Robot Reinforcement Learning for Object Isolation

Teodor Åstrand

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6255
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

This thesis employs deep reinforcement learning, a branch of machine learning, to carry out robotic tasks. The objective centers around teaching an agent controlling a seven-axis robot arm with a gripper tool, to complete an *object isolation* task. For this task, a robot manipulates a cluttered environment in such a way that a predetermined target object becomes isolated. Sub-tasks were developed to explore simpler robot tasks to evolve and combine into more complex tasks, where the goal was the *object isolation* task. Agent training took place in a simulated robot learning environment with the use of primarily a coordinate-based low dimensional state-space, where reward shaping was the primary tool to teach a given task.

The reinforcement learning algorithm Proximal policy optimization (PPO) implemented with a neural network architecture was used to train agents for the robotics tasks and the robot arm's joint velocities were used as the action-space for the agents. Multiple experiments were conducted for agents practicing different tasks and their performance was evaluated by measuring their task completion rate and rendering their behavior among others. Agents developed policies capable of different forms of cube manipulation and performing cube extraction tasks. Multiple different policies for completing robot tasks were learned, and their strategies were evaluated and discussed.

Acknowledgments

I extend my thanks to my supervisor Yiannis Karayiannidis for his guidance and support in reinforcement learning topics and helping with the direction of the thesis. I would also like to give a big thanks to my supervisor Alexander Dürr for presenting this topic, helping with programming, and providing insights into reinforcement learning topics. Furthermore, I would also like to thank my examiner Björn Olofsson for valuable feedback.

Contents

1 Introduction

Combining machine learning and robotics has led to a new era of intelligent systems and automation. Robots were previously confined to pre-programmed tasks relying on human-designed controllers with limited ability to carry out intelligent decisions on their own. The integration with machine learning unveils new opportunities for intelligent autonomous decisions, allowing robots to learn from and adapt to dynamic tasks. [Arents and Greitans, 2022]

For this thesis, tasks carried out by a robot arm will be in focus. Through deep reinforcement learning (RL), a pillar in machine learning, robots learn to carry out tasks through interaction with an environment where a reward encourages correct behavior. As an approach to solving the robotics tasks the reinforcement learning algorithm Proximal policy optimization (PPO) [Schulman et al., 2017] implemented with a neural network architecture was used.

1.1 Research questions

The purpose of this thesis was to teach a reinforcement learning agent controlling a seven-axis robot arm to carry out an *object isolation* task. For this task, an agent manipulates a cluttered environment in such a way that a predetermined object becomes extractable. This is to be done by removing surrounding objects such that a collision-free zone is created around the target object. To accomplish this, the thesis aims to answer the following research questions:

- **1.** How can reinforcement learning be utilized to train agents to carry out robot actions such as reaching, pushing, grasping, and avoiding?

- **2.** How can reinforcement learning be utilized to train an agent to complete an *object isolation* task utilizing and combining the action primitives from *research question 1*?

1.2 Research approach

To achieve the objectives set by the research questions, reinforcement learning agents were trained to carry out sub-tasks that in different ways support the final goal task of an *object isolation* task. For all tasks, cubes were used as the interactive objects, and the tasks were designed to explore cube manipulation, multi-cube interaction, and randomized environment interaction.

The purpose of the sub-tasks was to explore how to teach the primitive robot actions of reaching, pushing, grasping, and avoiding. Following this, the focus shifted to explore how these actions could be combined and evolved to solve more complex problems where the ultimate goal was the *object isolation* task.

The agents trained were all taught from scratch without any prior demonstrations where reward shaping was used as the primary tool to achieve the tasks. Low-dimensional observations were used as a state-space for the agents, and their actions directly controlled the joint velocities of the robot arm.

The agents were trained and the tasks were developed in the robot learning environment RLBench [James et al., 2020], where the robot used was the seven-axis Franka Emika Panda robot [Haddadin et al., 2022] with a gripper tool. During the training and evaluation of agents, a primary objective was that the strategies developed to solve the tasks could effectively be transferred to a real robot.

1.3 Related work

The *object isolation* task is a common task whose name and definition may differ slightly. However, at its core, it involves manipulating a group of objects such that a free zone is created around a target object. When using reinforcement learning to solve this task, multiple different approaches exist. Both [Sarantopoulos et al., 2020] and [Berscheid et al., 2019] explored using reinforcement learning for an *object isolation* task with different approaches.

In [Sarantopoulos et al., 2020] a split Deep Q-Network (DQN) was proposed where each network handles different actions. The state was extracted by generating a topographic map from a camera mounted to the end-effector. The object manipulation approach consisted of sliding and pushing using a mounted rod-shaped tool. As actions the agents could carry out pushing motions where the agent would determine the pushing trajectories.

In [Berscheid et al., 2019] the objective focused on teaching shifting an object's position to make it graspable, while also exploring the generalization of their model by training and testing on different objects. Also, a vision-based state supplied from a camera mounted to the robot was used. As actions, the agent would directly command the gripper's positioning.

The aim of this thesis focuses on the same tasks as these papers, however, a different approach is taken. This thesis gives the reinforcement learning agent direct control of the low-level actions of the robot by letting it command the robot's joint velocities. This means that the agent has to learn all robot actions from scratch and cannot utilize predefined actions such as the pushing in [Sarantopoulos et al., 2020]. Therefore, this thesis focuses on teaching sub-tasks that evolve into more complex tasks to gain insight into how an agent can carry out robot tasks using primitive actions. Furthermore, the used state-space differs from [Sarantopoulos et al., 2020] and [Berscheid et al., 2019] since a low-dimensional coordinate-based state-space was used for this thesis.

Proximal policy optimization (PPO) is a common reinforcement learning algorithm and in [Shahid et al., 2020] PPO was used for learning continuous control actions for robotic grasping. For the task, dense reward functions were used to encourage learning for the agent where agents were trained to generalize grasping strategies to an array of objects.

The use of reward shaping was employed in this thesis as in [Shahid et al., 2020]. This differs from for example [Sarantopoulos et al., 2020] where sparse-rewards were used. The approach of using reward shaping as a primary guide for agent behavior was selected in this thesis to gain insight into how to design reward functions to carry out a robotic *object isolation* task.

Having a reliable simulation environment is important for training agents to perform robotics tasks. The robot learning environment RLBench was used as an environment for this thesis. The benchmark includes a vast library of tasks and supports multiple machine-learning approaches to solving the tasks. [James et al., 2020] The benchmark also includes examples of how tasks can be designed in a simulated environment.

1.4 Robot control and machine learning

Controlling a robot arm becomes a complex problem because of the multiple joints and non-linear dynamics. Inverse kinematics is a fundamental concept of determining the required joint positions to achieve the desired pose and orientation of a robot end-effector. [Bajd et al., 2013]

For arms with a lower number of degrees of freedom, analytical approaches can be used to solve inverse kinematics problems, however, a problem arises as a result of the non-linearity of kinematic equations and the possibility of multiple solutions achieving the same end-effector position. This problem becomes amplified for more complex arms with multiple joints. In this case, kinematics problems can be solved numerically by using an iterative algorithmic approach. [Bajd et al., 2013]

Planning a robot's path is a common practice for industrial applications with repetitive tasks such as welding, material handling, or assembly. An industrial robot in its simplest form needs to be pre-programmed for each task and is incapable of carrying out intelligent decisions, learning, or adapting to a dynamic environment. New robot tasks result in the need for reprogramming and an uncertainty or change during the robot task can lead to failure. By introducing AI in robotics through implementing technologies such as AI-based computer vision, deep reinforcement learning, or imitation learning, robots gain the possibility of knowledge acquisition, generalization, and adaptive planning. [Arents and Greitans, 2022] This results in the possibility of intelligent decision-making within dynamic tasks.

Working in a dynamic environment becomes vital for collaborative robots working together with humans. By developing methods for robots to act and respond intelligently, human-robot collaboration can become more natural. Advances in computer vision and deep learning enable robots to detect and classify objects. This can be used to handle task-specific information by assessing a scene and acting accordingly. Furthermore, robust object detection can provide safety for humans working in the vicinity, e.g., if they are in the way of a robot. [Arents and Greitans, 2022]

By introducing reinforcement learning for robotics tasks, a robot can learn a vast variety of tasks through interaction. A reinforcement learning-based robot controller can be used for a wide array of situations making it adaptable for different tasks variations and tasks with irregularities. Furthermore, the use of reinforcement learning can eliminate the need for planning or inverse kinematics since an agent can be successful by directly controlling a robot's joints.

With the rise of agile robots such as Atlas [Boston Dynamics, 2024a] and Spot [Boston Dynamics, 2024b] from Boston Dynamics, robots are now becoming more capable of navigating and being a part of an urban environment. For these types of robots, robust walking strategies are important. The utilization of pre-coded human-designed walking strategies becomes limiting and difficult to adapt to a wide array of situations. With the use of deep reinforcement learning walking strategies can be taught through interaction in a way similar to how a living creature would learn from prior experiences. Reinforcement learning also allows robots to adapt their walking strategies to different terrains by learning from different environments. In [Wu et al., 2023] a quadruped walking robot learned walking strategies and the ability to recover from a tumble by only interacting with a real-life environment.

Furthermore, OpenAI has been successful in creating agents controlling robot hands capable of precise dexterity. Reinforcement learning algorithms were leveraged to teach the robot hand to manipulate a cube similarly to a human hand, making it capable of rotating the cube's orientations and even solving a Rubik's cube one-handedly. [Akkaya et al., 2019]

2

Reinforcement learning

Reinforcement learning (RL) is a pillar of machine learning that draws from fields such as computer science, mathematics, neuroscience, economics, engineering, and psychology, which revolves around developing strategies that achieve desired behavior through a trial-and-error process. [Silver, 2015] At its simplest form, reinforcement learning strives to develop an agent that through interaction with an environment develops behavior that maximizes an accumulated reward.

Within RL, a reward R_t is a scalar feedback signal that indicates an agent's performance at step t. The purpose of the agent is to choose actions that maximize the reward accumulated in the future. These rewards are not always instantaneous. Sometimes a sequence of low reward-yielding actions can lead to a high reward-yielding action. [Silver, 2015]

Reinforcement learning is one of the main pillars of machine learning together with supervised learning where an agent learns from an external supervisor providing examples, and unsupervised learning where an agent finds structure in a collection of data without any external supervisors. Reinforcement learning differs from these pillars of machine learning since the main focus is to maximize a reward signal and not to find a structure or rely on external guidance. [Sutton and Barto, 2018]

Reinforcement learning can be applied to a variety of fields such as robot control, optimal control, autonomous driving, and strategic games. Applying reinforcement learning has led to agents being successful in playing Atari games at a high level, executing advanced maneuvers for a remote-controlled stunt helicopter, and controlling a power station. [Silver, 2015] As mentioned previously, reinforcement learning has also been successful in robotic tasks, robot walking strategies, and dexterous robot manipulation.

Solving robotics problems with reinforcement learning often involves training in a simulated environment where an agent can iteratively learn from experience in less time than real-world training would allow. Once a satisfactory policy has been achieved this agent can be deployed on a physical robot.

2.1 Markov decision process

Reinforcement learning problems can be formalized in a Markov decision process (MDP), meaning that the process will consist of a finite set of states S and actions A, a state transition probability matrix P, a reward function R, and a discount factor γ. Together they describe the MDP as the tuple (S, A, P, R, γ). [Otterlo and Wiering, 2012]

Selecting appropriate states and actions, and defining a good reward function for a reinforcement learning problem is crucial. A state represents a unique characterization of all that is important for the modeled problem. [Otterlo and Wiering, 2012] For a given state to be considered a Markov state it must include all relevant information so that the history is no longer needed. A Markov state means that future states are independent of the history given the present state. [Silver, 2015] Actions are what an agent can use to influence and control the state of a system. For a given state, multiple actions are possible where each leads to a different outcome. [Otterlo and Wiering, 2012] To give certain states or actions a value, rewards are used. A reward function determines rewards for being in a state or acting in a state. This ultimately serves as a guide to reaching a goal by giving direction in which way the MDP should be controlled. [Otterlo and Wiering, 2012]

When defining a problem as a Markov decision process the environment in which an agent should interact is assumed to be fully observable and contain all relevant information for each current state. This assumption allows decision-making based solely on the present state without the need to take history into account.

