Reinforcement learning framework for window hardware installation

by

Tzu-Hao Huang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Construction Engineering and management

Department of Civil and Environmental Engineering

University of Alberta

ABSTRACT

The repetitiveness and precision of manufacturing tasks has increased the need for robots in the automation of the manufacturing industry; however, the complex and varied nature of manufacturing production lines poses challenges in terms of applying the rule-based automation approach. This has contributed to the trend of employing artificial intelligence-driven robotic systems. Previous applications relied on accomplishing a single automated task with its specifically designed AI model, and thus failed to provide a scalable AI solution that could accomplish a variety of tasks. In the present case study, window hardware installation was simulated with a reinforcement learning solution using the soft actor critic algorithm to improve model learning efficiency and scalability. Agent training techniques, such as rewarding shaping and curriculum training, were introduced into the model's learning configuration. The proposed curriculum guided reinforcement learning structure provides the training agent a gradual and effective way to comprehend assigned tasks. The proposed approach further increases the potential for artificial intelligence-driven robotic hardware installation systems to be more sensitive and flexible to environment change and target hardware variations with the implementation of training guidelines. The model's performance is demonstrated by refined self-driven motion planning and hardware sensitive decision-making. This application indicates that a robust and scalable artificial intelligence model can be realized by thoughtful agent incentives and learning pathways, and the case study demonstrates the framework required to facilitate such a possibility.

ACKNOWLEDGEMENTS

The thesis could not be completed without the help from so many people on my MSc journey. In the beginning, I would like to express my sincere gratitude to my supervisor, Dr. Mohamed Al-Hussein, for giving me the opportunity to enter his research group, for his mentorship, continuous support, encouragement, guidance, and inspiration throughout my studies.

Next, I would like to give my thanks to my co-supervisor, Dr. Shih-Chung Jessy Kang. Without him and his team, I could not have determined my research direction and learned competitive knowledge of robotic manipulators.

I am also deeply grateful to my seniors, Ting-han Wei, Jin-Bo Huang, and Simon Steinmann. Without their support, I could not have finished this research which was originally unfamiliar to me. They provided me with professional experience and encouragement for taking challenges.

Moreover, I would like to express my appreciation to my lab members, Nan Chang and Mengni Wang, for providing window-related knowledge and support for the BIM model and related equations.

Also, I am strongly grateful to my girlfriend, Jacqueline Tung. Thank you for having my back, for enduring the pressure of research progress, and for encouraging me to be confident in my work and effort.

Meanwhile, I would like to give my deep thanks to my roommate, Anson Wang, for the writing support and useful opinion of my thesis. Special thanks to his two adorable cats for accompanying me.

Finally, I would like to express my deepest appreciation to my parents for their sacrifice, love, and support. Sorry for the absence of nearly two years. I am grateful and blessed to have them as my parents.

Table of content

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial intelligence
BIM	Building information modelling
CNC	Computerized numerical control
CGRLS	Curriculum guided reinforcement learning structure
DOF	Degrees of freedom
DRL	Deep reinforcement learning
FK	Forward kinematics
HT	Hinge track
IK	Inverse kinematics
LIN	Linear motion
LR	Learning rate
OP	Operator
PTP	Point-to-point motion
PVC	Polyvinyl chloride
RP	Ramp
SAC	Soft actor critic algorithm
UR5e-1	The number one universal robot with vacuum gripper
UR5e-2	The number two universal robot with screwdriver system

1 INTRODUCTION

Robotics play a vital role in the field of modern manufacturing. The use of robotics in the off-site construction industry is a natural fit because of improved safety and higher operating efficiency compared to manual operation. Currently, the aim of Industry 4.0 is to build an intelligent semi-automated, or even fully-automated, production line. With the work force aging at an increasing rate and given the overall transformation of the market, robots are becoming more and more popular. The major challenge of the automated production line is to be adaptive, flexible, robust, and economically efficient (Lei, 2017). In the context of window and door manufacturing, a common usage of robotics includes a variety of CNC machines to accomplish material processing, welding, and accessories conveying. Among the more commonly used robots, the robotic manipulator was one of the first robots used in the context of manufacturing.

Many manufacturing tasks require coordination between two or more robotic arms. However, such coordination is complicated and poses challenges in real-world application, which explains why industrial robotic arms are not yet widely implemented. Due to the potential for increased precision and efficiency in industrial applications, the development of end-effectors and the adoption of industrial manipulators for combined operation are both becoming a growing trend (Evjemo, 2020).

