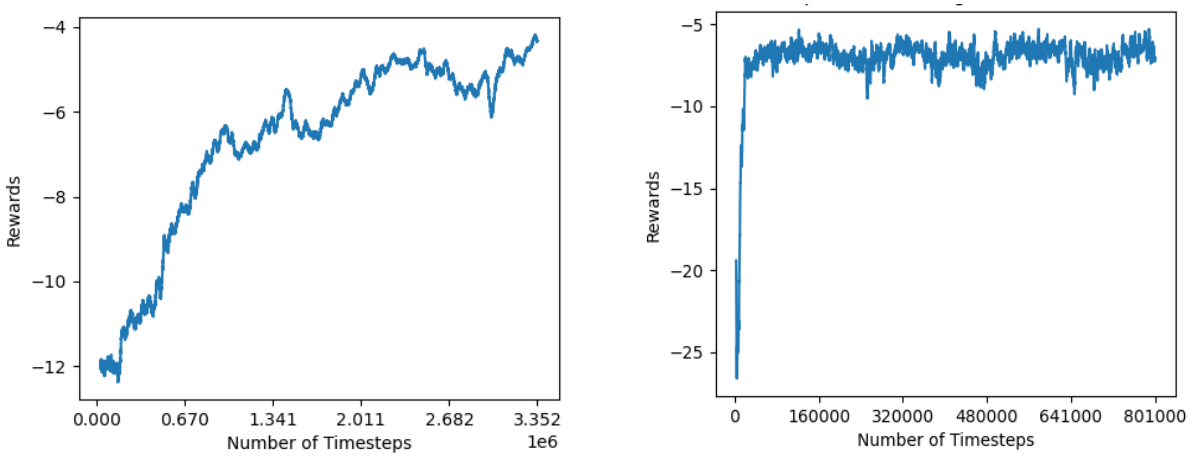



**Figure 33: Left: PPO Pand-grasp with dense rewards. Right: SAC Panda-grasp with dense rewards**

problem is shown in Figure 32. The complexity of the grasp task was higher than reach, so the model performed best with a higher relative learning rate, entropy, and noise.

As shown in Figure 33, the image-based grasping task was solved sub-optimally compared to the vector-based grasping. The PPO model took 3.4-million-time steps to converge to a reach reward greater than -3.0. The SAC model took 0.4-million-time steps to converge to a reach reward greater than -7.0. With this reward range, the PPO and SAC agents completed the task with 8% and 6% success rates respectively. Compared to the vector-based problems, the image-based grasp training is unstable, and the final training rewards are poor.

#### 5.1.2.3 Summary

For the image-based RL task, even with extensive hyperparameter exploration, the trial success rate for SAC and PPO was low. The trained policies perform much better than random, however the optimal solutions as seen with the vector-based problems could not be found. This result drove the need for modifications to the image-based training process.

### 5.1.3 Image-Based Semi Supervised RL

The results from Section 5.1.2 indicated that solving high level grasp and placement tasks for PPO and SAC through network and hyperparameter tuning alone was not feasible. As an alternative, the semi-supervised RL approach was developed and tested.

The semi-supervised approach leveraged known and well-documented networks [91], [99], [100] which were pre-trained to serve as accurate position vector extractors. The pre-trained networks are applied in coordination with the vector-based RL networks to take image inputs and return high accuracy action commands.

Each of the networks were originally designed for classification, so minor changes were required for application to the localization problem. Several limitations were applied during network selection. The networks had to be able to train effectively with <10,000 images to maintain sample efficiency, and the networks had to be able to train with a limited GPU memory (8 Gb).

The network pre-training involved sampling ~10,000 images and gripper positions from the simulated environment and training the network to identify target object positions. The pipeline for this training process can be viewed in Figure 14.

#### 5.1.3.1 Network Comparisons

As shown in Table 4, five networks were investigated in solving this problem. The older methods InceptionNet [101] and AlexNet [102] were ruled out due to the size of the network. For these network sizes, the batches were too small to have smooth convergence, and memory errors were common.

**Table 4: Comparison of CNN networks**

| *Network name* | *Year Released* | *Number of Trainable Parameters* | *Trainable with NVIDIA 3070* | *Best Performance* |
|---|---|---|---|---|
| ***SqueezeNet*** | 2016 | 724035 | Yes | 1 |
| ***DenseNet*** | 2016 | 6956931 | Yes | 3 |
| ***ResNet*** | 2015 | 11178051 | Yes | 2 |
| ***InceptionNet*** | 2014 | 27161264 | No | N/A |
| ***AlexNet*** | 2012 | 61100840 | No | N/A |

The 3 networks which were simulated and tested for this problem were SqueezeNet, DenseNet and ResNet. Convergence (5%) of each of the models occurred at 350 epochs. The results achieved from training these models can be seen in Figure 34. Relative to the other networks, DenseNet had the smallest number of trainable parameters, and the best convergence.