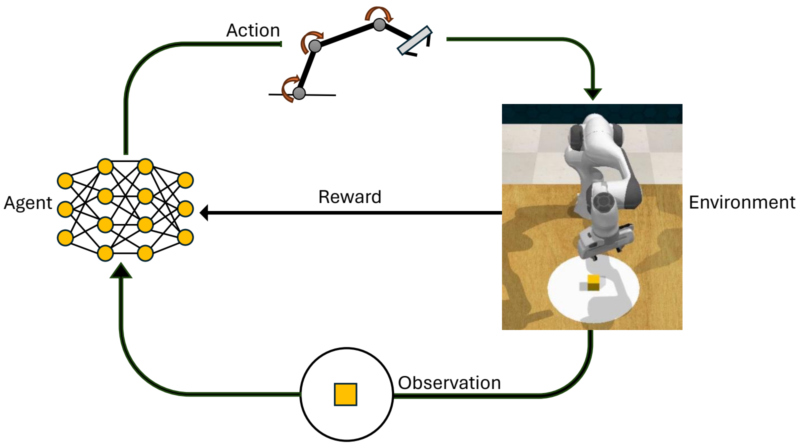


Figure 2.1 An overview of an agent's interaction with a simulated robot environment. For this example, the agent emits actions influencing the robot and receives observation of the robot's environment and rewards as feedback.

2.2 Agent

Agents are often described as the brains of a reinforcement learning system and are responsible for decision-making and environment interaction. An agent's interaction with the environment in step t is visualized in Figure 2.1. For each step, an agent emits an action to the environment and receives a new state and reward, which the agent uses to adapt its behavior. For this example, the action is controlling the joint velocities of a robot and the state consists of an object's position. Multiple approaches to designing a reinforcement learning agent exist, where the fundamental components are [Silver, 2015]:

Policy The behavior of an agent is referred to as its policy π and is what determines what action to execute from a given state. The purpose of an agent is to achieve a policy that knows how to carry out actions that achieve the most rewards, hence solving a given task. [Silver, 2015] As a general description, a policy can be viewed as a set of instructions or a map for how the agent should act given its current perception of the environment. A simple policy can take the form of a look-up table while more complex policies use more advanced methods. [Sutton and Barto, 2018] Policies can either be deterministic

$$a = \pi(s), \tag{2.1}$$

where each state s has only one corresponding action a or stochastic

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s], \tag{2.2}$$

where the policy specifies probabilities for each action at a given state.

Value function To evaluate the value of a given state and aid with selecting actions a value function can be used. The value function is used to calculate the expected total future reward for a given state from following a policy π, and can be described as

$$V_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right], \tag{2.3}$$

where a discount factor γ is used to weigh the value of current and future rewards. [Silver, 2015]

Model A model corresponds to the agent's representation of the environment and is used to predict what the environment will do next. Model-based reinforcement learning can utilize planning. With the use of a model, the next state and reward can be anticipated from a current state and action. However, to be successful with this approach an accurate model of the environment is crucial. In this thesis, the approach of model-free learning was pursued. For this type of learning the agent has no prior model and needs to rely on learning how an environment works through a trial-and-error process. [Sutton and Barto, 2018]

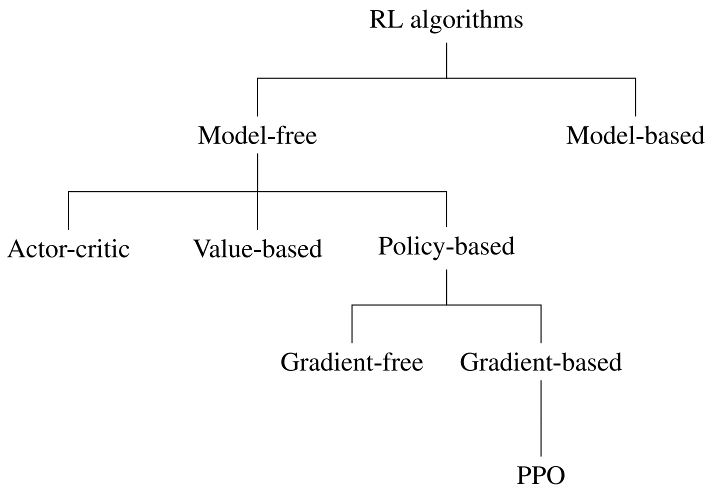


Figure 2.2 An overview of RL algorithm classifications, specifically showing how PPO can be categorized.

2.3 Reinforcement learning methods

By utilizing components such as policy, value function, and model different algorithms can be tailored. This leads to a wide array of approaches where methods can be customized for specific problems. The three fundamental reinforcement learning methods are categorized as *value-based, policy-based,* and *actor-critic*.

For value-based methods, an agent learns a value function that maps the value of being in a given state. The policy is thereafter defined manually to select states of high value. This differs from policy-based methods where the primary objective for an agent is to learn a policy and not a value function. [Simonini and Sanseviero, 2023]

Actor-critic methods inherit properties from both policy-based and value-based methods by combining a policy-based actor that determines agent behavior, with a value-based critic that evaluates how good actions taken by the actor are. [Simonini and Sanseviero, 2023]

From these categories, a plethora of different algorithms utilizing different methods and approaches to solve reinforcement learning problems exist. For this thesis, Proximal policy optimization (PPO) was used. PPO has showcased good ability for Atari gameplay and learning humanoid walking in [Schulman et al., 2017]. Furthermore, in [Shahid et al., 2020] PPO was used for robot grasping tasks and the algorithm was therefore seen fit to use for the robot tasks involved in this thesis.

2.4 Deep learning

In this thesis, deep reinforcement learning has been utilized. What differs between traditional algorithms and deep learning is the use of deep neural networks to solve the given reinforcement learning problems. For instance, for policy-based methods neural networks can be used to find the optimal policy. [Simonini and Sanseviero, 2023]

A neural network consists of layers of interconnected nodes inspired by a mammalian brain structure of neurons. This leads to similarities in the decision-making process for a neural network and a biological brain, where the network of nodes is used to make conclusions based on data. The fundamental process of learning for artificial neurons consists of changing connections between the neurons. Each neuron can be seen as a binary operator that activates when a threshold, also referred to as bias, is exceeded where connection strength between neurons is determined by weights. A network can be trained from a set of input data together with a list of values to target, by adjusting the connection strengths between neurons to return the targeted value from the given data. [Mehlig, 2021]

A standard neural network is built of layers of nodes where the first one is an input layer, the last one is an output layer, and between are intermediate hidden layers. For example, when a network learns to identify handwritten numbers the input layer receives images of handwritten numbers and the output layer emits what number the model believes it is. This is based on how the intermediate layers have been configured during the training process where the weight and biases for all nodes have picked up on patterns during the training process. What makes a process considered a deep learning process is that multiple layers are used in the network. [Sanderson, 2017]

As a basic description, the learning process for a neural network consists of the network adjusting its weights and biases to emit accurate answers based on its input data. As an example, a policy implemented using a neural network should learn to emit the optimal action given a state, and a value function neural network should learn to accurately predict expected cumulative rewards from states.

2.5 Proximal policy optimization (PPO)

The primary reinforcement learning algorithm utilized for the robotics tasks in this thesis was Proximal policy optimization (PPO) that, as seen in Figure 2.2, falls under gradient-based methods, a subclass of policy-based methods. For policy-based methods the main objective is to find the optimal policy, by finding θ to maximize the objective $J(\theta)$, ultimately maximizing the accumulated reward. What distinguishes gradient-based methods is that the method does this by searching for local maximums in $J(\theta)$ by ascending the policy gradient. [Silver, 2015] For deep reinforcement learning, finding θ corresponds to the network adjusting its weights and biases.

For policy gradient methods, advantage functions serve as guidance of the relative goodness of an action a in a state s, ultimately describing the advantage of an action compared to others. The advantage is calculated as the difference between the on-policy value action function $Q_\pi(s,a)$ and the on-policy value function $V_\pi(s)$,

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s), \tag{2.4}$$

where $Q_\pi(s,a)$ gives the expected reward for taking the action a and then following the policy π and $V_\pi(s)$ returns the average expected reward of all actions according to the policy π. [Achiam, 2018]

PPO focuses on improving its policy at each step and uses certain strategies to restrain the policy from changing too much at the time and risking a performance collapse. The utilized version of PPO for this thesis can be referred to as PPO-clip. This version optimizes an objective function and uses clipping to remove the incentive for a new policy to veer far from the old policy. [Schulman et al., 2017]

To achieve this, the ratio

$$r_t(\theta) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)}, \tag{2.5}$$

between the former and the current policy, is clipped in a range $[1-\varepsilon, 1+\varepsilon]$. This is done by introducing a surrogate objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right] \tag{2.6}$$

The objective function selects the minimum of the product of a $r_t(\theta)$ or the clipped version of $r_t(\theta)$ and the estimated advantage $\hat{A}_t$. The clipping makes the function constrained since the ratio $r_t(\theta)$ is penalized if it diverges too far from 1. Hence, large policy changes are prevented and more careful policy updates are encouraged. To implement PPO using a neural network architecture a loss function combining the clipped surrogate objective L^{CLIP} and the value function error term L_t^{VF} is used. This is referred to as an actor-critic PPO implementation where the policy serves as an actor and the value function as a critic. To encourage the algorithm to explore an entropy bonus can be added. [Schulman et al., 2017]

For this thesis, a version of PPO implemented by Stable-Baselines3 [Raffin et al., 2021] was utilized. This version uses clipping for the surrogate objective function and uses a neural architecture based on PyTorch [Paszke et al., 2019]. The implementation consists of the actor-critic style policy with a policy and a value function network. The implementation utilizes a clipped surrogate objective function as in Equation (2.6). In addition to this a value loss function $L_t^{VF}(\theta)$ that aims to minimize the difference between actual values and predicted values from the value function is included. Furthermore, the algorithm implementation has the option of including an entropy bonus $Ent(\pi_\theta)$ used to encourage exploration. Together a total

PPO loss function is constructed as:

$$L_t(\theta) = -L_t^{CLIP}(\theta) + c_1 L_t^{VF}(\theta) + c_2 Ent(\pi_\theta), \tag{2.7}$$

where c_1 and c_2 are parameters for scaling. The pseudo-code in Algorithm 1 gives an overview of how the implementation works. [Raffin et al., 2021]

Algorithm 1 PPO Clip

```
 1: **Input:** Initial policy and value function parameters.
 2: **for** each policy update step **do**
 3:     Collect a set of trajectories by running policy $\pi$ in the environment for $N$ steps.
 4:     Compute rewards $R_t$.
 5:     Compute advantage estimates $\hat{A}_t$.
 6:     Compute the policy ratio
            $$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.8}$$
 7:     Compute the clipped surrogate objective function
            $$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right] \tag{2.9}$$
 8:     Compute the value loss function $L_t^{VF}(\theta)$ and the entropy bonus $Ent(\pi_\theta)$
 9:     Compute the total loss function.
            $$L_t(\theta) = -L_t^{CLIP}(\theta) + c_1 L_t^{VF}(\theta) + c_2 Ent(\pi_\theta) \tag{2.10}$$
10:     Update the policy by minimizing the loss function through gradient descent.
11: **end for**
```

The loss function is calculated over batches of trajectories from running a policy in an environment. How long an agent interacts with an environment before a policy update, can be determined for each agent task. For example, if a task has a limit of ten steps before resetting, the algorithm could be given fifty steps to interact with the environment before an update. This would give the agent at least five attempts to execute a task and the ability to collect trajectories for each attempt. A trade-off is made between how much information an agent gets to collect before an update and the total time needed to train.

2.6 RLBench

To develop robot tasks, train agents, and simulate their performance, the robot learning environment RLBench [James et al., 2020] was used. RLBench aims to serve as a benchmark for robot learning and is focused on facilitating research combining robot tasks with reinforcement learning, and imitation learning among others. The benchmark provides both proprioceptive and vision-based observations, allowing the use of low dimensional task observations directly extracting positions of objects in a scene or the use of images supplying RGB (red, green, and blue) images, depth, and segmentation masks.

Within RLBench the user has the opportunity to build custom robot tasks with dynamic objects in the form of simple shapes or more advanced 3D models. Tools such as proximity and touch sensors can be utilized to add logic to the task success criteria and reward shaping. [James et al., 2020]

3 Method

The approach chosen to teach agents to carry out desired tasks was through an iterative task design, training, and evaluation process. The method to teach desired behavior was through reward shaping functions guiding the agent to perform each task. Information from the simulator regarding aspects such as the state of an object in a given task was used for the reward shaping functions. For example, tracking the velocity of a cube allowed reward functions based on the movement of the cube.

Once an agent had been trained, its performance was evaluated by rendering its behavior in the simulator and evaluating data such as its success rate, rewards per episode, and episode length. Different types of reward shaping and training parameters were tested until an agent could perform the desired task. Thereafter, a new task would be developed often built upon prior successful tasks.

3.1 Task design

For each task, a custom environment was created for an agent to interact with. Each environment included task-specific objects, sensors, state-space, and reward shaping functions. A maximum amount of steps per episode was set for each specific task, based on the task complexity, amount, or distance the robot end-effector must travel. These steps determine how long the agent had to complete its task, and once the limit was reached the environment reset. For each task, N amount of steps were selected between each policy update, and a total amount of training steps were selected. As a starting point, the N steps would correspond to five times the maximum episode steps, letting the agent collect trajectories for at least five task attempts.