1.1 Automated assembly in window manufacturing

In the window and door manufacturing industry specifically, the usage of industrial manipulators has been introduced to many parts of the production line. Common applications of robot

integration include material handling, framing, robotic glazing, palletizing, object lifting, etc. Assembly is one of the most complex tasks on the production line. One of the trends in the robotic industry reported by the International Federation of Robotics (IFR) is that more smart factories are being built to automate assembly lines with industrial robots (Frankfurt, 2021). However, most of the assembly processes are still done manually in many manufacturing industries (Wachol, 2020). This phenomenon is generally attributable to the higher initial cost of automation, the negative impact on employment rate, and a lack of qualified technical support in robotics, which includes translating the products' manufacturing procedure into robotic language. The complexity of the detailed hardware installation tasks and the lack of motion planning techniques are also the main technical challenges the manufacturing industry encounters.

Humans can learn complex assembly tasks with relative ease compared to a robotic learning system. While today's robots are precise and capable of completing repetitive tasks with a precision to within a thousandth of an inch over thousands of hours of operation, such efficiency can only be easily achieved within a simple environment with limited variation of the tasks. If the error tolerances in the installation process is low, or the dimensions and surfaces of the target hardware are not consistent, a conventional robotic system using a position-based robotic control strategy based on nominal part dimensions will be ineffective as the positional uncertainty of the robot will exceed the assembly error tolerance. With that being said, end-effector sensing and an adaptive control scheme must be implemented in order to meet the requirements of the automated assembly operation. Plus, window orders have different sizes and varied hardware, which means engineers require a lot of time to plan manipulators' motions for each order, even

with a simulator. The critical drawback of these kinds of methods is the rules need to be re-designed and re-programed when the object is changed. Intelligent robotic motion generation is a significant contribution to the field because it automates the robotic motion planning such that engineers are not required to manually plan the robotic motion from start to end. Moreover, hardware assembly itself is a complex manual task because one hand is typically required for holding while a screwing motion is required by the other hand. Often, the assembly process also needs real-time control in order to deal with various scenarios such as cooperated robots are moving or collision happening.

1.2 Reinforcement learning for assembly tasks

In the last couple of decades, deep reinforcement learning (DRL) has become an inevitable part of machine learning (ML) and artificial intelligence (AI). Recent developments have proven AI's ability to perform 2D (image) object recognition with the aid of deep neural networks trained on large datasets of human defined labels. However, the process of data collection is repetitive and time-consuming.

The agent from reinforcement learning (RL) framework can learn to complete tedious tasks by interacting with the environment and adjusting its policy critically to the rewards given. It eliminates the need for labeling and manual collection of data, which could resolve possible human error within the process. Since pre-existing domain knowledge is not necessary for the agent, reinforcement learning can be applied to different types of robot arms and tasks.

Reinforcement learning algorithms garnered attention due to their capability on robotic tasks. The general idea is to utilize and reuse the replay buffer to reach sample efficiency through an off-policy involved algorithm. For example, the DQN-NAF (Deep Q Network with normalized advantage function) can achieve sub-optimal performance in reaching task solutions. TRPO (Trust Region Policy Optimization (Schulman et al., 2015)) shows versatility through its reward scaling and parameterization of invariance. DDPG (Deep Deterministic Policy Gradient) (Gu et al., 2016) is unstable for continuous space tasks, as it has high sensitivity on parameters tuning.

Comparisons were also made among different robotic motion related tasks. Varin (2019) found that both the two modern RL algorithms PPO (Proximal Policy Optimization) (Schulman, J. et al., 2017) and SAC (Soft Actor Critic) (Haarnoja et al., 2018) have superior performance. One can conclude from their results that learning references can increase sample efficiency for a task-space impedance control agent.

Haarnoja et al. (2018) found that SAC achieves the best performance by comparing with other popular algorithms such as DDPG, PO, and TD3 among most benchmarks. Even though those benchmarks do not share a direct relation to robot manipulators, the SAC's superior performance and tight fit to the robotic environment make it promising in the robotic field.

In summary, reinforcement learning has gained an increasing amount of attention from developers given its great potential. Developers can utilize reinforcement learning to develop an AI-driven robotic control system. Digitally simulated manufacturing activities, planning and optimization of the robots' motions, collision and error avoidance, and accurate motion trajectory

execution are all made possible by the potential offered by reinforcement learning. Based on these benefits, reinforcement learning with the SAC algorithm was chosen as the approach for development of an intelligent agent in the present research.