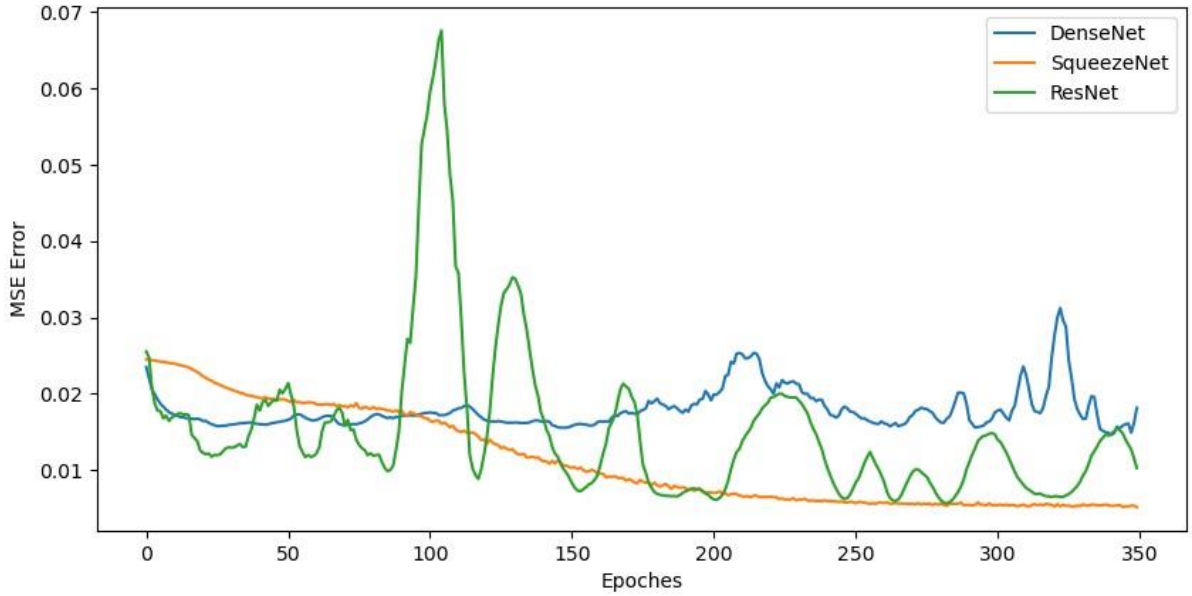



**Figure 34: Supervised learning position-based training for various CNNs**

### 5.1.3.2 Network Tuning

For the models shown, minor tuning for batch size (32), learning rate (0.001), and number of agents (8) was completed. Learning rate exploration involved testing constant learning rates (0.1 and 0.001), as well as the Cyclical Learning Rate (CLR) approach [103]. The CLR approach converged quickly to a less accurate solution, so a fixed learning rate was selected.

In addition to batch size, learning rates and number of agents, a factor which was considered for reducing the MSE for the model was image size. Intuitively, 80x80 images would have a lower accuracy than 120x120 images due to compression effects. To find the optimal image size, a study was completed comparing image size to accuracy of the output positional vector. The results can be seen in Figure 35. The study revealed that networks trained with 100x100 images resulted in the highest output accuracy. Images with 90x90 and 120x120 pixels had a very similar output accuracy, which indicates that beyond the threshold of 90 pixels, most positional information compressed into the image is preserved.

The PPO and SAC networks were tested with this semi-supervised RL approach for Grasp and Pick-and-place. The implementation involved using the pre-trained vector-based networks, in combination with the pre-trained SqueezeNet position vector extractor. The SqueezeNet position extractor had a maximum deviation from the real position of 0.5 inches.

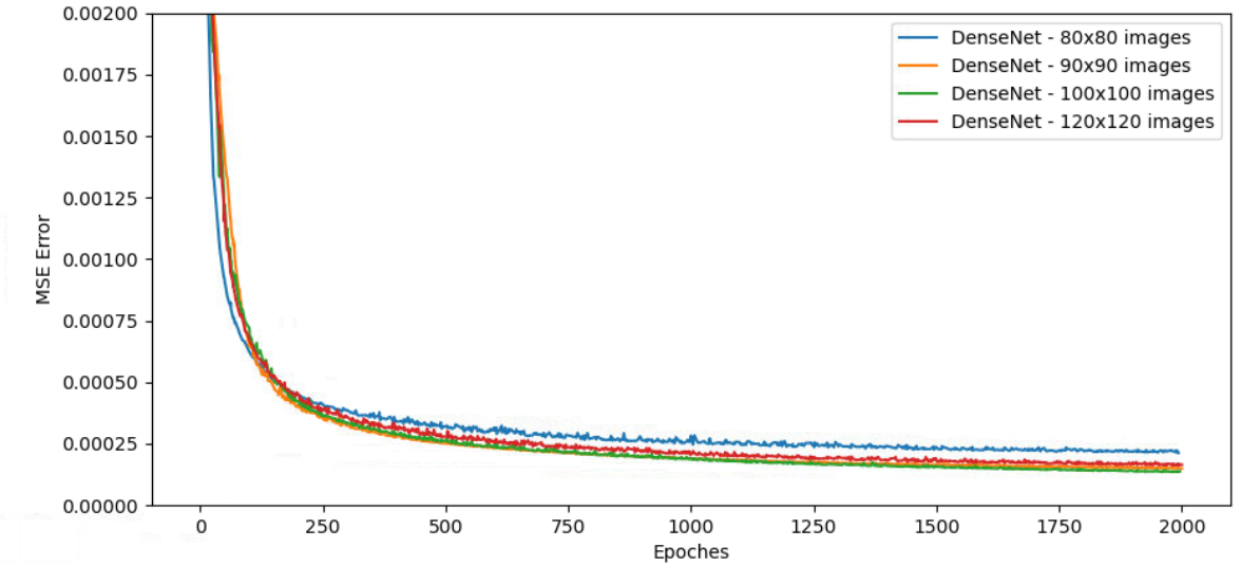



**Figure 35: Image size vs localization accuracy**

The grasp task with PPO and SAC networks implementing the SqueezeNet positional vector extractor was completed with 82% and 85% success rates respectively. This approach therefore provides a massive uptick in model performance, bringing the completion rate for the tasks with image-based feedbacks to accuracy levels comparable to the models implementing vector-based feedback.

The pick-and-place task with PPO and SAC networks implementing the SqueezeNet positional vector extractor was completed with 8% and 14% success rates respectively. Due to the difficulty of this task, it is not surprising that the task success rate is low; however, this shows promise for the performance of this model with further tuning.

### 5.1.4 Summary

Table 5 shows a summary of all the simulation results for tasks with image and vector-based feedback. As shown, the RL agent was able to learn to complete all vector-based tasks consistently. Alternatively, with image-based positional inputs, the RL agent performed poorly. To improve simulation results, a

pre-trained DenseNet feature extractor was developed. The combination of vector-based RL with a supervised feature extractor enabled RL to be applied successfully to image-based tasks.

The results for pick-and-place for both vector and image-based tasks were noticeably lower than some of the results from the literature shown in Section 2.9. When comparing the literature and the completed studies, the primary rational 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 move. Most of the failures noticed in simulation were due to interference between the gripper and this block. Although this change makes the task significantly more difficult, the results are more realistic for application in the real world.