The initial training runs of a task would be done conservatively to evaluate the task functionality. Thereafter, longer runs were carried out until the agent's accumulated reward would stop increasing. An array of different parameters could be tested and tuned within the Stable-Baselines version of PPO. To limit the variability, the altered algorithm-specific parameters were limited to N steps per policy update and the use of an entropy bonus. The entropy bonus uses a corresponding scaling coefficient $c1$ in Algorithm 1. The coefficient was set to 0.01 when in use, and this number was determined from initial runs where some coefficient values were tested to determine a value that can lead to a noticeable impact in the agent learning process.

State-space

As a primary approach, the low-dimensional environment representation was used as a state-space. The utilized state-space for all tasks could be divided into two main parts: The first part includes general information regarding the robot and everything that remains the same for each task, and the second part includes task-specific information such as the position of objects in the scene. Each object was represented in low-dimensional coordinate-based observations, meaning that an object in the robot task scene would be represented as a point in the scene. For example, a cube would be represented solely by its x, y, and z positional coordinates.

Table 3.1 Descriptions of low-dimensional observations.

Low-dimensional observations
General observations for all tasks
• Velocities of the robot's joints • Positions of the robot's joints • Forces experienced by the joints • Pose of the robot's gripper • Positions of the gripper's joints • Forces detected by the gripper
Task-specific observations
• Low-dimensional state of the task

As a secondary approach, the vision-based observation mode consisting of RGB, depth, and segmentation masks provided by a virtual camera in the simulation environment was tried. The purpose was to investigate if the designed task could be carried out using only vision feedback and not a state-space where the locations of all objects were precisely known. A vision-based state-space was not selected as the primary approach since the more advanced and large state-space would lead to drastically longer training time. By creating and refining tasks for the low dimensional state-space they could further be tried utilizing the vision-based state-space.

Action mode

There are multiple approaches to carrying out actions with a robot arm. A reinforcement learning agent can use more high-level action modes involving inverse kinematics and planning or low-level action modes where the agent directly commands a joint's velocity, torque, or position. For this thesis, a low-level action mode was selected consisting of the joint velocities for the seven joints of the Panda robot, and a separate action mode for the gripper. The gripper's action was discrete, namely, the gripper had a closed and an open mode.

The decision to use a low-level action mode was made to gain insight into how an agent can be successful while having complete control of the robot and not only commanding trajectories or positions to a planner that executes the robot movements.

All actions were normalized and the agent could select values between - 1 and 1 for joint and gripper actions. For the gripper, negative values would lead to opening and positive to closing.

For grasping within the simulator a simplified approach utilizing fake grasping was used. For this form of simulated grasping the grasped object becomes frozen at the end-effector during grasping. This results in the dynamics of grasped objects being ignored and the force needed to grasp an object not being considered. This simplification was considered acceptable. However, simplified grasping also allows grasping positions that would be impossible to adapt to the real robot. As exemplified in Figure 3.1, grasping an object by its very edge would work in the simulator. To counter this, tasks were designed specifically to teach good grasping positions.

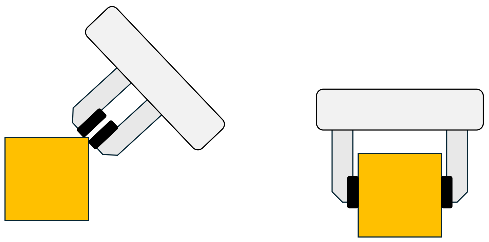


Figure 3.1 An illustration highlighting two acceptable grasp positions within the simulated environment, where the left version displays undesired grasp positioning difficult to implement with a real robot.

3.2 Task categories

To achieve an agent that can complete the *object isolation* task, agents were developed to carry out sub-tasks that support the final goal in their distinct matter. The purpose of the variety of tasks was to achieve agents with specific objectives, and once all these objectives were achieved an agent inheriting all the desired properties could be brought forward. The objectives that the sub-tasks were aimed to develop can be categorized according to:

- **Cube manipulation**
  This could be done through both pushing and grasping strategies and both were explored. Furthermore, improved grasping strategies were explored to find solutions that could be transferred from simulation to a real robot.

- **Multi-cube interaction**
  To work toward completing an *object isolation* task, agents were trained to carry out tasks in an environment with multiple cubes. This involved the manipulation of multiple cubes at the same time and carrying out tasks where distractor cubes served as obstacles for a cube extraction. For a robot to carry out an *object isolation* effectively, it must be capable of extracting a predetermined object with minimal interference with the surrounding objects.

- **Randomized environment interaction**
  For an agent to be able to work in a dynamic environment, it should adapt to situations where the position and orientation of objects are random. To achieve this, agents were trained in environments where cube orientation and position vary from episode to episode.

The task categorization allowed the exploration of action primitives such as reaching, avoiding, pushing, and grasping, and how the action primitives could be combined to solve more complex tasks. Furthermore, the sub-tasks allowed the investigation of different strategies developed by the agent to solve a different task and allowed insight into how the strategies could be influenced to solve tasks more robustly.

3.3 Tasks

The following section gives an overview of the design and reward shaping functions for the robot tasks. The approach to creating the tasks consisted of an iterative process of trying task variations, adjusting reward shaping functions, training the agent, and evaluating their performance until satisfactory performance was reached. The primary tasks explore cube manipulation, thereafter the focus shifts to tasks involving multiple cubes and exploring training agents for randomized environments.

Slide cube to target

To achieve agents that could manipulate a cube by pushing, the task *slide cube to target* was explored. The essential parts of the task were to navigate to the cube and push it into a target zone as shown in Figure 3.2, where the cube and target position were the same for all episodes.

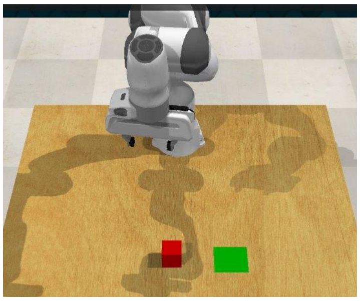


Figure 3.2 The initial setup for the *slide cube to target* task.

Success criterion. Once the red cube has entered the green target section shown in Figure 3.2 the task is deemed successful.

Task-specific low-dimensional observations. The task-specific observations consisted of the cube position and the green square position.

Reward shaping. The reward shaping function $R(s,a,s')$ for the *slide cube to target* task consists of the following parts:

1. **Gripper to cube penalty ($R_{\mathbf{GripperToCube}}$)**: A negative reward based on the Euclidean distance between the robot gripper and the target cube $d_{\text{GripperToCube}}$ scaled by a positive constant α, that decreases for smaller distances.
$$R_{\text{GripperToCube}}(s,a,s') = -\alpha d_{\text{GripperToCube}} \tag{3.1}$$

2. **Cube to target reward ($R_{\mathbf{CubeToTarget}}$)**: A reward based on the Euclidean distance between the cube and the target zone $d_{\text{CubeToTarget}}$ scaled by a positive constant β.
$$R_{\text{CubeToTarget}}(s,a,s') = -\beta d_{\text{CubeToTarget}} \tag{3.2}$$

3. **Cube velocity reward ($R_{\mathbf{CubeVelocity}}$)**: A reward determined by the cube-velocity v_{Cube}, made to encourage cube movement.
$$R_{\text{CubeVelocity}}(s,a,s') = \begin{cases} 1 & \text{if } v_{\text{Cube}} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

4. **Task complete reward ($R_{\textbf{TaskComplete}}$)**: A large reward for fulfilling the task success criterion.

$$R_{\text{TaskComplete}}(s,a,s') = \begin{cases} 1000 & \text{if the task is complete} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

The total reward shaping function can be represented as the sum of these components:

$$\begin{aligned} R(s,a,s') = & R_{\text{GripperToCube}}(s,a,s') \\ & + R_{\text{CubeToTarget}}(s,a,s') \\ & + R_{\text{CubeVelocity}}(s,a,s') \\ & + R_{\text{TaskComplete}}(s,a,s') \end{aligned} \tag{3.5}$$

Remove from zone

The *remove from zone* task was designed to explore multi-cube and randomized environment interaction. The task consisted of a zone represented by a white circle and three cubes as seen in Figure 3.3. The cubes were placed randomly in each episode and the goal was for the agent to find a policy to remove all the cubes from the white zone by any means. No guidance for how the cubes were to be removed was provided in the reward shaping.

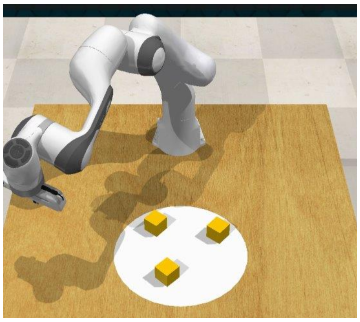


Figure 3.3 The initial setup for the *remove from zone* task.

Success criterion. Once the white zone no longer contains any yellow cubes shown in Figure 3.3 the task is deemed successful.

Task-specific low-dimensional observations. The task-specific low-dimensional observations consisted of all the cube positions and the center position of the zone.

Reward shaping. The reward shaping function $R(s,a,s')$ for the *slide cube to target* task consists of the following parts:

1. **Gripper to zone reward ($R_{\textbf{GripperToZone}}$)**: A penalty based on the Euclidean distance between the robot's gripper and the center of the white zone $d_{\text{GripperToZone}}$ scaled by a positive constant α. Once the end-effector reached the inside of the zone, no penalty was given.

$$R_{\text{GripperToZone}}(s,a,s') = \begin{cases} 0 & \text{if end-effector in zone} \\ -\alpha d_{\text{GripperToZone}} & \text{otherwise} \end{cases} \tag{3.6}$$

2. **Gripper movement in zone reward ($R_{\textbf{GripperMovementInZone}}$)**: A reward based on the end-effector's movement in the zone to encourage exploration.

$$R_{\text{GripperMovementInZone}} = \begin{cases} 0.1 & \text{if end-effector is in the zone and } v_{\text{end-effector}} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

3. **Cube velocity reward ($R_{\textbf{CubeVelocity}}$)**: A reward that is determined by the cube velocity v_{Cube_i} for all N cubes present in the zone. This is to encourage cube movement and hence interaction with the cubes.

$$R_{\text{CubeVelocityTotal}}(s,a,s') = \sum_{i=1}^{N} R_{\text{CubeVelocity}_i}(s,a,s') \tag{3.8}$$

where

$$R_{\text{CubeVelocity}_i}(s,a,s') = \begin{cases} 1 & \text{if } v_{\text{Cube}_i} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

4. **Cube exit reward ($R_{\textbf{Cube Exit}}$)**: A reward for a cube exiting the zone, where each cube could only be rewarded for its exit once. This was to avoid policies where an agent can accumulate a large reward by moving a cube in and out of the zone.

$$R_{\text{CubeExitTotal}}(s,a,s') = \sum_{i=1}^{N} R_{\text{CubeExit}_i}(s,a,s') \tag{3.10}$$

where

$$R_{\text{CubeExit}_i}(s,a,s') = \begin{cases} 500 & \text{if Cube}_i \text{ exits zone for the first time} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

5. **Task complete reward ($R_{\mathbf{TaskComplete}}$)**: A large reward for fulfilling the task success criterion.

$$R_{\text{TaskComplete}}(s,a,s') = \begin{cases} 1000 & \text{if the task is complete} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

The total reward shaping function can be represented as the sum of these components:

$$\begin{aligned} R(s,a,s') =& R_{\text{GripperToZone}}(s,a,s') \\ &+ R_{\text{GripperMovementInZone}}(s,a,s') \\ &+ R_{\text{CubeVelocity}}(s,a,s') \\ &+ R_{\text{CubeExit}}(s,a,s') \\ &+ R_{\text{TaskComplete}}(s,a,s') \end{aligned} \tag{3.13}$$

Grasp extraction

To explore cube manipulation through grasping, the *grasp extraction* task was designed. The task consisted of a zone with a yellow cube in the middle as shown in Figure 3.4 and the objective for the robot was to extract the cube from the zone with grasping. To encourage policies utilizing grasping, shaping rewards favoring grasping were introduced.

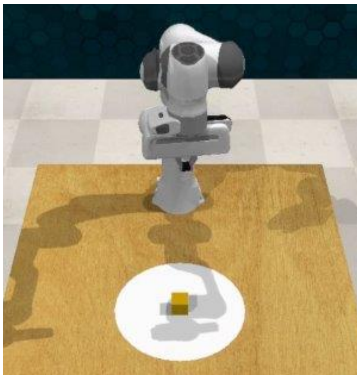


Figure 3.4 The initial setup for the *grasp extraction* task.

Success criterion. Once the yellow cube has exited the white zone shown in Figure 3.4 while being grasped the task was deemed successful.