1.3 Research objectives

In this research, the motivation is to provide an automated solution of hardware installation to window manufacturers. Due to the challenges of robot control in assembly tasks and the lack of previous case studies on hardware installation using multiple automated manipulators, this research aims to propose a framework of reinforcement learning for automated robot motion path planning. To achieve this goal, the specific research objectives are listed as follows:

- Define a representative scope of window hardware for validation.
- Apply BIM model to generate window and hardware model.
- Develop a simulation environment to model the manipulators and hardware.
- Solve kinematics of manipulators to provide usable action for training purposes and implement pick-and-place tasks.
- Develop an AI agent to meet the requirement of screw and handle motion.
- Train the agent and execute several tools and self-developed methods of reinforcement learning to speed up the training process and improve the success rate.

1.4 Thesis organization

Chapter 2 introduces the selected simulator and the related reinforcement learning technique be implemented. Chapter 3 provides an overview of the research problem, including an overview of the structure of different types of windows, an investigation of the working mechanism of the

windows under study, the requirements of the industry partner, and also a procedure for addressing the problem and for analysis. It was determined that problems faced by window manufacturers are able to be automated with the implementation of sets of robotic arm manipulators. The requirements of the robot manipulator and hardware involved in the installation process were strictly followed. Chapter 3 also describes the preparation work undertaken for the development of the simulation, and illustrates the environment and manipulator settings for the agent's initial setup and for the reinforcement learning training, while Section 3.5 describes the techniques of reward function shaping and parameter tuning to improve the agent's performance in the early training stage. Section 3.6 proposes the curricular guided reinforcement learning structure (CGRLS), which provides a framework for building gradual learning for the agent by arranging the related curricula from the simplest to the hardest. It also shows the application of the CGRLS framework in the context of two different tasks. Chapter 4 analyzes the result after the implementation of reward function shaping, and provides experimental results for training on a single-hardware task and a multiple-hardware task. Finally, conclusions, research contributions, limitations, and proposed future work are described in Chapter 5.

2 RELATED WORK

2.1 Webots simulation software

Webots software is widely used in development of robots control and robotic behavior simulation. As shown in Figure 1, the virtual environment is built upon the scene tree located on the left. The scene tree is composed of a hierarchy of nodes. Different nodes serve different purposes. Solid nodes describe shapes and physical behavior of objects. Transform nodes define the position and orientation of a specific base point or object. Physics nodes define physical behaviours such as mass, joints, friction, coefficients, etc. Robot nodes follow their attached controller script to control the targeted robot behaviour. The combination of nodes gives users the ability to build their own simulation environment and robot models to realize situation-specific simulations. Webots supports programming languages such as Python, C, C++, and languages that fit into the universal robots simulation as long as their ROS system is written by Python. Webots also comes with the ODE physical engine to be able to simulate precise physical behaviour.

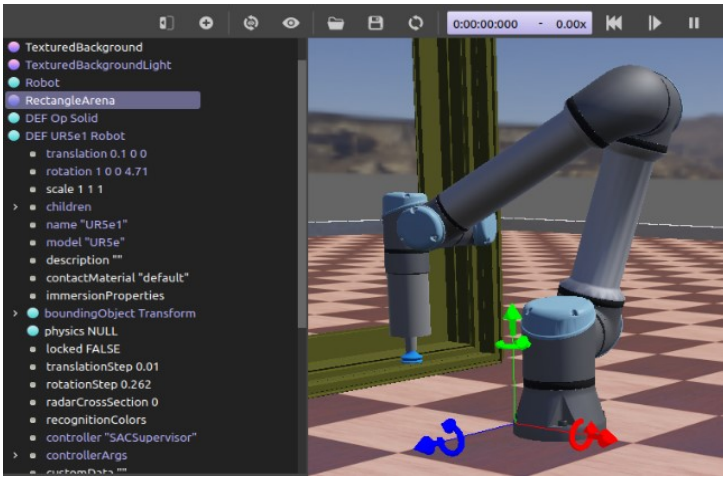


Figure 1. Webots interface

In addition to the regular robot controller, Webots addresses another type of controller called "Supervisor". The supervisor implements operations that are generally carried out by humans rather than by robots. It is able to collect information about every object in the scene and further modify them. The supervisor controller acts like a human supervisor who has control on all the robots. For example, a supervisor can move a robot or a motor to any position, obtain specific object position information, or even reset or finish a simulation. In the case of reinforcement learning, the supervisor performs as an agent to observe and command robots to interact with the environment.