The comparison of PPO and SAC reveals several patterns in training time, performance, and convergence. For all problems explored, SAC required a minimum of one order of magnitude less steps than PPO to obtain convergence. Additionally, for problems with low difficulty, such as vector-based panda reach, grasp, and pick-and-place, SAC converged to a more optimal solution than PPO. These SAC advantages contrast starkly with the convergence difficulties experienced by SAC. For problems with high difficulty, SAC was significantly outperformed by PPO, due to problems with divergence, or convergence to local optima. SAC was highly hyperparameter sensitive compared to PPO, which resulted in additional time being required to determine the ideal hyperparameters for each task.

The results seen 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 while SAC tended to diverge when overtrained.

**Table 5: 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 | 71 |
| ***Panda Pick and Place*** | Dense | Vector | SAC | 85 |
| ***Panda Reach*** | Dense | Image | PPO | 32 |
| ***Panda Reach*** | Dense | Image | SAC | 13 |
| ***Panda Grasp*** | Dense | Image | PPO | 11 |
| ***Panda Grasp*** | Dense | Image | SAC | 6 |
| ***Panda Grasp*** | Dense | Image | SL + PPO | 82 |
| ***Panda Grasp*** | Dense | Image | SL + SAC | 85 |
| ***Panda Pick and Place*** | Dense | Image | SL + PPO | 8 |
| ***Panda Pick and Place*** | Dense | Image | SL + SAC | 14 |

## 5.2 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 is 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 future real-world RL implementation. The communication between PyBullet, ROS, MoveIt and the FCI was established, as shown in Figure 21.

During testing, the Panda agent is instantiated in PyBullet, and each incremental step the agent takes in the simulation space is replicated in the real world. After each action step is executed in PyBullet, the agent waits to take a new step until the real-world arm has moved to match its digital twin. During each step, the Frank-ROS feedback loop ensures the positional accuracy of the end effector. The incremental stepping approach is 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 6 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 which 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.

The target object implemented in the lab was a box which was slightly taller and narrower than the simulated target. The difference in geometry resulted in the real-world target object being bumped during several real-world tests. In addition to geometry differences, some minor variances were noticed between the physics of the environment and the real-world. Several times in simulation, the target object was bumped. In simulation, these collisions caused the target block to move several inches, while in the real world these collisions resulted in the target block moving 0.25-0.5 inches. Friction coefficients in simulation have been modified, but still the kinematics between the real and simulated worlds do not match. It is probable that this challenge could be caused by inaccuracies of the physics in the Bullet physics engine.

**Table 6: 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 | 60 |
| ***Panda Pick-and-Place*** | Dense | Vector | SAC | 70 |

To improve real world testing, several tactics should be implemented. First, the simulated or real-world target could be modified so both target objects match. This change is relatively small and 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 implemented to ensure that the real and simulated target positions match during each trained task.

# Chapter 6: Conclusion and Future Work

## 6.1 Conclusion

Considerable progress has been made in this project towards the goal of creating a simulated and real-world autonomous robotic agent capable of performing basic tasks such as reach, grasp, and pick-and-place.

As reviewed in Chapter 3, custom simulation environments were created, a tuning pipeline was established, RL network training methods were implemented, and a custom training technique was developed. As discussed in Chapter 4, the real-world robotic arm was installed, communication between the workstation, control box, and sensors, was established, and the advanced Libfranka, Franka-ROS, and MoveIt control system was installed and tested. Additionally, a ROS connection between the RL simulated PyBullet agent with the real-world Panda was established.

The results from the development of this project, the tuning of the hyperparameters, the training of PPO and SAC models, and the implementation of the Semi-Supervised RL framework were reviewed in Chapter 5. These results indicate that for reach, grasp, and pick-and-place problems with vector state representations, both SAC and PPO frameworks perform well at training an autonomous simulated agent.

For the same simulated tasks with image state feedback, both models perform poorly and are not able to learn optimal policies, indicating that alternative methods were required. To improve model performance with an image-based input, the new semi-supervised RL approach was developed and tested. The semi-supervised approach enabled the RL agent to complete image-based grasp and reach tasks with high consistency.

Due to time constraints, real-world testing on this project was limited to tasks with vector-based state feedback. The real-world testing validated the communication framework between the simulated and real environments and indicated that real-world policy transference is possible. Accuracy of coordinate-based reach, grasp, and pick-and-place was reduced by 10-20% compared to the simulation environment, which indicates that further calibration of the simulation environment and modifications to the target object are required.

## 6.2 Future Work

This project has served as a proof of concept for real-world robotic RL; however, there are many gaps which need to be filled for the RL to be applied at a high level to the real world. Major topics which require future work include: 1) domain adaptation and randomization for sim-to-real transfer, 2) imitation learning for reward shaping and solution sampling, and 3) image-based transfer learning.

### 6.2.1 Sim-to-Real Transfer

The benefits of training offline are often contrasted with the loss of accuracy when the trained model is applied in the real world. Inaccuracies of model parameters such as friction, weight, or dimension have been seen to have significant influence during model testing. The difference between the offline and online models is referred to as the "reality gap". The primary approaches for coping with the reality gap are domain randomization and domain adaptation [15].

Domain randomization is a method of training the agent in an inconsistent environment to force the agent to learn to ignore "noise" and changes in the background. Noise added for domain randomization could include lighting, color, positions, textures, or other features that serve to confuse the scene [104]. Domain adaptation is an approach that implements a form of generative adversarial networks (GANs) to generalize the test data onto the target domain. The process implements data from the test domain (online) to train the model in the training domain (offline) so that training in the simulation generalizes well to online testing [15], [105].

As shown in Section 0, sim-to-real transfer is a problem with this implementation of RL and will need to be addressed for the vector and image feedback problems. Due to the major difference in the rendering characteristics between the real and simulated (PyBullet) environments, it is expected that for the image-based tasks there may need to be a change to MuJoCo.

An alternative method to improve the sim-to-real transfer for the image-based tasks, would be to implement point cloud mapping for this environment rather than simple RGBD image feedback. Point cloud mapping could be implemented to improve positional accuracy of the target block and reduce noise in the feedback signal.

### 6.2.2 Imitation Learning

Reward shaping is one of the tasks which require human intervention in RL. With only a monolithic goal, the agent may be forced to search the environment for an extended period to refine its actions. An alternative path to manual reward shaping is imitation learning [17], [106], [107].