Task-specific low-dimensional observations. The task-specific observations consisted of the cube position and the center position of the zone.

Reward shaping. The reward shaping function $R(s,a,s')$ for the grasp removal task consists of the following parts:

1. **Gripper to cube reward ($R_{\textbf{GripperToCube}}$)**: A reward based on the Euclidean distance between the robot's gripper and the target cube $d_{\text{GripperToCube}}$ scaled by a positive constant α. For this function, the reward grows as the gripper nears the target cube.

$$R_{\text{GripperToCube}}(s,a,s') = \frac{\alpha}{100 \cdot d_{\text{GripperToCube}} + 1} \tag{3.14}$$

2. **Cube distance from center grasped reward ($R_{\textbf{CubeFromCenter}}$)**: A reward based on the Euclidean distance between the cube and the center of the zone $d_{\text{CubeFromCenter}}$ scaled by the positive constant β, this to encourage the agent to move the cube out of the zone.

$$R_{\text{CubeFromCenter}}(s,a,s') = \begin{cases} \beta d_{\text{CubeFromCenter}} & \text{if cube is grasped} \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

3. **Task complete reward ($R_{\textbf{TaskComplete}}$)**: A large reward for fulfilling the task success criterion.

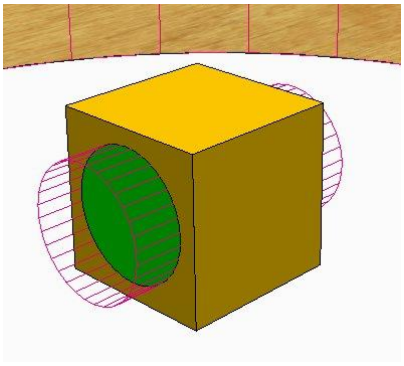
$$R_{\text{TaskComplete}}(s,a,s') = \begin{cases} 1000 & \text{if the task is complete} \\ 0 & \text{otherwise} \end{cases} \tag{3.16}$$

The total reward shaping function can be represented as the sum of these components:

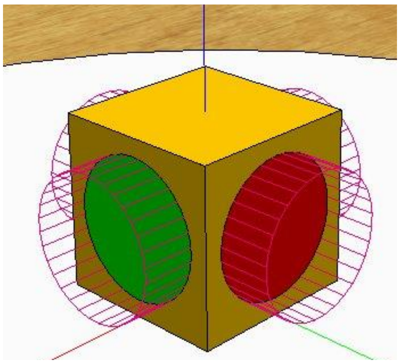
$$\begin{aligned} R(s,a,s') =& R_{\text{GripperToCube}}(s,a,s') \\ &+ R_{\text{CubeFromCenter}}(s,a,s') \\ &+ R_{\text{TaskComplete}}(s,a,s') \end{aligned} \tag{3.17}$$

Extraction with improved grasping

The prior grasping task did not address the issue of impossible grasping positions as in Figure 3.1. To address this, a new task favoring improved grasping positioning was designed. The task was based on the grasp extraction task. However, specific *grasping areas* were introduced on the cube. These areas were used to reward good gripper positioning on the cube. Two versions of this task were tested, one with one pair of *grasping areas* and another with two pairs. Good grasping would be rewarded when both gripper tips would be in one pair of areas.



(a) A cube with one pair of grasping areas.



(b) A cube with two pairs of grasping areas.

Figure 3.5 Renderings showing the grasping areas encouraging specific gripper positioning.

Success criterion. Once the yellow cube has exited the white zone while being grasped the task was deemed successful, where the initial position corresponds to Figure 3.4.

Task-specific low-dimensional observations. For the prior tasks, a cube was only represented in the state-space as its center point coordinate. This gives no information regarding a cube's orientation, hence not enough information to find good grasping positions. For this task, points for each respective grasping area were introduced according to Figure 3.5. The extended observation of the cube now provides information regarding a cube's orientation and enhances the opportunity for finding good grasping strategies. This is exemplified in Figure 3.6.

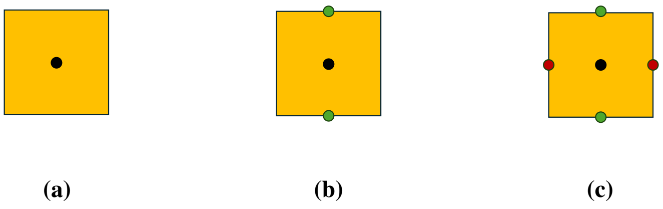


Figure 3.6 Low dimensional representation of the three cube variations where each grasping area is represented as a point on the cube. a) shows a cube without grasp areas, while b) and c) show cubes with one and two pairs of grasp areas.

Reward shaping. The reward shaping function $R(s,a,s')$ for the extraction with improved grasping task consisted of the following parts:

1. **Gripper to cube reward ($R_{\mathbf{GripperToCube}}$)**: Same as Equation (3.14).
2. **Cube distance from center grasped ($R_{\mathbf{CubeFromCenter}}$)**: Same as Equation (3.15).
3. **Good grasping position reward ($R_{\mathbf{GraspingPos}}$)**: A reward for maintaining a good grasping position determined by the zones on the cube.

$$R_{\text{GraspingPos}}(s,a,s') = \begin{cases} 10 & \text{if the grasping is within the grasping zones} \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

4. **Task complete reward ($R_{\mathbf{TaskComplete}}$)**: A large reward for fulfilling the task success criterion. However, for this version, an additional reward is given for completing the task with a good grasp positioning.

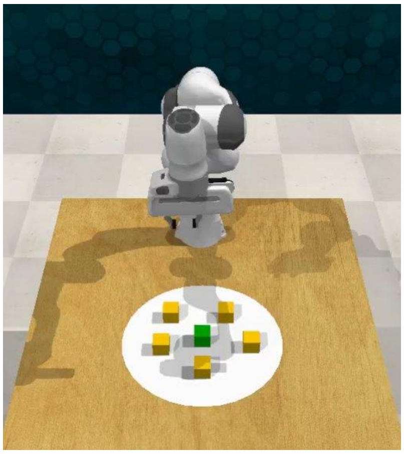
$$R_{\text{TaskComplete}}(s,a,s') = \begin{cases} 1000 & \text{if the task is complete} \\ 2000 & \text{if the task is complete with good grasping} \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

The total reward shaping function can be represented as the sum of these components:

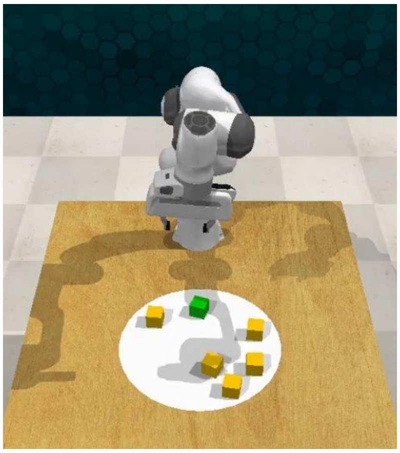
$$\begin{aligned} R(s,a,s') =& R_{\text{GripperToCube}}(s,a,s') \\ &+ R_{\text{CubeFromCenter}}(s,a,s') \\ &+ R_{\text{GraspingPos}}(s,a,s') \\ &+ R_{\text{TaskComplete}}(s,a,s') \end{aligned} \tag{3.20}$$

Extraction with distractors

To teach an agent to interact with a cube for a scenario involving multiple cubes, an extraction task including distractor cubes was created. The task built on the prior *grasp extraction* tasks still strongly rewarding grasping strategies. However, for this task to be deemed successful the green cube, shown in Figure 3.7, must exit the white zone while the yellow cubes remain within. Two versions of the task were created: one where the cubes were initialized with the same positions and orientations for each episode, and another where cube positions and orientations were randomized for each episode. The purpose of this is that if an agent successfully extracts a predetermined object with minimal interference with the surrounding objects, an *object isolation* can be executed by deploying the agent multiple times to extract cubes surrounding a target that should be isolated. For the agents trained for fixed formations, different formations were tried for different training runs.



(a) Initial setup for the version with the same fixed setup each episode.



(b) Initial setup for the version with scrambling each episode.

Figure 3.7 Initial setup for the extract with distractors task.

Success criterion. For the task to be deemed successful the green cube must exit the zone while all the yellow cubes remain within. The yellow cubes are allowed to relocate within the zone but not exit.

Reward shaping. The reward shaping function $R(s,a,s')$ for the *extraction with distractors* task consists of the following parts:

1. **Gripper to cube reward ($R_{\textbf{GripperToCube}}$)**: Same as Equation (3.14).
2. **Cube distance from center grasped ($R_{\textbf{CubeFromCenter}}$)**: Same as Equation (3.15).
3. **Distractor exit punishment ($R_{\textbf{DistractorExit}}$)**: A punishment for any distractor leaving the zone.

$$R_{\text{DistractorExit}}(s,a,s') = \begin{cases} -100 & \text{if a distractors leaves the zone} \\ 0 & \text{otherwise} \end{cases} \tag{3.21}$$

4. **Task complete reward ($R_{\textbf{TaskComplete}}$)**: A large reward for fulfilling the task success criteria.

$$R_{\text{TaskComplete}}(s,a,s') = \begin{cases} 1000 & \text{if the task is complete} \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

The total reward shaping function can be represented as the sum of these components:

$$\begin{aligned}R(s,a,s') =&R_{\text{GripperToCube}}(s,a,s')\\&+R_{\text{CubeFromCenter}}(s,a,s')\\&+R_{\text{DistractoreExit}}(s,a,s')\\&+R_{\text{TaskComplete}}(s,a,s')\end{aligned} \tag{3.23}$$

Reach and grasp a randomly positioned cube

The transition from teaching agents capable of completing the *extraction with distractors* task in a non-varying environment to teaching agents to complete the task in a randomized environment was considered a big jump. Therefore, two new simpler tasks were created consisting of reaching, and reaching and grasping a randomly positioned cube. The task design and reward functions were the same as the *grasp extraction* task, however, the success criterion differed. The reaching task was deemed successful when the robot gripper reached a predetermined proximity to the cube, and the reach and grasp task was deemed successful when the cube was grasped.

Cube extraction with vision-based observations

To explore how well the designed tasks could carry over to a more advanced state-space, the cube grasp extraction task was trained using the vision-based state-space consisting of RGB, depth, and segmentation mask. Two variations of the task were trained. One allowed only grasping strategies and one allowed the extraction of the cube by any means.

4

Results

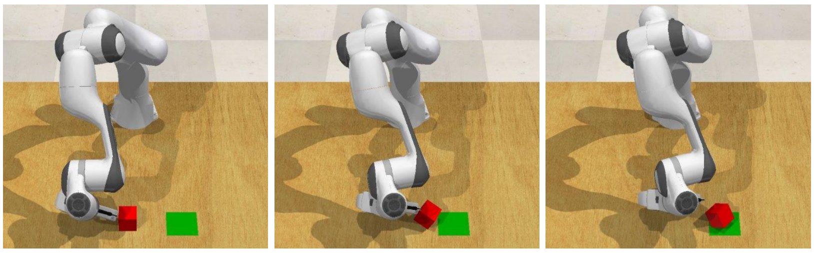
4.1 Task performance

Multiple simulations were done for multiple task variations. In this section, the tasks resulting in successful agents will be highlighted by displaying their performance through the mean of the accumulated reward per episode and the task completion success rates. Furthermore, images from renderings are used to illustrate the strategies of chosen policies.

Slide cube to target

The agent for the *slide cube to target* task learned different policies that completed the task, where the best-performing policy reached a success rate of around 80% to 90% as seen in Figure 4.7.

The successful policies consisted of the robot shoving the cube in the direction of the target, often resulting in the cube tumbling to its target. The methods were often unconventional and the robot would often use parts of its body other than the grippers as seen in Figure 4.1.



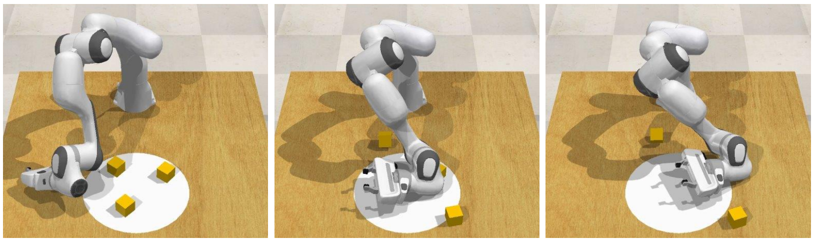
(a) Approach. **(b)** Push. **(c)** Cube entering the target.

Figure 4.1 Rendering showing a common strategy for the *slide cube to target* task, where the agent employs pushing which causes the cube to tumble into the target position.

Remove from zone

The agent for the *remove from zone* task learned different policies that completed the task with quite low success rates, where the best-performing policy reached a success rate of around 20% as seen in Figure 4.8.