Webots simulator requires relatively less memory and CPU in comparison to other simulators (Ayala et al., 2020). Webots is chosen as the simulation tool in consideration of its user-friendliness, universal robots programming language fitness, and the enormous real time data collection demand.

2.2 Reinforcement learning

2.2.1 Basic concept

A typical reinforcement learning (RL) framework consists of interactions between an agent and an environment. The interaction framework between the agent and the environment is shown in Figure 2. The environment is modeled as a Markov decision process (MDP) and is defined by a set of states S, a set of actions A, a reward function $R(s)$ which provides the agent incentive, a state transition probability $P(s_{t+1}| s_t, a_t)$ and a discount factor γ within the range from 0 to 1. The

policy parameterized by θ is indicated as p_θ. A typical agent goal is to maximize the expected discounted total reward return followed, in which the reward relation is shown in Equation (2-1). Typically, the policy network will be initialized as the parameter $\boldsymbol{\theta}$'. By sampling with the environment, a bunches of trajectories set = (s_1, a_1, s_2, a_2, … s_T, a_T) will be generalized and calculated for the new reward. By using policy gradient method (Sutton et al., 2000) to minimize the loss with the former reward, the policy network can be optimized to improve the decision making of the action. The policy gradient method is shown in Equations (2-2) and (2-3) where $\boldsymbol{\theta}$ is the parameter of the policy network; Ƞ represents the learning rate; $E_{\tau\sim p_\theta(\tau)}$ is the possibility of the distribution to select the trajectory; and ∇ is the gradient ascent calculation.

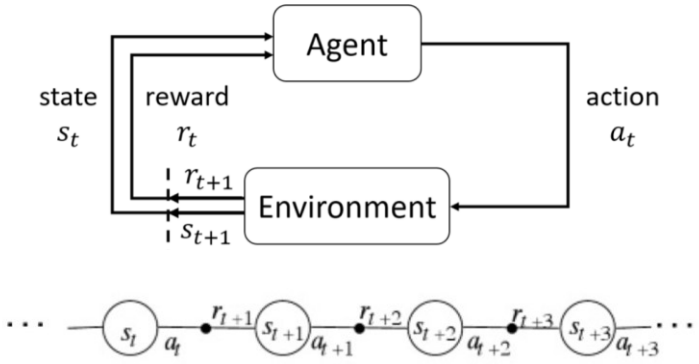


Figure 2. Reinforcement learning basic framework

$$R(s) = \sum_{t=1}^{T} \gamma^t r(s_t, a_t) \tag{2-1}$$

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta \tag{2-2}$$

$$\nabla \bar{R}_\theta = E_{\tau\sim p_\theta(\tau)}[R(\tau)\nabla log p_\theta(\tau)] \tag{2-3}$$

2.2.2 The off-policy concept and experience replay

On-policy algorithms use a single policy network to sample trajectories and finish optimization in the same network, while, in contrast, off-policy algorithms make use of two policy networks. The behaviour policy network samples trajectories within the environment, then the optimization is done by a separate target policy network. Off-policy algorithms have two main advantages over on-policy methods (Lilian, 2018). First, due to using two different networks, the agent can keep exploring while updating the parameters, which can mitigate the total data collection time. Second, the agent can utilize the replay buffer to reuse episodes from the past. Their behaviour policy for sample collection is different from the target policy, resulting in more sample efficiency. When training with off-policy trajectories, distribution shift problems arise, which is defining a way to do importance sampling. Experience replay with the trajectories storage, replay buffer, is one key technique which allows the agent to learn from earlier memories generated by the behavior agent. The replay of experiences can effectively improve its learning rate by providing a high sample efficiency through the reuse of previous data. Such a method saves time by reducing the need for trajectory collection of the agent's interaction with the environment.