Imitation learning can be implemented to learn a reward function from examples. The reward function can be used to refine the policy depending on the scenario, allowing for domain adaptation [107]. Imitation (apprenticeship) learning includes several different approaches, such as Behavior Cloning (BC) and Inverse Reinforcement Learning (IRL) [107].

Many papers have been published on the topic of IRL [106]–[108] and its successes. To improve the performance of the image-based RL for robotics, imitation learning may be a useful tool for implementation in the future.

### 6.2.3 Network Generalization

Problem generalization is an open topic, which relates to the issue of applying learning from the training set to a broader spectrum of problems. [23] gave an excellent example of a technique that can scale a crude model to a large test set; however, this approach still requires online testing. Further exploration is required to determine the feasibility of developing generalized CNNs which can be applied efficiently for positional-vector extraction on any image.

One potential opportunity for the semi-supervised RL approach is to pre-train the DenseNet CNN on a standard image-based data set, and subsequently to freeze the weights in all but the final layers of the network. Pre-training this network would mean that only minor modifications would be required to the final layer, which would allow for reduced training time. Pre-trained networks have been shown to improve results for classification and regression tasks [109], so further research is required to determine if this could be leveraged for RL.

# References


[1] RobotShot Distrobution Inc., "History of Robotics: Timeline," *Robotshop*, 2008. https://www.robotshop.com/media/files/PDF/timeline.pdf

[2] A. Gasparetto and L. Scalera, "A Brief History of Industrial Robotics in the 20th Century," *Advances in Historical Studies*, vol. 8, pp. 24–35, Feb. 2019.

[3] D. Massa, M. Callegari, and C. Cristalli, "Manual guidance for industrial robot programming," *Emerald Group Publishing Limited*, pp. 457–465, 2015, doi: 10.1108/IR-11-2014-0413.

[4] G. Biggs and B. Macdonald, "A Survey of Robot Programming Systems," Brisbane, Australia, Dec. 2003, p. 27.

[5] S. K. Saha, *Introduction to Robotics*, Second Edition. New Delhi, India: McGraw Hill Education, 2014.

[6] J. Craig, *Introduction to Robotics Mechanics and Control*. Upper Saddle River, NJ, USA: Pearson Education International, 2005.

[7] M. T. Ballestar, Á. Díaz-Chao, J. Sainz, and Joan Torrent-Sellens, "Impact of robotics on manufacturing: A longitudinal machine learning perspective," *Technological Forecasting & Social Change*, vol. 162, p. 120348, Oct. 2020.

[8] H. F. Al-Selwi, A. Abd. Aziz, and F. S. Abas, "Reinforcement Learning for Robotic Applications with Vision Feedback," presented at the 2021 IEEE 17th International Colloquium on Signal Processing & Its Applications (CSPA), Langkawi, Malaysia, Mar. 2021.

[9] L. Tai, J. Zhang, M. Liu, J. Boedecker, and W. Burgard, "A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation," Dec. 2016.

[10] J. Kober, A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013, doi: 10.1177/0278364913495721.

[11] D. Liu, Z. Wang, B. Lu, M. Cong, H. Yu, and Q. Zou, "A Reinforcement Learning-Based Framework for Robot Manipulation Skill Acquisition," *IEEE Access*, vol. 8, pp. 108429–108437, Jun. 2020, doi: 10.1109/ACCESS.2020.3001130.

[12] D. Kalashnikov *et al.*, "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," presented at the 2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland, Oct. 2018.

[13] M. Q. Mohammed, K. L. Chung, and C. S. Chyi, "Pick and Place Objects in a Cluttered Scene Using Deep Reinforcement Learning," *International Journal of Mechanical & Mechatronics Engineering*, vol. 20, no. 04, pp. 50–57, 2020.

[14] Rongrong Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Drespp-Langley, "Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review," *MDPI: Robotics*, Jan. 2021.

[15] K. Kleeberger, R. Bormann, W. Kraus, and M. Huber, "A Survey on Learning-Based Robotic Grasping," *Current Robotics Reports*, pp. 239–249, 2020, doi: 10.1007/s43154-020-00021-6.

[16] Y. Xiao, S. Katt, A. ten Pas, S. Chen, and C. Amato, "Online Planning for Target Object Search in Clutter under Partial Observability," presented at the 2019 International Conference on Robotics and Automation (ICRA), Montreal, Canada, May 2019.

[17] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts, London, England: The MIT Press, 2018.
[18] S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*, Fourth Edition. Hoboken, NJ, USA: Pearson Education, Inc.
[19] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, Nov. 2017.
[20] A. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations theory and application to reward shaping," in *In Proceedings of the Sixteenth International Conference on Machine Learning*, San Francisco, California, United States, Jun. 1999, pp. 278–287.
[21] M. Gualtieri, A. Pas, and R. Platt, "Pick and Place Without Geometric Object Models," Brisbane, QLD, Australia, 2018, pp. 7433–7440. doi: 10.1109/ICRA.2018.8460553.
[22] Marcus Gualtieri and Robert Platt, "Learning 6-DoF Grasping and Pick-Place Using Attention Focus," *arXiv:1806.06134*, Sep. 2018.
[23] C. Beltrain-Hernandez, P. Damien, K. Harada, and I. Ramirez-Alpizar, "Learning to Grasp with Primitive Shaped Object Policies," *SII*, pp. 468–473, 2019, doi: 10.1109/SII.2019.8700399.
[24] L. Berscheid, P. Meißner, and T. Kröger, "Robot Learning of Shifting Objects for Grasping in Cluttered Environments," presented at the IROS, Macau, China, Nov. 2019.
[25] C. Finn, S. Levine, and P. Abbeel, "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization," in *Proceedings of the 33 rd International Conference on Machine Learning*, New York, NY, USA, Jun. 2016, vol. 48.
[26] M. Pedersen *et al.*, "Robot skills for manufacturing: From concept to industrial deployment," *Robotics and Computer-Integrated Manufacturing*, vol. 37.
[27] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer, 2008.
[28] R. Haschke, "ROS Planning For Franka Emika," *Moveit*, Jun. 30, 2021. https://ros-planning.github.io/moveit_tutorials/doc/getting_started/getting_started.html (accessed Dec. 12, 2021).
[29] Franka Emika, "Franka Control Interface." https://frankaemika.github.io/docs/overview.html (accessed Dec. 12, 2021).
[30] S. Lavalle, "THe Homogeneous Transformation Matrix," *Cambridge University Press*, 2006.
[31] Peter Corke, "Robotics Toolbox." https://petercorke.com/toolboxes/robotics-toolbox/
[32] C. Hughes and T. Hughes, *Robotoic Programming: A Guide to Controlling Autonomous Robots*. Indianapolis, Indiana, USA: Que, 2016.
[33] G. Ajaykumar, M. Steele, and C.-M. Huang, "A Survey on End-User Robot Programming," *arXiv*, vol. arXiv:2105.01757, May 2021, doi: 10.1145/3466819.
[34] J. Fu, S. Levine, and P. Abbeel, "One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors," 2016, pp. 4019–4026. doi: 10.1109/IROS.2016.7759592.
[35] F. Lewis, D. Dawson, and C. Abdallah, *Robotic Manipulator Control Theory and Practice*, Second Edition, Revised and Expanded. New York, NY, U.S.A: Marcel Kekker, Inc, 2005.
[36] B. Stapelberg and K. M. Malan, "A survey of benchmarking frameworks for reinforcement learning," *South African Computer Journal*, vol. 32, no. 2, Nov. 2020, doi: 10.18489/sacj.v32i2.746.
[37] M. Gualtieri and R. Platt, "Learning 6-DoF Grasping and Pick-Place Using Attention Focus," presented at the 2nd Conference on Robot Learning, Zürich, Switzerland, Oct. 2018. [Online]. Available: https://arxiv.org/abs/1806.06134