The successful policies consisted of the robot arm sweeping across the white zone and knocking cubes out of it. The robot would make contact with the cubes with unconventional parts as seen in Figure 4.2, while moving fast resulting in the cubes tumbling as in the *slide cube to target* task.



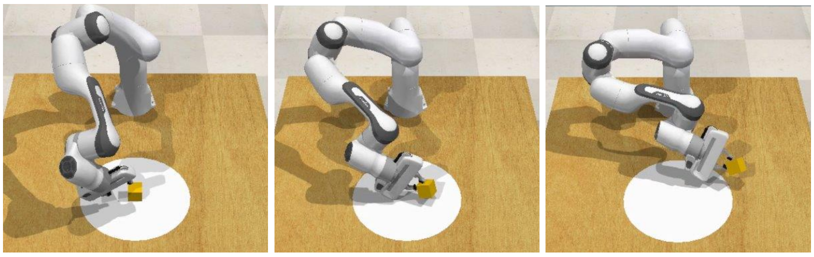
(a) Approach. **(b)** Contact. **(c)** Task complete.

Figure 4.2 Common strategy for the *remove from zone* task, where an agent employs a strategy consisting of a sweeping motion.

Grasp extraction

The agent for the *grasp extraction* task learned different policies that completed the task with varying success rates, where the best-performing policy reached a success rate of around 85% as seen in Figure 4.9.

Due to task design, only policies with grasp strategies were learned. Different policies exhibited different approaches to grasping the cube where some grasped at the very edge of the cube as seen in Figure 4.3, and others exhibited impossible grasping strategies as in Figure 3.1.



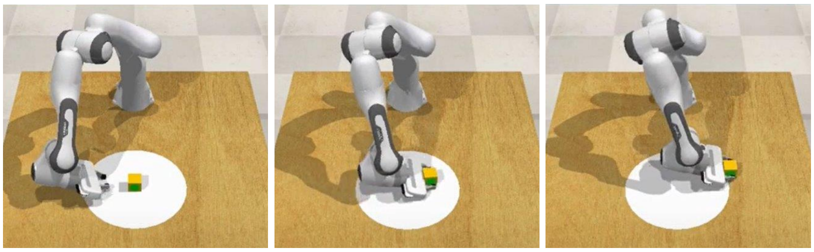
(a) Approach. **(b)** Grasp. **(c)** Extract.

Figure 4.3 Strategy for an agent completing the *grasp extraction task*.

Extraction with improved grasping

For the task dedicated to teaching improved grasping positions the best-performing agent reached a success rate of 90% for both variations as seen in Figure 4.10 and Figure 4.11. Agents for both cube area varieties demonstrated policies striving to grasp the cubes in the grasping areas, effectively reducing the number of policies practicing impossible or difficult grasping strategies as in Figure 3.1.

Two main approaches to reach the grasping zones were noticed. One was the robot arm adapting its approach to be able to grasp within the grasping areas directly as seen in Figure 4.4, and the other was nudging the cube slightly to align the grasping areas with the gripper.



(a) Approach. **(b)** Grasp. **(c)** Extract.

Figure 4.4 Strategy from an agent performing the *extraction with improved grasping* task.

Extraction with distractors

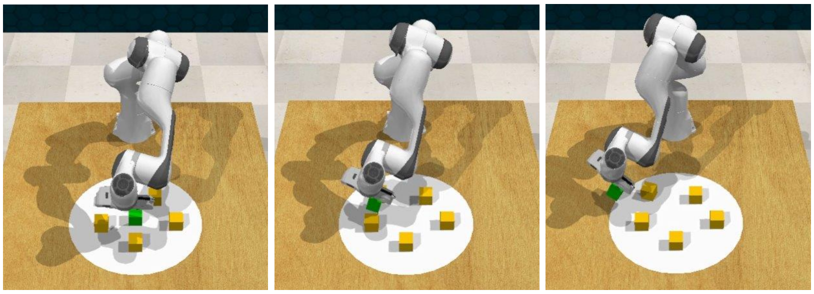
For the *extraction with distractors* task for a non-varying environment multiple fixed formations were tried. In this section two formations, A seen in Figure 4.5 and B seen in Figure 4.6, will be exhibited.

The agent for formation A learned different policies that completed the task with varying success rates, where the best-performing policy reached a success rate of around 60% to 70% as seen in Figure 4.12. However, it was common that some runs failed to learn a good policy such as A, G and I in Figure 4.12.

The successful policies consisted mostly of navigating the green target cube between the yellow distractor cubes through grasping only as in Figure 4.5 or a combination of grasping and pushing.

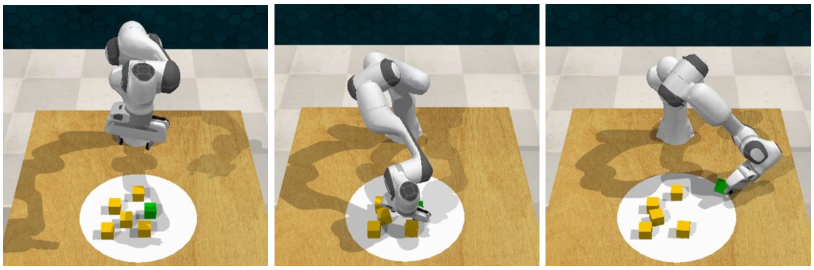
The best-performing policy for the agent trained for formation B achieved a success rate of around 90% as seen in Figure 4.13. The successful policies consisted of the robot navigating down to the cube and pulling it out in the opposing direction of distractor cubes. An example of such a policy is shown in Figure 4.6.

For the corresponding task with randomized cube positioning in each episode, no successful policies were achieved.



(a) Approach. **(b)** Grasp. **(c)** Extract.

Figure 4.5 A strategy for an agent completing the *extract with distractors* task for formation A, where the agent navigates the cube out between the distractors.



(a) Approach. **(b)** Grasp. **(c)** Extract.

Figure 4.6 A strategy for an agent completing the *extract with distractors* task for formation B, where the agent navigates the cube out of the zone while avoiding the distractors.

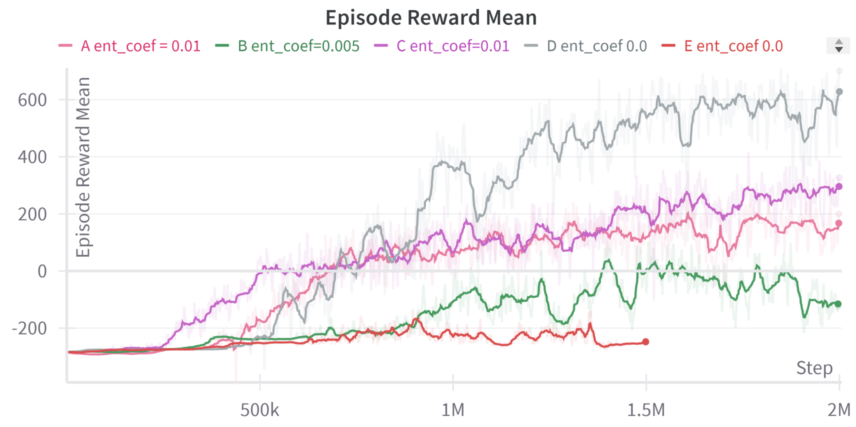
Reach and grasp a randomly positioned cube

The agent for the *reach a randomly positioned cube* task learned policies that completed the task with a success rate of around 65% as seen in Figure 4.14, and the agent for the *grasp a randomly positioned cube* task learned policies that completed the task with a success rate of around 30% as seen in Figure 4.15.

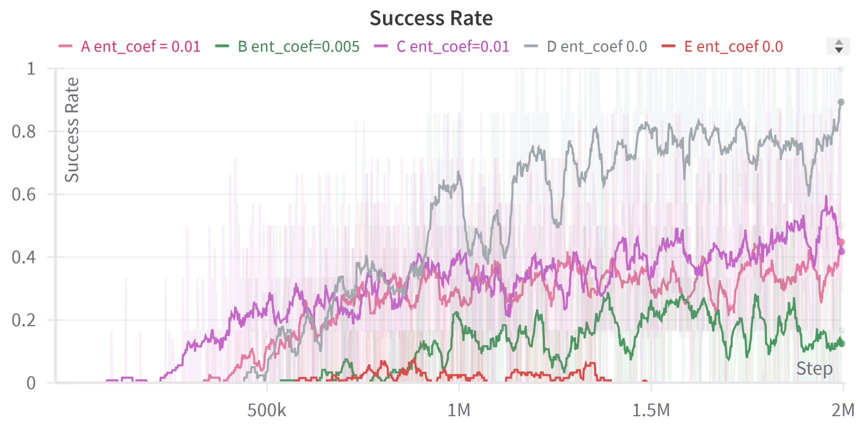
The successful policies consisted of navigating down to the area of the cube and then moving around in the general direction of the cube before reaching or grasping it.

Cube extraction with vision-based observations

For the vison-based agent, fewer and shorter runs were done as a result of the longer training time. The agent for the extraction through grasping failed to learn as seen in Figure 4.16, and the agent for extracting the cube by any means learned a policy that achieved a success rate of around 30% as seen in Figure 4.17, where the policy consisted of pushing the cube out of the zone.

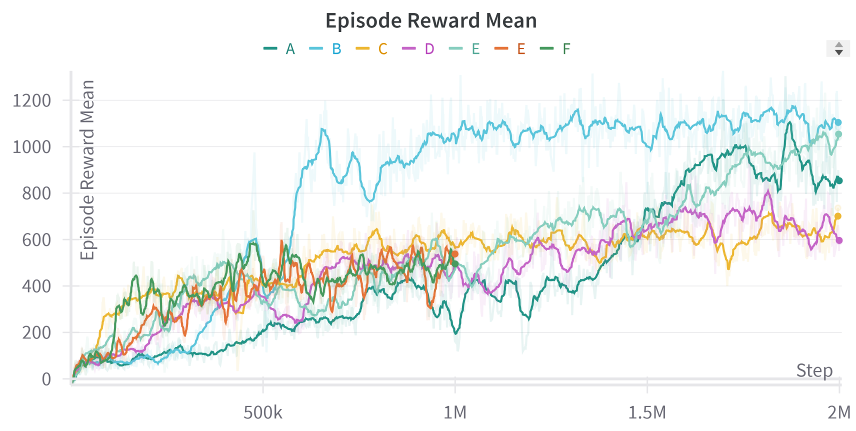


(a)

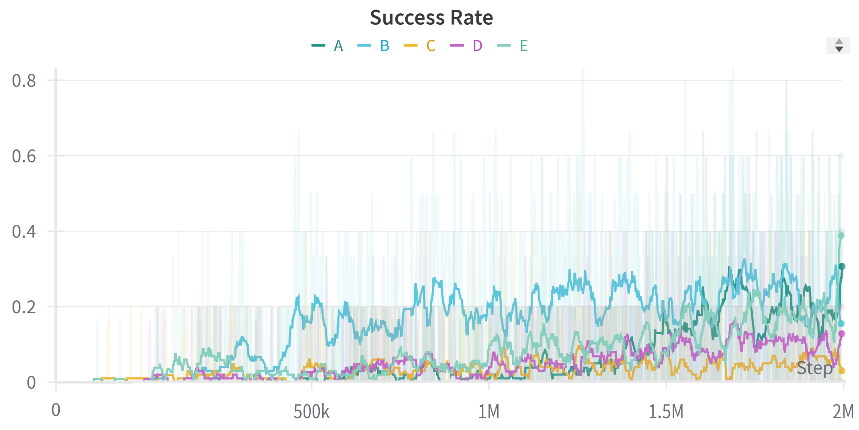


(b)

Figure 4.7 Reinforcement learning graphs for the *slide cube to target* task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 30 steps and the policy updated every 180 steps. Lines E and D used an entropy coefficient of 0, A and C 0.01, and B 0.005.