2.2.3 Soft Actor Critic (SAC) algorithm

A basic reinforcement learning structure is to train a policy network called the actor. Actor critic algorithms add a critic network on top of the actor in order to criticize its actions taken. Based on this concept, Soft Actor Critic (SAC) algorithms utilize one actor network with two critic networks (conducted by Q function) under a stochastic policy through an off-policy terminology. The main property of the SAC algorithm is the implementation of entropy regularization. The

policy is designed to find a trade-off equilibrium between expected return and entropy, a measure of randomness. This has a tight relation to the exploration-exploitation trade-off. The SAC objective function is presented in Equation (2-4). The notation $\boldsymbol{\pi}$ ($a_t|s_t$) represents the distribution of the action a_t taken by an agent in state s_t. When the action is more random, the log value is lower. The variable α is a non-negative temperature parameter which controls the proportion between the expected return (first summand) and the expected entropy (second summand). During training, α is automatedly adjusted from 0 as an entropy constraint to maximize the total objective. High entropy results in more exploration, which can result in more random actions, while high expected return means that it can more effectively sample the good performance to update the network. SAC maximizes this objective by parameterizing a Gaussian distribution actor policy and a Q-function (critic) with a neural network and optimizing them using approximate dynamic programming.

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t} r(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log(\pi(\mathbf{a}_t|\mathbf{s}_t))\right] \tag{2-4}$$

2.3 Curriculum learning in reinforcement learning

The approach used to train an agent has an impact on the agent's efficiency both in real-world and simulated scenarios. The main idea of curriculum learning is to train the agent with a progressive approach. The training would have gradually increasing difficulty and complexity, which is realized by modifying the environment settings to provide opportunities for the agent to

adapt to a more confounded environment with its previous knowledge learned from a simpler setup.

The idea of training neural networks with curricula was initially proposed by Jeffrey Elman (1993). It demonstrated the importance for a gradual increase of training samples' complexity to the agent's learning. Bengio, et al. (2009) later stated an overview of Task-Specific Curriculum learning which addressed the importance of cleaner examples and gradual increase of the difficulty. Zaremba and Sutskever (2014) trained a LSTM model to predict the output of a Python program. They found that mixing curricula in easy tasks during training can avoid forgetting, and they were able to find an optimal curriculum learning strategy. Weng (2018) then proposed a curriculum guided reinforcement learning case study which utilizes robotic arms with visual input to complete object-grasping automation. Huang (2017) applied a case study of grasping tasks with robot manipulator using visual observation. He demonstrated that sequencing tasks by curriculum learning is able to increase the learning rate and improve the performance. Moreover, Narvekar et al. (2020) proposed a framework that categorizes the elements of curriculum guided learning to task generation, sequencing, and transfer learning. They also mentioned that the tasks may differ during the transferring policies between tasks. The main difference lies in the aspects of the MDP such as starting states, reward functions, or transition functions.

Based on the review of the literature, it was determined that preparations are mandatory in order to follow the mentioned curriculum learning approach. In the context of the present research, a metric to quantify and sort the difficulty of tasks must first be developed. Then a task sequence

with an ascending degree of task difficulty must be provided to the model for training. These preparations have the potential to significantly improve the agent's learning efficiency.

3 METHODOLOGY

3.1 Problem overview

3.1.1 Window overview

A large number of window types exist in the market. Each window type serves a different purpose. Due to cost-effectiveness and supply chain concerns, the industry partner only produces the four kinds of windows shown in Figure 3. The company further welds or mulls these four kinds of windows into a combination window according to the customer's requirements. An awning window is a horizontally hung window that can be swung outward upon opening; a casement window is a window that can be swung outward and hinged on the side. Both fixed and picture windows do not contain moving parts, while the PVC frame profile of the fixed window is thicker than that of the picture window. Windows with moving parts such as the awning and casement windows consist of an interior skeleton part called the frame, and an openable part called the sash.