[38] C. Atkeson and J. Santamaria, "A Comparison of Direct and Model-Based Reinforcement Learning," presented at the IEEE International Conference on Roboticsand Automation, Albuquerque, New Mexico, Apr. 1997.
[39] O. Sigaud and O. Buffet, *Markov Decision Processes in Artificial Intelligence*, 2nd ed. John Wiley & Sons, Inc, 2010.
[40] M. P. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–114, 2013, doi: 10.1561/2300000021.
[41] S. Levine and V. Koltun, "Guided Policy Search," in *Proceedings of Machine Learning Research*, Atlanta, Georgia, USA, Jun. 2013, vol. 28, pp. 1–9. [Online]. Available: http://proceedings.mlr.press/v28/levine13.html
[42] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31 st International Conference on Machine Learning*, Beijing, China, Jun. 2016, vol. 32.
[43] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of The 33rd International Conference on Machine Learning*, New York, New York, USA, Jun. 2016, vol. 48, pp. 1928–1937.
[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Stockholm, Sweden, Jul. 2018.
[45] J. Schulman, F. Wolski, Prafulla Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv:1707.06347*, Jul. 2017.
[46] S. Karagiannakos, "Trust region and Proximal Policy Optimization (TRPO and PPO)," *AI Summer*, Jan. 11, 2019. https://theaisummer.com/TRPO_PPO/ (accessed Dec. 13, 2021).
[47] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *arXiv*, Feb. 2015.
[48] A. D. Laud, *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.
[49] R. Nagpal, A. U. Krishnan, and H. Yu, "Reward engineering for object pick and place training," *arXiv preprint arXiv:2001.03792*, 2020.
[50] M. Grzes and D. Kudenko, "Learning shaping rewards in model-based reinforcement learning," in *Proc. AAMAS 2009 Workshop on Adaptive Learning Agents*, 2009, vol. 115, p. 30.
[51] M. J. Mataric, "Reward functions for accelerated learning," in *Machine learning proceedings 1994*, Elsevier, 1994, pp. 181–189.
[52] Y. Luo, K. Dong, L. Zhao, Z. Sun, C. Zhou, and B. Song, "Balance Between Efficient and Effective Learning: Dense2Sparse Reward Shaping for Robot Manipulation with Environment Uncertainty," *arXiv:2003.02740v1*, Mar. 2020.
[53] S. Jang and M. Han, "Combining reward shaping and curriculum learning for training agents with high dimensional continuous action spaces," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 1391–1393.
[54] A. C. Tenorio-Gonzalez, E. F. Morales, and L. Villasenor-Pineda, "Dynamic reward shaping: training a robot by voice," in *Ibero-American conference on artificial intelligence*, 2010, pp. 483–492.
[55] G. Konidaris and A. Barto, "Autonomous shaping: Knowledge transfer in reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 489–496.

[56] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
[57] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-Driven Grasp Synthesis - A Survey," *Transactions in Robotics (IEEE)*, vol. 30, no. 2, pp. 289–309, Apr. 2016.
[58] T. Hodan, J. Matas, and S. Obdrzalek, "On Evaluation of 6D Object Pose Estimation," presented at the European Conference on Computer Vision, Oct. 2016.
[59] R. Brégier, F. Devernay, L. Leyrit, and J. Crowley, "Defining the Pose of any 3D Rigid Object and an Associated Distance," *International Journal of Computer Vision*, pp. 571–596, Nov. 2017.
[60] M. Gualtieri, A. ten Pas, K. Saenko, and R. Platt, "High precision grasp pose detection in dense clutter," presented at the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea (South), Oct. 2016. [Online]. Available: https://arxiv.org/abs/1603.01564
[61] R. Suarez and M. Roa, "Grasp quality measures: review and performance," *Autonomous Robots*, vol. 38, pp. 65–88, Jul. 2014.
[62] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3D object grasp synthesis algorithms," *Robotics and Autonomous Systems*, 2011.
[63] Y. Jiang, S. Moseson, and A. Saxena, "Efficient Grasping from RGBD Images: Learning using a new Rectangle Representation," May 2011.
[64] A. Zeng, S. Song, K.-T. Yu, E. Donlon, and F. Hogan, "Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching," Brisbane, QLD, Australia, 2018, pp. 3750–3757. doi: 10.1109/ICRA.2018.8461044.
[65] I. Popov *et al.*, "Data-efficient Deep Reinforcement Learning for Dexterous Manipulation," *arXiv:1704.03073*, Apr. 2017.
[66] J. Mahler and K. Goldberg, "Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences," Mountain View, United States, Nov. 2017, pp. 515–524.
[67] A. Sehgal, H. M. La, S. J. Louis, and H. Nguyen, "Deep Reinforcement Learning using Genetic Algorithm for Parameter Optimization."
[68] G. Zuo, J. Lu, K. Chen, J. Yu, and X. Huang, "Accomplishing Robot Grasping Task Rapidly via Adversarial Training," Irkutsk, Russia, Aug. 2019.
[69] C. Chen, H.-Y. Li, X. Zhang, X. Liu, and U.-X. Tan, "Towards Robotic Picking of Targets with Background Distractors using Deep Reinforcement Learning," Beijing, China, Aug. 2019.
[70] B. Li, T. Lu, J. Li, N. Lu, Y. Cai, and S. Wang, "ACDER: Augmented Curiosity-Driven Experience Replay," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, Aug. 2020, pp. 4218–4224.
[71] A. Pore and G. Aragon-Camarasa, "On Simple Reactive Neural Networks for Behaviour-Based Reinforcement Learning," presented at the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, Aug. 2020.
[72] L. Marzari, A. Pore, D. Dall'Alba, G. Aragon-Camarasa, A. Farinelli, and P. Fiorini, "Towards Hierarchical Task Decomposition Using Deep Reinforcement Learning for Pick and Place Subtasks," *arXiv*, no. arXiv:2102.04022.
[73] M. Anca and M. Studley, "Twin Delayed Hierarchical Actor-Critic," presented at the 7th International Conference on Automation, Robotics and Applications (ICARA), Prague, Czech Republic, Feb. 2021.