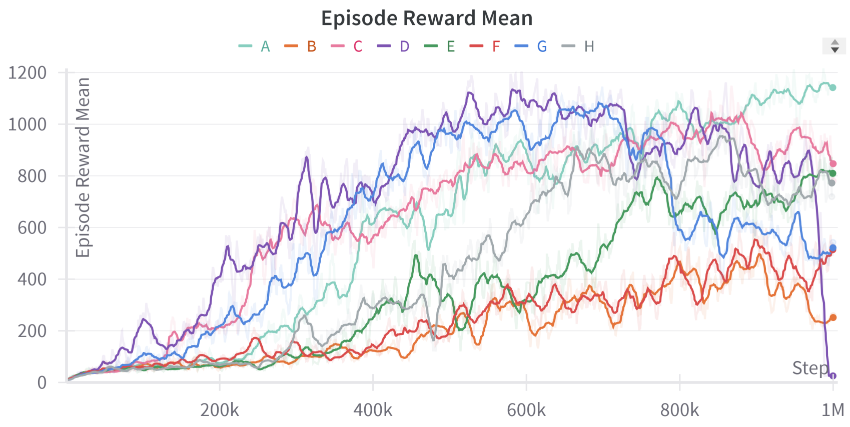


(a)

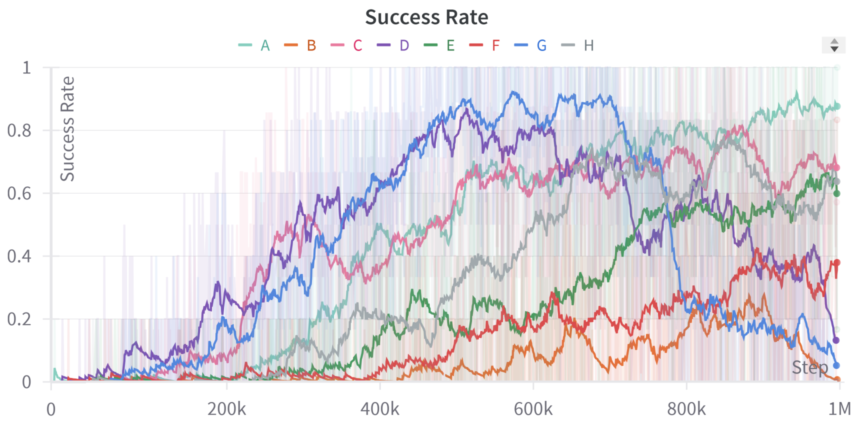


(b)

Figure 4.8 Reinforcement learning graphs for the *remove from zone* task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 40 steps and the policy was updated every 200 steps. All plots used an entropy coefficient of 0.

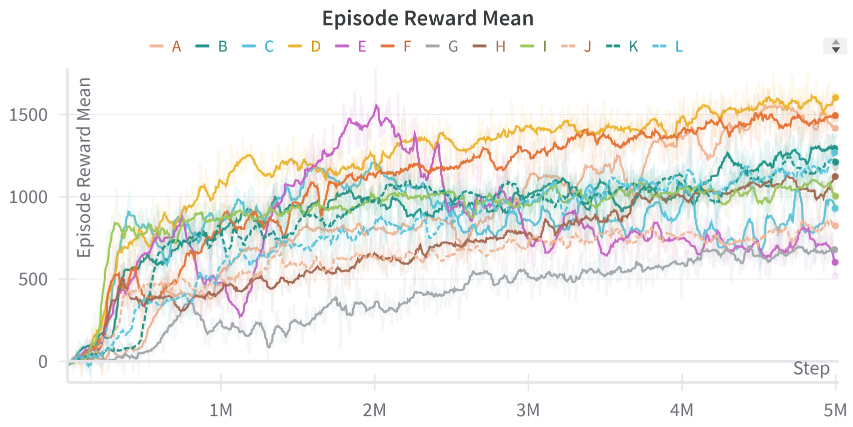


(a)

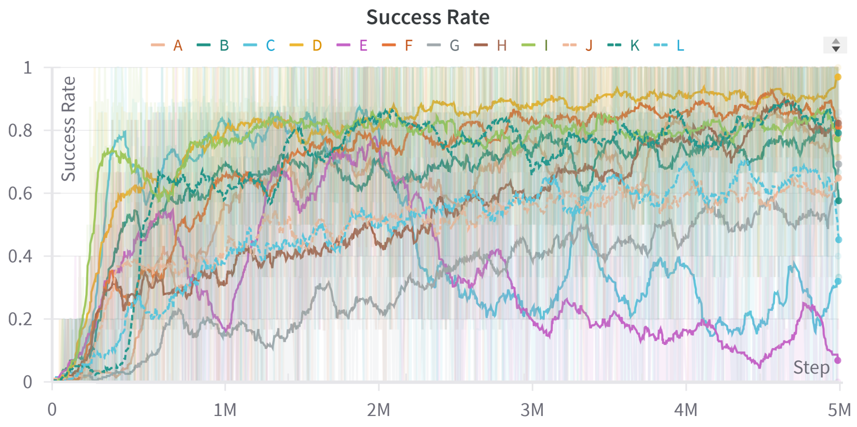


(b)

Figure 4.9 Reinforcement learning graphs for the *grasp extraction* task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 50 steps and the policy updated every 250 steps. Lines A, B, C, D, and E used an entropy coefficient of 0, and, F, G, and H 0.01.

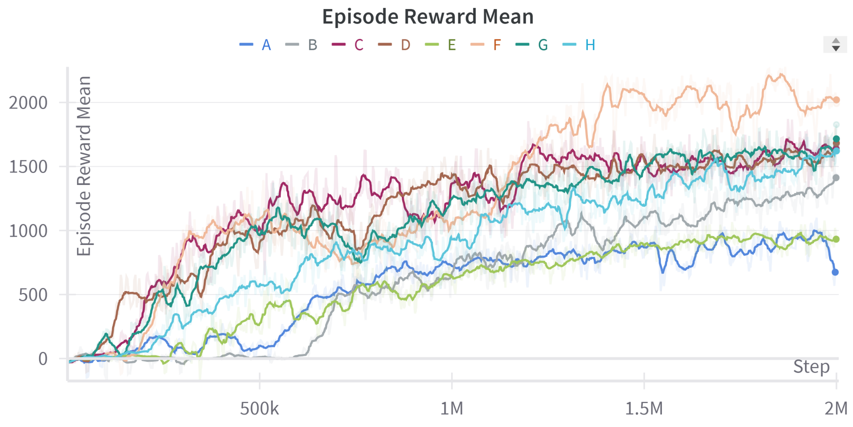


(a)

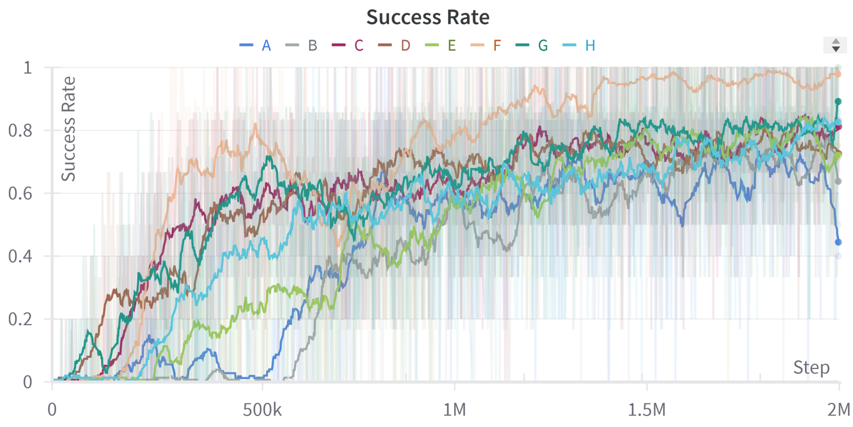


(b)

Figure 4.10 Reinforcement learning graphs for the *extraction with improved grasping* task with one pair of grasping areas where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 60 steps and the policy updated every 300 steps. Lines A, B, C, D, E, and F used an entropy coefficient of 0, and G, H, I, J, K, and L 0.01.

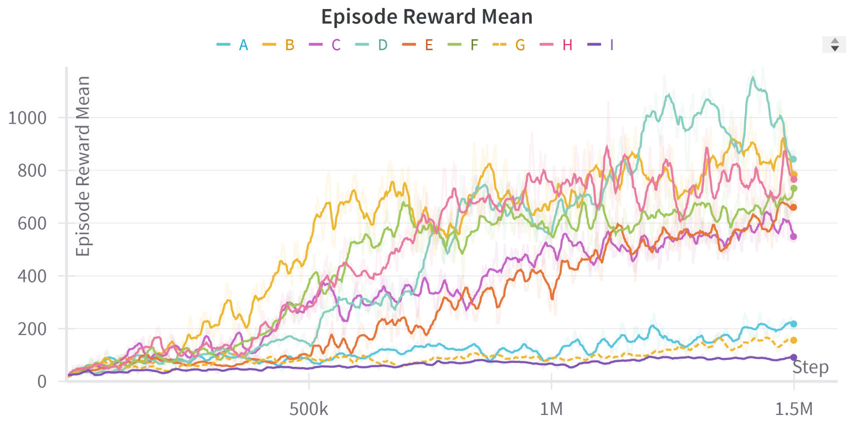


(a)

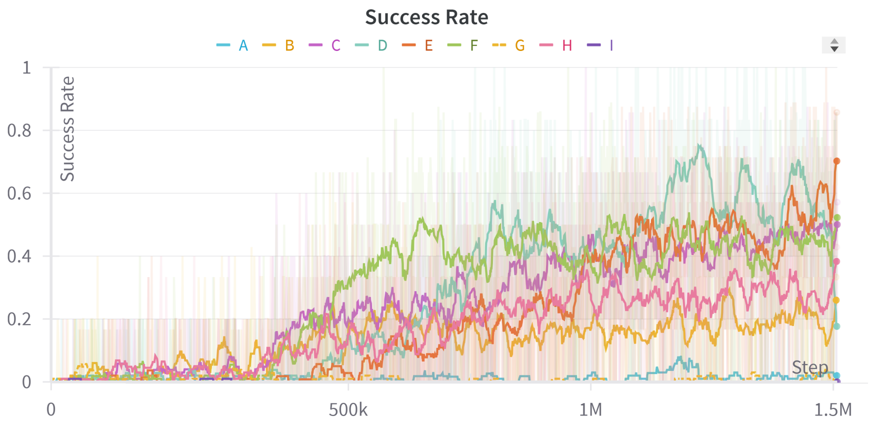


(b)

Figure 4.11 Reinforcement learning graphs for the *extraction with improved grasping* task with two pairs of grasping areas where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 60 steps and the policy updated every 300 steps. All lines used an entropy coefficient of 0.

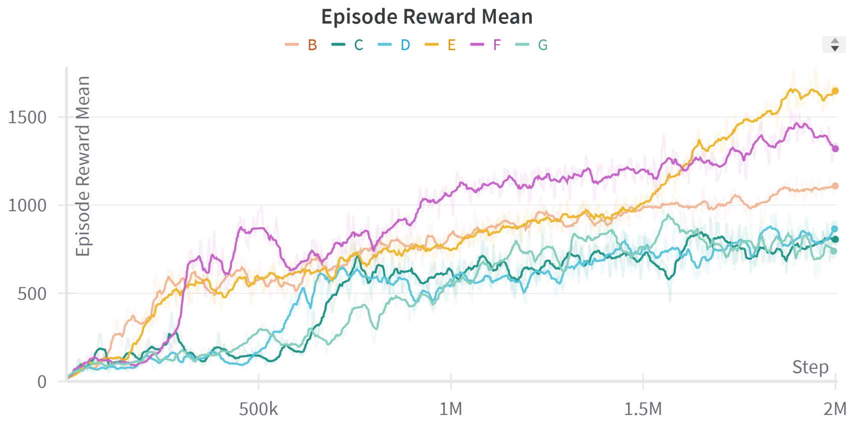


(a)

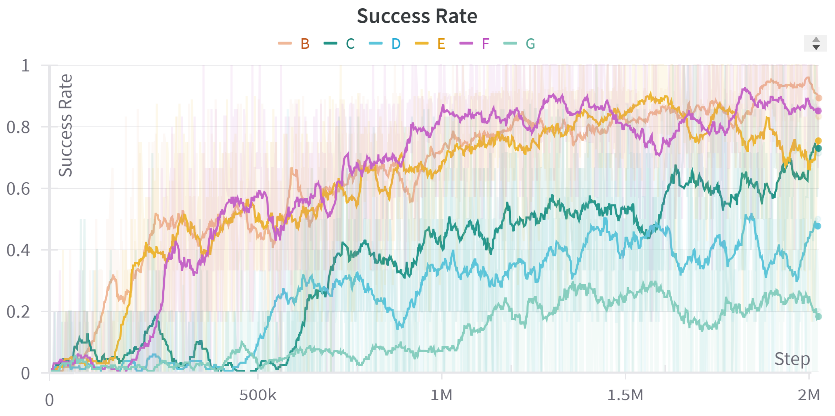


(b)

Figure 4.12 Reinforcement learning graphs for the *extraction with distractors* for formation A task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 80 steps and the policy updated every 400 steps. Lines A, B, C, D, E, F, and G used an entropy coefficient of 0, and H, and I used 0.01.