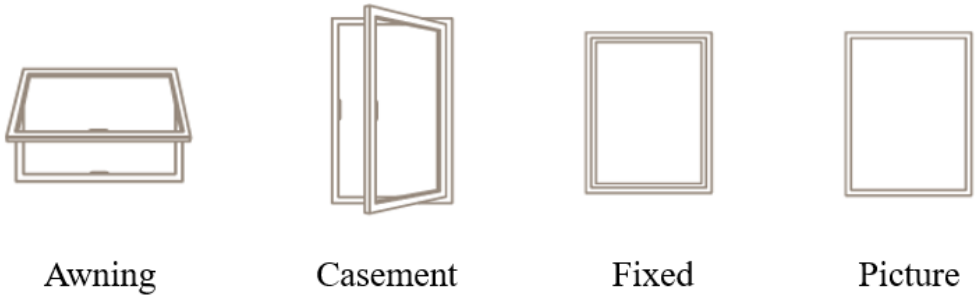


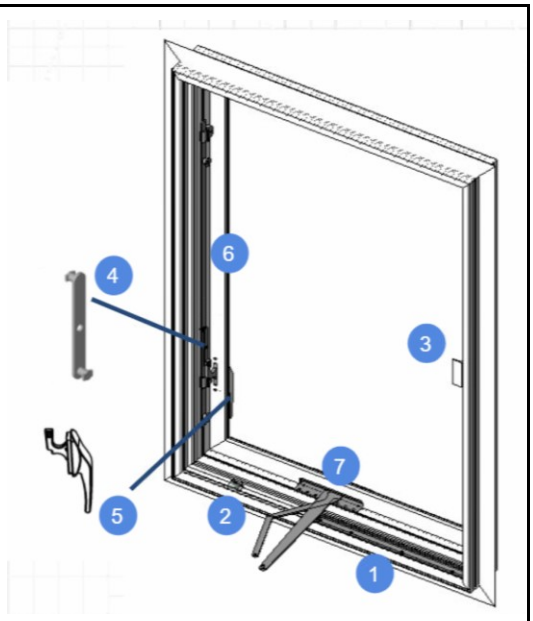
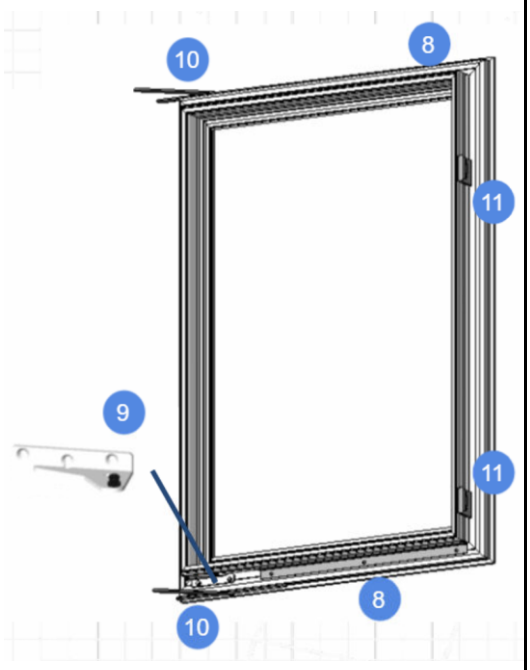
Figure 3. Window types produced by industry partner (Mengni, 2021)

Awning windows and casement windows require hardware mechanisms to make the sashes openable. As shown in Table 3-1, the types of mechanical hardware are grouped by frame and sash. A notable difference between the parts needed to construct an awning window and a casement window is that a casement window requires a ramp and an operator bracket (Mengni, 2021). The amount of screws and exact hardware details used during the hardware installation process are shown in Table 3-1.

Based on the CAD design and provided documents, it was determined there are three different screw sizes, and most hardware items have more than one screw point. This categorization provided direction in terms of selecting the target tasks and choosing an adequate screwdriver system in the later section.

Table 3-1: Summary of hardware

No.	Location	Hardware component name	Screw size	Number of screw	Image
1	Frame	Hinge track	#10 × 9/16 in	4	
2		Ramp	#6 × 3/4 in	1	
3		Snubber	#10 × 9/16 in	2	
4		Tie bar connector			
5		Multi-lock handle	#8 × 3/8 in	2	

6		Tie bar + Tie bar guider	#6 × 3/4 in	1	
7		Operator	#10 × 9/16 in	6	
8	Sash	Operator track	#10 × 9/16 in	2	
9		Operator bracket	#10 × 9/16 in	3	
10		Hinge arm	#10 × 9/16 in	3 or 4	
11		Tie bar keeper	#10 × 9/16 in	4	