[74] D. Morrison, P. Corke, and J. Leitner, "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach," Pittsburgh, PA, USA, Jun. 2018.
[75] B. Wu, I. Akinola, and P. K. Allen, "Pixel-Attentive Policy Gradient for Multi-Fingered Grasping in Cluttered Scenes," presented at the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, Nov. 2019.
[76] Y. Deng *et al.*, "Deep Reinforcement Learning for Robotic Pushing and Picking in Cluttered Environment," presented at the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, Nov. 2019.
[77] T. Kim, Y. Park, Y. Park, and I. H. Suh, "Acceleration of Actor-Critic Deep Reinforcement Learning for Visual Grasping in Clutter by State Representation Learning Based on Disentanglement of a Raw Input Image," *arXiv:2002.11903v1*, Feb. 2020.
[78] G. Brockman and V. Cheung, "Gym," *OpenAI*, Oct. 01, 2021. https://gym.openai.com/
[79] A. Howard, "Gazebo," *Gazebo Sim*, Dec. 13, 2021. http://gazebosim.org/
[80] T. Erez, Y. Tassa, and E. Todorov, "Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX.," Seattle, WA, USA, 2015, pp. 4397–4404. doi: doi: 10.1109/ICRA.2015.7139807.
[81] DeepMind, "Opening up a physics simulator for robotics," *DeepMind*, Oct. 18, 2021. https://www.deepmind.com/blog/opening-up-a-physics-simulator-for-robotics (accessed Jul. 11, 2022).
[82] E. Coumans, "Tiny Differentiable Simulator," *Bullet Real-Time Physics Simulation*, May 21, 2021. https://pybullet.org/wordpress/
[83] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, "Multi-Goal Reinforcement Learning Enviroments for Simulated Franka Emika Panda Robot," *arXiv*, no. 2106.13687v1.
[84] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, Nov. 2021.
[85] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama, "Optuna Documentation," Release 2.10.0.
[86] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," Anchorage, AK, USA, Aug. 2019.
[87] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, "Algorithms for Hyper-Parameter Optimization," Granada Spain, Dec. 2011, pp. 2546–2554.
[88] M. Kiran and M. Ozyildiri, "Hyperparameter Tuning for Deep Reinforcement Learning Applications," *arXiv:2201.11182v1*, Jan. 2022.
[89] S. Zhang and N. Jiang, "Towards Hyperparameter-free Policy Selection for Offline Reinforcement Learning," *arXiv:2110.14000v3*, Nov. 2021.
[90] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, and M. G. Bellemare, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, Feb. 2015.
[91] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," presented at the 5th International Conference on Learning Representations, Palais des Congrès Neptune, Toulon, France, Apr. 2017.

[92] "Franka Emika DataSheet Robot - Arm & Control," *Franka Emikda*, Apr. 2020. https://pkj-robotics.dk/wp-content/uploads/2020/09/Franka-Emika_Brochure_EN_April20_PKJ.pdf (accessed Jul. 13, 2021).
[93] F. Emika, "Panda's Instruction Handbook."
[94] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," presented at the Conference: ICRA Workshop on Open Source Software, Kobe, Japan, May 2009.
[95] Michael Görner, Robert Haschk, Helge Ritter, and Jianwei Zhang, "MoveIt! Task Constructor for Task-Level Motion Planning," presented at the 2019 International Conference on Robotics and Automation, Palais des congres de Montreal, Montreal, Canada, May 2019.
[96] E. Coumans and Y. Bai, "PyBullet Quickstart Guide." https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3
[97] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, and Rachel Fong, "Hindsight Experience Replay," presented at the 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA., Dec. 2017.
[98] Meng Fang, Cheng Zhou, Bei Shi, Boqing Gong, Jia Xu, and Tong Zhang, "DHER: Hindsight Experience Replay for Dynamic Goals," presented at the Seventh International Conference on Learning Representations (ICLR), New Orleans, Louisiana, USA, May 2019.
[99] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger, "Densely Connected Convolutional Networks," presented at the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, Jul. 2017. doi: 10.1109/CVPR.2017.243.
[100] K. He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, Jun. 2016.
[101] Christian Szegedy *et al.*, "Going deeper with convolutions," presented at the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, Jun. 2015.
[102] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Dec. 2012.
[103] Leslie Smith, "Cyclical Learning Rates for Training Neural Networks," presented at the 2017 IEEE Winter Conference on Applications of Computer Vision (WACV), Santa Rosa, CA, Usa, Mar. 2017.
[104] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," 2017, pp. 23–30. doi: 10.1109/IROS.2017.8202133.
[105] S.-W. Huang, C.-T. Lin, S.-P. Chen, Y.-Y. Wu, P.-H. Hsu, and S.-H. Lai, *Cross Domain Adaptation with GAN-based Data Augmentation*, vol. 11213. Munich, Germany: Springer, Cham, 2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-01240-3_44#citeas
[106] A. Ng and S. Russell, "Algorithms for Inverse Reinforcement Learning," presented at the ICML '00:, San Francisco, CA, USA, 2000.
[107] T. Osa, J. Pajarinen, G. Neumann, A. Bagnell, P. Abbeel, and J. Peters, "An Algorithmic Perspective on Imitation Learning," *Foundations and Trends in Robotics*, vol. 7, no. 1–2, pp. 1–179, 2018.