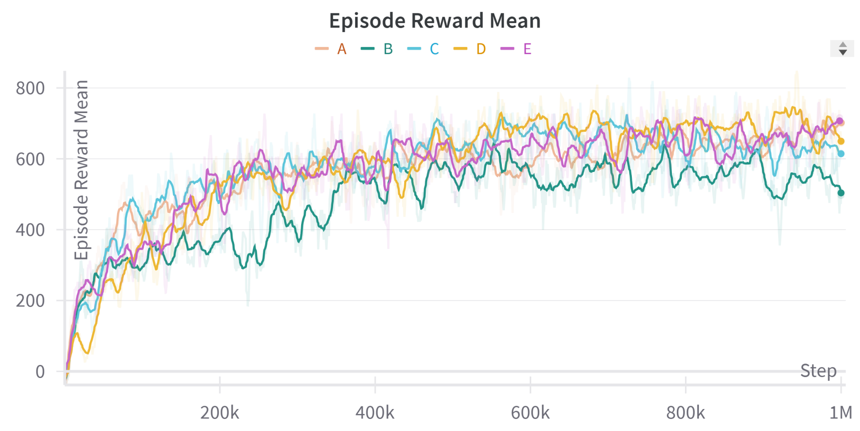


(a)

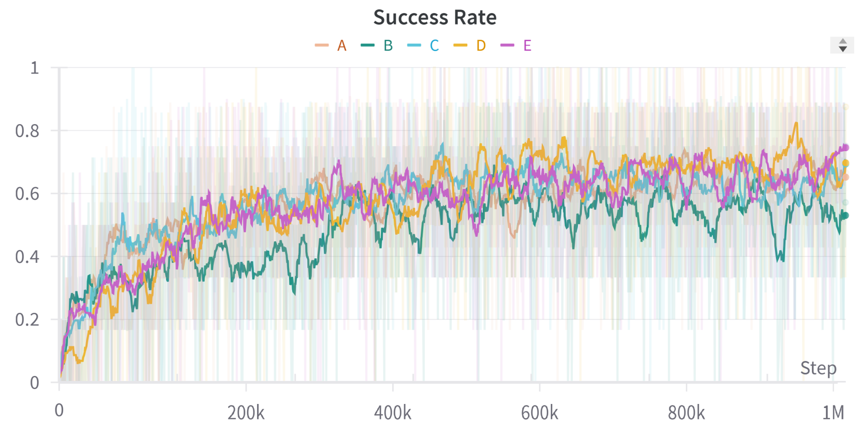


(b)

Figure 4.13 Reinforcement learning graphs for the *extraction with distractors* for formation B task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 80 steps and the policy updated every 400 steps. All lines used an entropy coefficient of 0.

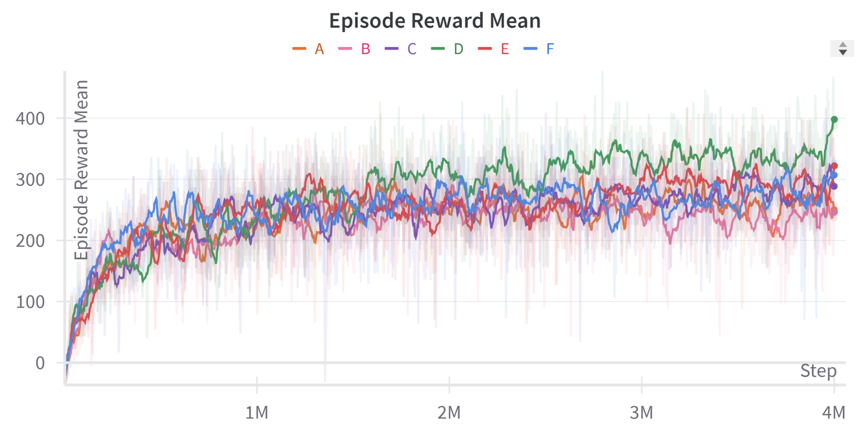


(a)

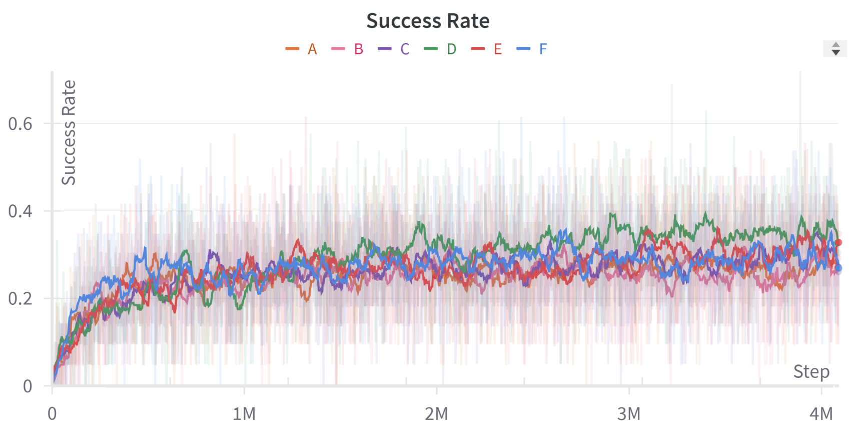


(b)

Figure 4.14 Reinforcement learning graphs for the *reach a randomly positioned cube* task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 60 steps and the policy updated every 300 steps. All lines used an entropy coefficient of 0.



(a)



(b)

Figure 4.15 Reinforcement learning graphs for the *grasp a randomly positioned cube* task where a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 60 steps and the policy updated every 1200 steps. All lines used an entropy coefficient of 0.

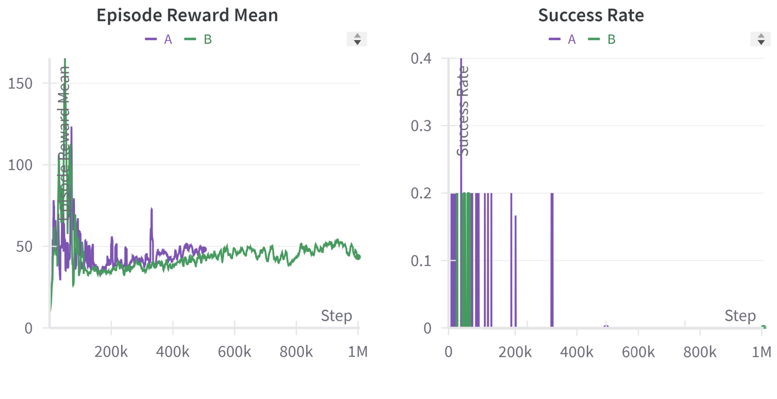


Figure 4.16 Reinforcement learning graphs for a task using a vision-based state. The task consisted of a cube extraction task where only grasping strategies for extraction were allowed. a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 50 steps and the policy was updated every 250 steps. No entropy bonus was used.

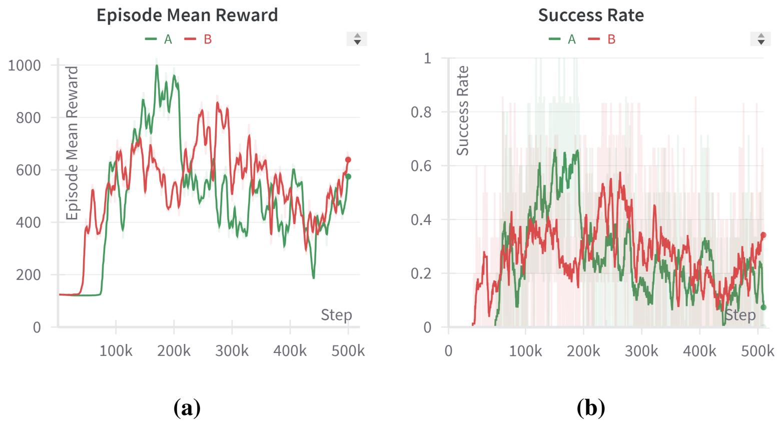


Figure 4.17 Reinforcement learning graphs for a task using a vision-based state. The task consisted of a cube extraction task where any strategy for extraction was allowed. a) shows the mean of the accumulated reward for each episode on the Y-axis and the total step count on the X-axis, and b) shows the success rate of each policy update on the Y-axis and the step count on the X-axis. Each episode had a maximum length of 50 steps and the policy updated every 250 steps. No entropy bonus was used.

4.2 Hardware and training time

For the training process for the agents utilizing the low-dimensional observations, the policy and critic network both used two intermediate hidden layers, each containing 64 neurons. The input to both networks consisted of the observations in Table 3.1. The training was done on three different machines, however, most runs were done on a computer with an Intel Core i9 i9-13900KF processor [Intel Corporation, 2022] and an NVIDIA GeForce RTX 4090 graphics card [NVIDIA Corporation, 2022]. The agent training process utilized both the CPU (Central processing unit) and GPU (Graphics processing unit), and training times for the task would vary slightly depending on the task itself, policies learned and the number of runs done simultaneously. For the mentioned computer, a general observation was that the *extraction with improved grasping* task in Figure 4.11 would need around twelve hours of training time, and the *extract with distractors task* in Figure 4.13 would need around twenty-four hours, both for two million total steps per run. For both cases, at least five runs were conducted simultaneously and the training times mentioned correspond to two of the tasks that required the longest training duration with the low-dimensional observations.

For the attempts using vision-based state-space the same computer and network architecture was used, however, now the input consists of the vision-based observations containing RGB, depth, and segmentation masks. The training for the run of one million steps for the vision-based task in Figure 4.16 required two days and twenty-three hours to train.

5

Exploring the sim2real gap

This section will give a general overview of an attempted sim2real implementation and its outcome. The purpose of the attempt was to gain insight into what challenges exist when applying a reinforcement learning agent to a real robot process. Furthermore, the attempt aims to explore what strategies can lead to a successful sim2real implementation.

5.1 Approach

The simulated robot task environment was replicated as seen in Figure 5.1. To bring an agent from simulation to reality, a real robot environment should take the place of the simulated robot environment. This means that the observations from the simulations should be replaced by information from the real robot, and the robot should respond to the actions sent by the agent.

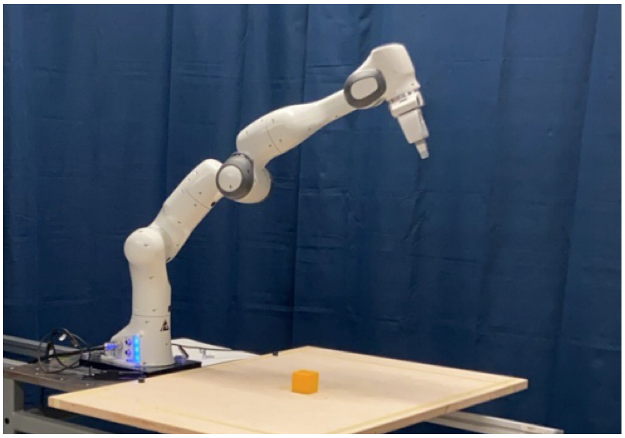


Figure 5.1 An image of the real version of the Franka Emika Panda robot [Haddadin et al., 2022] setup to replicate the experiments done in the simulated robot environment.

A successful agent for the *extract with improved grasping* task was chosen as a candidate for the sim2real process. The agent was selected since it learned policies achieving a high success rate in the simulation, and from rendering its behavior it also showed reliable strategies to solve its task.

Observations

The observations provided to the agent consisted of all observations in Table 3.1. Observations regarding the robot were extracted and sent to the agent, and task-specific observations were entered manually based on the simulator. The task-specific information, in this case, consisted of the position of the cube. The simplification of adding the cube positioning manually was done to avoid the need to implement an additional system to track the cube. Furthermore, the cube would remain stationary for the initial parts of the robot task, which consisted of reaching and grasping the cube.

Actions

The joint velocities commanded by the agent were now based on the observations from the real robot. The desired joint velocities were scaled to the real robot capabilities and were sent to a controller for the robot joint velocities in intervals close to the step time in the simulator.

5.2 Outcome

The agent was successful in controlling the robot arm, however, the performance of the agent did not resemble the simulated environment. The robot arm would move and the gripper would open and close. Despite this, the actions did not show any signs of working toward completing the task that the agent was trained for.

When comparing the observations from the robot and the simulated environment, differences were apparent. Furthermore, since no countermeasures were added for ignoring actions that led to joint constraints, some runs would end in the agent sending action leading the robot to reach its joint limits. Different implementations of controlling the joint velocities were used, however, none were successful in reproducing the performance achieved in the simulation.

6 Discussion and conclusion

The following section discusses the learned policies for the different agents and highlights some key findings regarding the policy performance and the agent's abilities to learn a given task. Suggestions for future improvements and a discussion of the sim2real gap will also be addressed, followed by a conclusion tied to the research questions.

6.1 Agent evaluation

Undesired versus desired policies

Throughout the task development and agent training, it became apparent that the reward shaping had a large impact on how the tasks were completed. An agent could have developed a policy effectively completing a task while displaying undesired approaches and strategies. For example, the agents for the *remove from zone* and *slide cube to target* tasks both displayed strategies that could result in the cubes being manipulated in an uncontrolled manner, specifically highlighted by the agent in Figure 4.2. Developing such strategies can be seen as unuseful or difficult to implement in a real-life application. To develop more robust cube manipulation approaches, grasping strategies were rewarded, leading to agents developing policies utilizing grasping as the agent in Figure 4.3.