3.1.2 Task selection

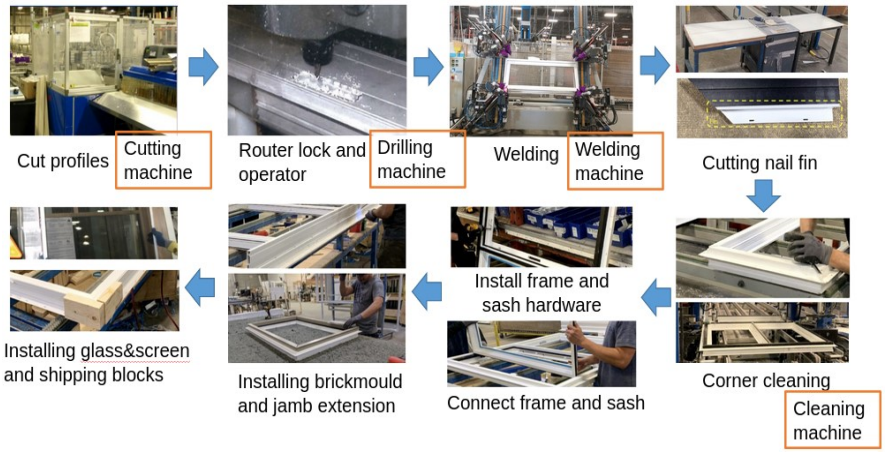


Figure 4. The standard window production line at the industry partner manufacturing facility

As shown in Figure 4, the industry partner already implemented CNC machines at the cutting, drilling, welding, and cleaning workstations. One of the opportunities for improving the level of automation is installing hardware on frames and sashes. The research began with the classification of motions involved with the window hardware installation, which can be categorized into 5 categories as shown in Table 3-2. The result shows that the pick-up, insertion and handle and screw motions are the most common processes during the installation, which makes these three motions economically viable and suitable for automation. The development of a “handle and screw” motion automation was the focus of the present research, as the pick-up and insertion motion can be done by a single manipulator, and thus their automations are more common and mature. Since the “handle and screw” motion involves more than one robotic arm, it is more complicated in nature, which means it is a candidate for being executed using AI-based methods.

In the current model-based robot planning simulator without visual recognition, the traditional method for the pick-up and placement of an object is to pre-define a pivot and destination for the object. The robot can then calculate self-defined Cartesian coordinates or rotation offset to the connection point to adjust accordingly. In the present case specifically, the screwing point is known but the connection point is unknown due to unpredictable hardware shape and existence of external manipulators. And most importantly, this issue cannot be resolved by a manually predefined motion plan. Without visual recognition, manipulators will collide with each other, damaging the consistency of the automation. Although the main concern was the automation of the "screw and handle" motion, the feasibility of full automation for the remainder of the motions involved in hardware installation still need to be considered. It is necessary to make the automation flexible and more economically efficient for the whole industrial process.

Table 3-2. Hardware installation motion category

	Pick-up	Insertion	Screw&handle	Stuck	Punch
Frame					
Install hinge track	√	√	√		
Install snubber	√		√		
Install tie bar guiders	√			√	
Insert multi-lock handle	√	√	√		
Connect tie bar connector	√	√			
Connect tie bar	√	√			
Punch tie bar guider					√
Screw tie bar guider			√		
Install operator	√	√	√		
Install ramp	√	√			

Sash					
Install tie bar keeper	√	√	√		
Install hinge arm	√		√		
Install operator bracket & track	√		√		

3.1.3 Manipulator and end-effector selection

For the abovementioned reasons (described in Section 3.1.2), two manipulators with different responsibilities were selected. One manipulator was equipped with a vacuum gripper and is responsible for all the hardware pick-up, placement, and handling tasks. The other manipulator was equipped with a screwdriver that is able to be modified for different types and sizes of screws. Since the hardware from the industry partner has a weight range from 0.01 kg to 3.2 kg, UR5e Universal robots were used in the simulation for the research. The Robotiq EPick, shown in Figure 5 (a), was picked as the vacuum gripper. As for screwdriver, the Easy Pick and Drive Screwdriving System by Southwestern PTS was chosen shown in Figure 5 (b). The Robotiq EPick will hereinafter be referred to as UR5e-1 and the Easy Pick and Drive Screwdriving system will hereinafter be referred to as UR5e-2.

After several trials with two manipulators in the simulation, the results showed that there is the least chance for them to collide if they face each other. Therefore, the approach of facing the manipulators to each other was selected to achieve optimal results. The frame and sash will be transported to a clamp system which is located between the two manipulators in preparation for the hardware installation process. Figure 6 shows an illustration of the process. Grey circles represent the reachable range of each manipulator.