[108] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, pp. 4565–4573, 2016.
[109] Sinno Jialin Pan and Qiang Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, Oct. 2009.

# Appendices

## Appendix A

### The Gym Communication Framework

The methodology for implementing Gym proceeds as follows. An instance of the environment is instantiated with the "make" method. The environment is reset to the base position with the environment reset method. Reset returns the state information for the agent, which traditionally is a vector describing the state position, or an image representing the state position. Actions can be specified or generated at random, following which the actions may be executed by calling the environment step method. The environment step returns the new state position, the reward for the position, a Boolean describing whether the target position is reach (True/False) and any additional information describing the new state.

**Interacting with the Agent:**

```
env = gym.make('PandaReachDense-v2')

obs = env.reset()

action = env.action_space.sample() # random action
obs, reward, done, info = env.step(action)
```

**Output from the Agent:**

```
Selected Action:  [ 0.14633748 -0.51242566 -0.8687553 ]
Updated Robot Gripper Position:  [ 0.04756706 -0.01814914  0.17448885]
Target Object Position:  [-0.03799899  0.1119605   0.00487336]
Reward:  -0.23025960374620155
```

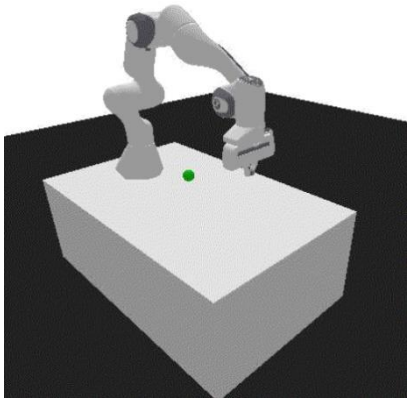


**Figure 36: Panda-gym interactions**

## Appendix B

### Gazebo-Environment Rendering with Kinect Camera

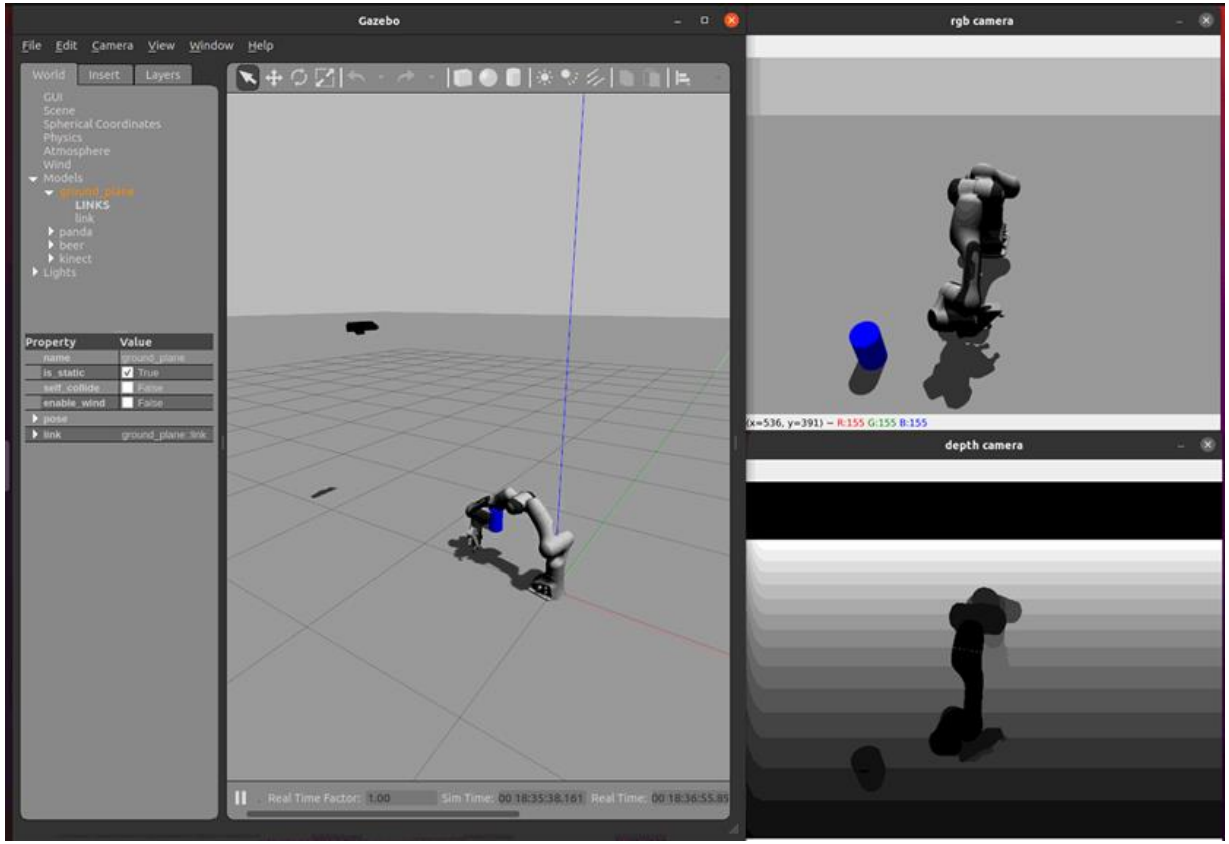



**Figure 37: Gazebo Environnent**

# Appendix C

## Code Explanation

### Environment Modifications:

The base Panda PyBullet classes are the simulation environment, the robot, and the task.

**Simulation Environment Class**: The simulation environment class specifies many characteristics of the environment and model. Examples include number of sub-steps, render characteristics, and base methods which are called to set model dynamics and to instantiate geometries. The primary modification to the simulation environment class was to the render method, which had to be modified to return both depth and RGB image sensor feedback.