However, other traits were developed for the grasping approach that could be seen as undesired. For example, in the first grasping task *grasp extraction* the problem of impossible grasping positions, as illustrated in Figure 3.1, became apparent. Throughout the training and task development, the ability to bring an agent from simulation to real life was always in mind. Therefore tasks to encourage better grasping positions were created, resulting in overall better grasping approaches for the developed policies as exemplified by the policy shown in Figure 4.4. The improved grasping tasks also achieved a good success rate as seen in Figure 4.11.

However, a strategy occurred that could be seen as difficult to implement for a real robot. This strategy consisted of the robot nudging the cube into a different position before grasping in the rewarded grasping area. This behavior can be seen as an unfavorable approach to achieving a precise grasping position since it relies on an accurate simulation of the cube dynamics. Pushing the cube in the simulation versus a real cube can lead to two different positions since the simulated environment might not line up with the real-life physical properties of the cube, such as friction and mass. The policies involving directly pursuing a good grasping position were therefore considered superior.

The introduction of the cube grasping areas proved successful in deterring impossible grasping strategies and influencing the robot's grasp positioning. The ability to influence an agent's grasp positioning could have further benefits. An example could be when dealing with fragile or irregularly shaped objects requiring specific grasping approaches.

For the *extraction with distractors* task involving extracting a cube surrounded by distractors, the successful policies consisted of removing the cube by gripping it and extracting it by navigating in between or away from the distracting cubes as shown in Figure 4.5 and Figure 4.6. From an outside perspective, gripping the cube and lifting it straight up would seem like a simpler and more effective approach. The same goes for the other grasp extraction tasks where policies involving lifting were uncommon. The reason for this behavior could come from the reward shaping. When gripping the cube, reward is given for moving it out from the center of the zone. If the robot moves the cube close to the work surface it can let it go, re-grasp, and continue moving. However, if the robot would move the cube vertically and let it go, the cube would drop, making the task more difficult to resume. This could ultimately lead to policies involving lifting the cube further from the work surface less common. To combat this, reward shaping could be added to vertical movement.

Randomization

With a randomized version of the *extraction with distractors* task, the agents failed to learn and complete the task. The simpler task of reaching, and reaching and grasping a cube did succeed in learning as seen in Figure 4.14 and Figure 4.15, however, total training time for simpler tasks in a randomized environment was significantly larger compared to the advanced task in a fixed environment.

Vision-based observations

For the vision-based agents, it became apparent that there is a potential for transferring the tasks and reward functions designed for a low-dimensional state-space to a vision-based state-space since one of the agents managed to learn a policy with a success rate of 30% as seen in Figure 4.17. However, more experiments are necessary to improve on the vision-based agents. As seen in Figure 4.16, the vision-based grasping task the agent starts to learn, however, after around a hundred thousand steps failure occurs. To determine what causes this failure to learn and sudden drop would require further runs.

6.2 Algorithm parameters

The purpose of the clipping in the surrogate objective function, Equation (2.6) in PPO, is to make sure the policy does not change too much at the time, ultimately leading to failure and the policy being worse. As mentioned previously and as seen in Figure 4.9 for runs D and G, some runs showed signs of the policy veering off too far leading to failure. To avoid this the clipping range could be altered for tasks with this problem, reducing the risk of large policy updates.

For some runs the entropy bonus was tested with the scaling entropy coefficient of 0.01. It was observed that the agent's ability to learn was influenced by the inclusion of the entropy bonus. As seen in Figure 4.7, Figure 4.10, and Figure 4.12 the agent using the entropy bonus would consistently learn and the spread in performance would vary less compared to agents without the bonus. This also resulted in that the best-performing agents were the ones without the entropy bonus. A reason for this behavior could be that the entropy bonus encourages exploration over exploitation, resulting in consistency in finding a successful policy for the use of an entropy bonus. In contrast, no entropy bonus allows exploitation if a successful policy is found early, which can lead to the optimal policy being found faster.

Furthermore, this thesis focused on employing PPO to solve robotics tasks, however, there are multiple other reinforcement learning algorithms and approaches to combining machine learning with robotics. For this thesis PPO proved to be a viable algorithm for the designed tasks, however, testing other reinforcement learning algorithms could provide different results and insights into how to work toward completing an *object isolation* task.

6.3 Sim2real

The agent's diverging behavior when deployed in a real robot process can be caused by multiple different factors. Regarding the observations given to the agent, differences were found when comparing the real process and the simulation. This would ultimately lead to the agent working with observations that could seem foreign, ultimately impacting its decision making.

Another factor that influences the agent's performance in a real process is whether the actions in a real process would result in the same outcome as the actions in a simulation. For an agent sending joint velocities as references, there is no guarantee that the sent joint velocities for a given time would result in the same pose for a real robot and a simulation. This is highly dependent on the controller of the robot's joints and experimenting with the controller and the scaling of the actions sent by the agent are necessary steps to work toward a successful sim2real implementation.

The problem of accurately communicating an agent's action to a robot can be seen as specifically apparent when using actions directly controlling the robot joints.

If the approach of utilizing an agent together with a planner was selected instead, the end-effector position could be directly sent from the agent to a planner executing the robot movement. This would eliminate the problem of actions sent by the agent resulting in different poses when comparing a real robot with a simulation.

Ideally, the environment that the agent has been trained in should closely resemble the real environment in which it is to be deployed. To achieve this data could be collected from the robot, and thereafter the simulation could be modified to more closely resemble the real robot in which the agent should be implemented. Furthermore, the training process of the agent could continue when applied to a real robot. This enables the policies developed to adapt and be refined for a real process.

6.4 Conclusion and future work

Agents capable of primitive robot actions such as reaching, avoiding, pushing, and grasping were achieved. These actions were used to complete more complex tasks such as grasp extractions, where a method to improve grasping approaches was explored and shown to be successful. Furthermore, an agent capable of extracting a target cube without removing the surrounding distractor blocks was achieved for a fixed environment without episode variations. For tasks with an environment that scrambles each episode, agents with the ability to reach and grasp randomly placed cubes were achieved.

This ultimately satisfies *research question 1*, furthermore, some tasks combine multiple actions into one task. For example, the *extract with distractors* task combines reaching, grasping, and avoiding to complete the task. This works towards satisfying *research question 2*, where combining and utilizing the action primitives was an objective. However, the objective of training an agent capable of completing an object isolation task was not reached, and therefore *research question 2* was not fully satisfied.

The selected approach to complete the objective was to refine the agent capable of extracting the target cube without affecting the surrounding cubes in such a way that it can complete the task for an environment that scrambles the cube positions each episode. With this approach, the agent could be deployed for cubes surrounding the target cube, to complete an object isolation.

A suggestion for future work, to achieve an agent capable of extracting randomly positioned cubes, is to pursue further testing and training for the tasks involving reaching and grasping for randomized positions. Once a satisfactory success rate has been reached, the extraction part can be added to the task and thereafter the distractors can be reintroduced. This ultimately repeats the process for the successful task achieved for fixed positions.

In addition to this, further testing could explore tuning the different algorithm-specific parameters, scaling between the shaping rewards, and trying longer training sessions. Furthermore, the tasks developed for this thesis were focused on extraction. To further develop the agents releasing the cubes in a determined area should also be considered.

Bibliography

Achiam, J. (2018). “Spinning Up in Deep Reinforcement Learning”. Accessed: 2024-04-15. URL: `https://spinningup.openai.com/en/latest/`.

Akkaya, I., M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. (2019). “Solving Rubik’s cube with a robot hand”. *arXiv preprint arXiv:1910.07113*.

Arents, J. and M. Greitans (2022). “Smart industrial robot control trends, challenges and opportunities within manufacturing”. *Applied Sciences* **12**:2, p. 937.

Bajd, T., M. Mihelj, and M. Munih (2013). *Introduction to Robotics [E-book]*. Springer Netherlands, Dordrecht. ISBN: 9789400761018.

Berscheid, L., P. Meißner, and T. Kröger (2019). “Robot learning of shifting objects for grasping in cluttered environments”. In: *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, pp. 612–618.

Boston Dynamics (2024a). *Atlas: the world’s most dynamic humanoid robot*. Accessed: 2024-06-18. URL: `https://www.bostondynamics.com/atlas`.

Boston Dynamics (2024b). *Spot: the agile mobile robot*. Accessed: 2024-06-18. URL: `https://www.bostondynamics.com/spot`.

Haddadin, S., S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin (2022). “The Franka Emika Robot: A reference platform for robotics research and education”. *IEEE Robotics Automation Magazine* **29**:2, pp. 46–64. DOI: `10.1109/MRA.2021.3138382`.

Intel Corporation (2022). *Intel® Core™ i9-13900KF Processor*. `https://www.intel.com/content/www/us/en/products/sku/230497/intel-core-i913900kf-processor-36m-cache-up-to-5-80-ghz/specifications.html`. Accessed: 2024-06-23.

James, S., Z. Ma, D. Rovick Arrojo, and A. J. Davison (2020). “RLBench: The robot learning benchmark & learning environment”. *IEEE Robotics and Automation Letters*.

Mehlig, B. (2021). *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press. ISBN: 9781108494939. DOI: `10.1017/9781108860604`. URL: `http://dx.doi.org/10.1017/9781108860604`.

NVIDIA Corporation (2022). *GeForce RTX 4090 Graphics Card*. `https://www.nvidia.com/sv-se/geforce/graphics-cards/40-series/rtx-4090/`. Accessed: 2024-06-23.

Otterlo, M. van and M. Wiering (2012). "Reinforcement Learning and Markov Decision Processes". In: Wiering, M. et al. (Eds.). *Reinforcement Learning: State-of-the-Art*. Springer Berlin Heidelberg, pp. 3–42. ISBN: 978-3-642-27645-3. DOI: `10.1007/978-3-642-27645-3_1`. URL: `https://doi.org/10.1007/978-3-642-27645-3_1`.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019). "PyTorch: An imperative style, high-performance deep learning library". *Advances in neural information processing systems* **32**.

Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations". *Journal of Machine Learning Research* **22**:268, pp. 1–8. URL: `http://jmlr.org/papers/v22/20-1364.html`.

Sanderson, G. (2017). *But what is a neural network? | Chapter 1, Deep learning*. URL: `https://www.youtube.com/watch?v=aircAruvnKk`. Accessed: 2024-05-10.

Sarantopoulos, I., M. Kiatos, Z. Doulgeri, and S. Malassiotis (2020). "Split deep Q-learning for robust object singulation". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6225–6231.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). *Proximal policy optimization algorithms*. arXiv: `1707.06347 [cs.LG]`.

Shahid, A. A., L. Roveda, D. Piga, and F. Braghin (2020). "Learning continuous control actions for robotic grasping with reinforcement learning". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, pp. 4066–4072.

Silver, D. (2015). *Lectures on reinforcement learning*. URL: `https://www.davidsilver.uk/teaching/`. Accessed: 2024-03-20.

Simonini, T. and O. Sanseviero (2023). *The hugging face deep reinforcement learning class*. `https://github.com/huggingface/deep-rl-class`. Accessed: 2024-05-17.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement Learning: An Introduction*. Second. Accessed: 2024-04-10. The MIT Press. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

Wu, P., A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg (2023). “Daydreamer: world models for physical robot learning”. In: *Conference on Robot Learning*. PMLR, pp. 2226–2240.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* July 2024 *Document Number* TFRT-6255
Author(s) Teodor Åstrand	*Supervisor* Yiannis Karayiannidis, Dept. of Automatic Control, Lund University, Sweden Björn Olofsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Robot Reinforcement Learning for Object Isolation

Abstract

This thesis employs deep reinforcement learning, a branch of machine learning, to carry out robotic tasks. The objective centers around teaching an agent controlling a seven-axis robot arm with a gripper tool, to complete an object isolation task. For this task, a robot manipulates a cluttered environment in such a way that a predetermined target object becomes isolated. Sub-tasks were developed to explore simpler robot tasks to evolve and combine into more complex tasks, where the goal was the object isolation task. Agent training took place in a simulated robot learning environment with the use of primarily a coordinate-based low dimensional statespace, where reward shaping was the primary tool to teach a given task.

The reinforcement learning algorithm Proximal policy optimization (PPO) implemented with a neural network architecture was used to train agents for the robotics tasks and the robot arm's joint velocities were used as the action-space for the agents. Multiple experiments were conducted for agents practicing different tasks and their performance was evaluated by measuring their task completion rate and rendering their behavior among others. Agents developed policies capable of different forms of cube manipulation and performing cube extraction tasks. Multiple different policies for completing robot tasks were learned, and their strategies were evaluated and discussed.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316	*ISBN*

Language English	*Number of pages* 1-58	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/