**Panda Class:** Panda, the PyBullet robot, is derived from the PyBulletRobot base class. The Panda class creates a robot based on the robotic URDF file (in this case generated by Franka Emika) and sets the PyBullet settings for friction, joint forces, and positions joint indices for the instantiation of the robotic arm. The Panda class has custom methods which are implemented for setting actions, retrieving positions, observations, etc. The main modification to this class were to increase the gripper force and the gripper friction coefficients to enable grasping and manipulation of target objects.

**Task Environment Class:** Most of the changes required to create custom PyBullet gyms were done in the task environment class. Each task required reward shaping, the assignment of goal positions, and modification to the action vectors to enable the completion of each of the tests. For pick-and-place, blocks for stacking and moving were instantiated at random for grasping.

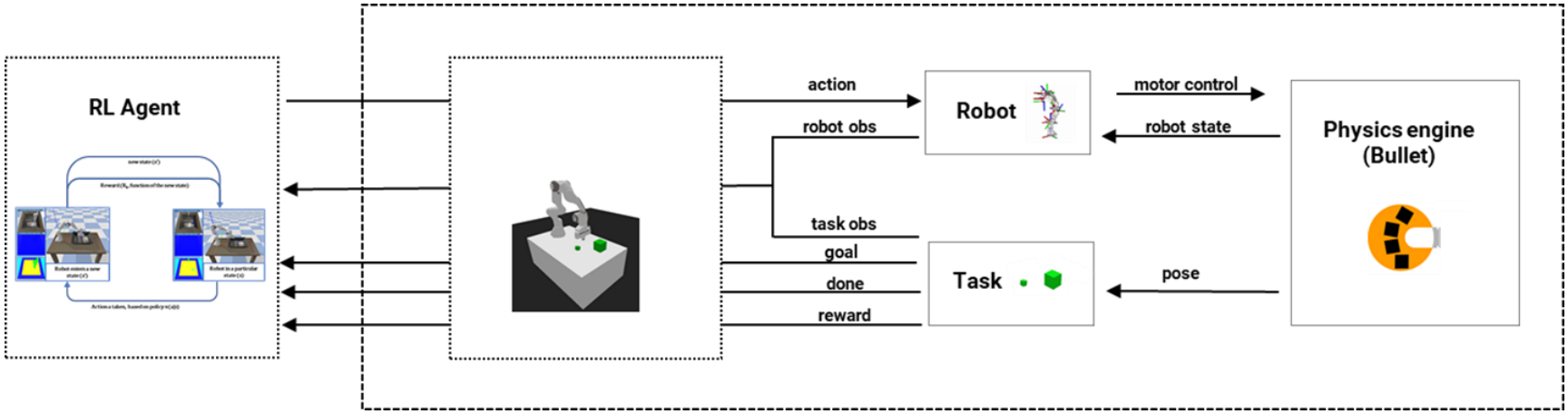



**Figure 38: The Panda PyBullet Framework**

## Appendix D

### Gym Environnent Communication

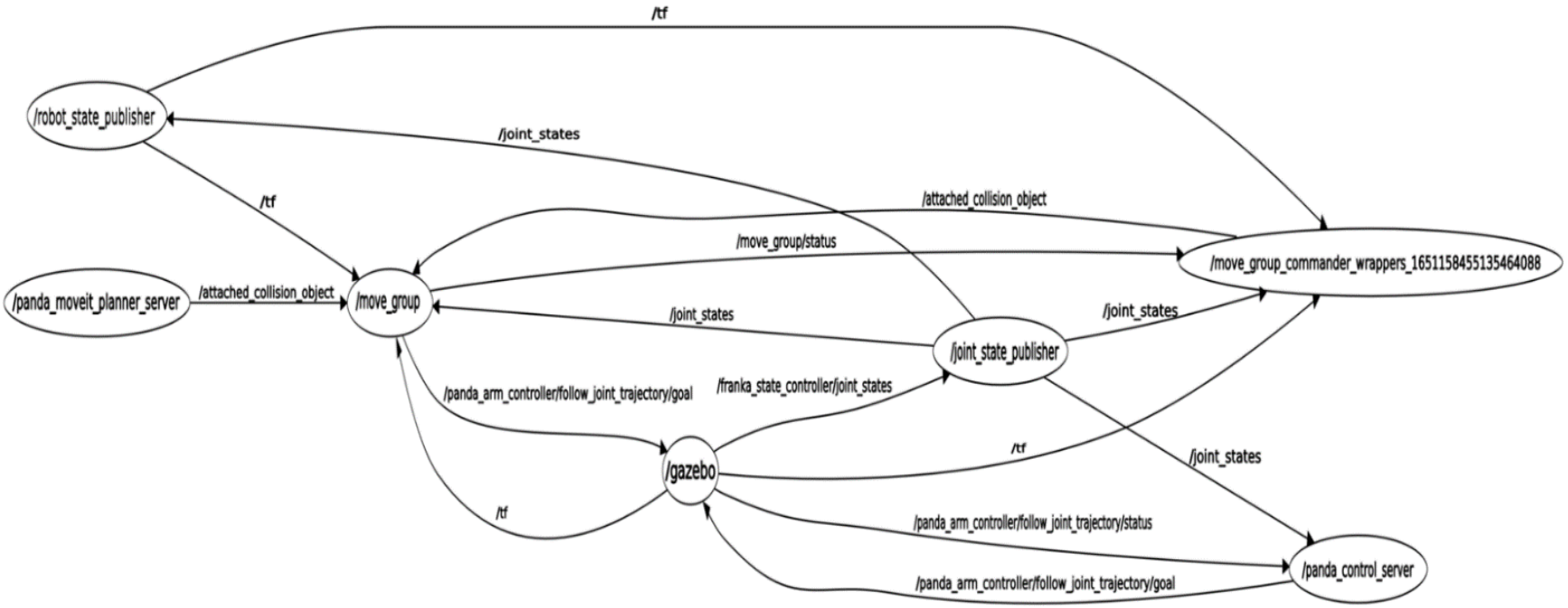



**Figure 39: ROS Communication Node Graph**