(a) (b)

Figure 5. (a) EPick vacuum gripper, (b) Screwdriving System by Southwestern PTS

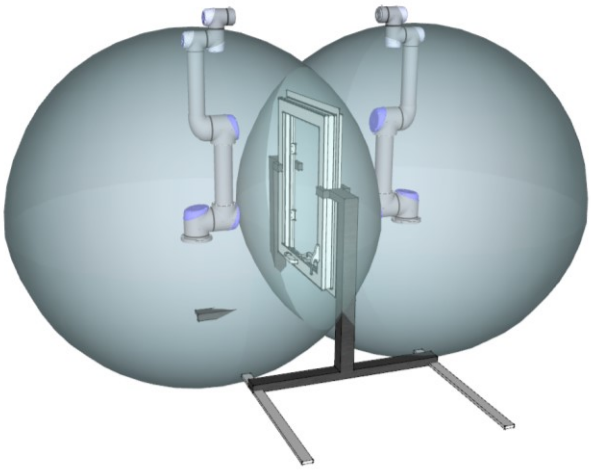


Figure 6. Set up of two UR5e and clamp system

3.1.4 Hardware selection

During the research, three kinds of hardware were employed: hinge track, operator, and ramp, as shown in Figure 7. The goal is then to install these three hardware components to the casement window frame structure. The hardware components were selected because all three have a different size, shape, and possible connection position and orientation. The variety of the hardware provides a gradual learning curve for the process of machine learning. The AI started

the learning process by first installing a cube-shaped ramp, then a long strip hinge track, then finally a T-shaped operator. In addition, the installation position also varied from a single hole, a line of holes, to an anomalous pattern of holes.

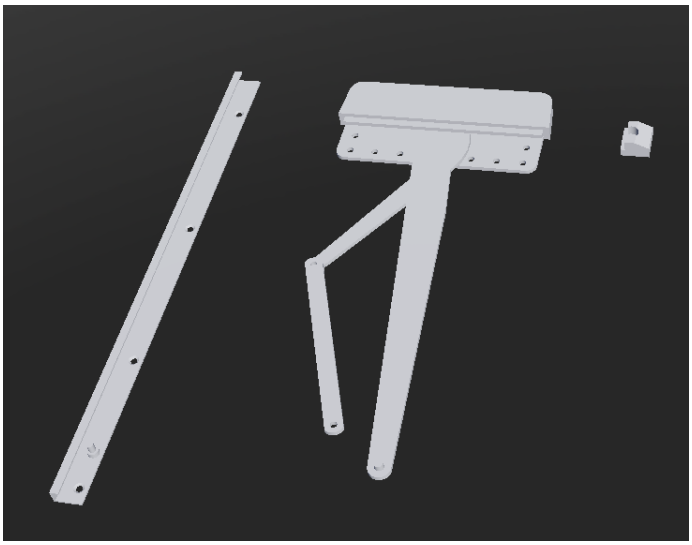


Figure 7. Hinge track (Left), Operator (Middle), and Ramp (Right)

3.1.5 Case study method

Our main goal was to automate an assembly process of installing the three hardware (Hinge track, operator, and ramp) that collisions are avoided when using two manipulators. In general, the installation process can be divided into two phases: the “pick and place” phase, and the screwing phase.

In the first phase, the UR5e-1 was used to pick up the target hardware from a known storage position, then place it in the hardware’s final position on the frame. Next, the UR5e-2 fastens the

screw through the hardware installation point. In the simulation, the hardware was dramatically trembling both vertically and horizontally during the screwdriving. The hardware needed to be held in place during the screwdriving, just like workers using one hand to screw and the other to handle. Hence, finding a proper handling point of the hardware during the screwdriving became the main challenge.

The “pick and phase” motion can be easily executed without the implementation of machine learning in the present study; therefore, the hardware fixation and screwdriving procedure at its final position was the main focus. The largest barrier was to produce an optimal motion for the UR5e-1 to find a handling point and hold it tightly with its vacuum gripper to provide enough support for UR5e-2 to finish the screwing motion. Figure 8 illustrates the detailed process of the “screw and handle” phase. Based on the number of screw points, the UR5e-2 has to switch to other installation spots and UR5e-1 needs to find another handle point when the installation spot is changed. This requires the collaboration of two manipulators and careful motion planning to find a closest gripping position to avoid collision and uncalculated hardware movement.

It is difficult to use traditional methods to manage the motion planning for such a dynamic situation. Hence, a curriculum reinforce learning method was proposed to solve this problem. The primary objective was to train a single multi-tasks model to install the hardware components described in Section 3.1.4 under various conditions.

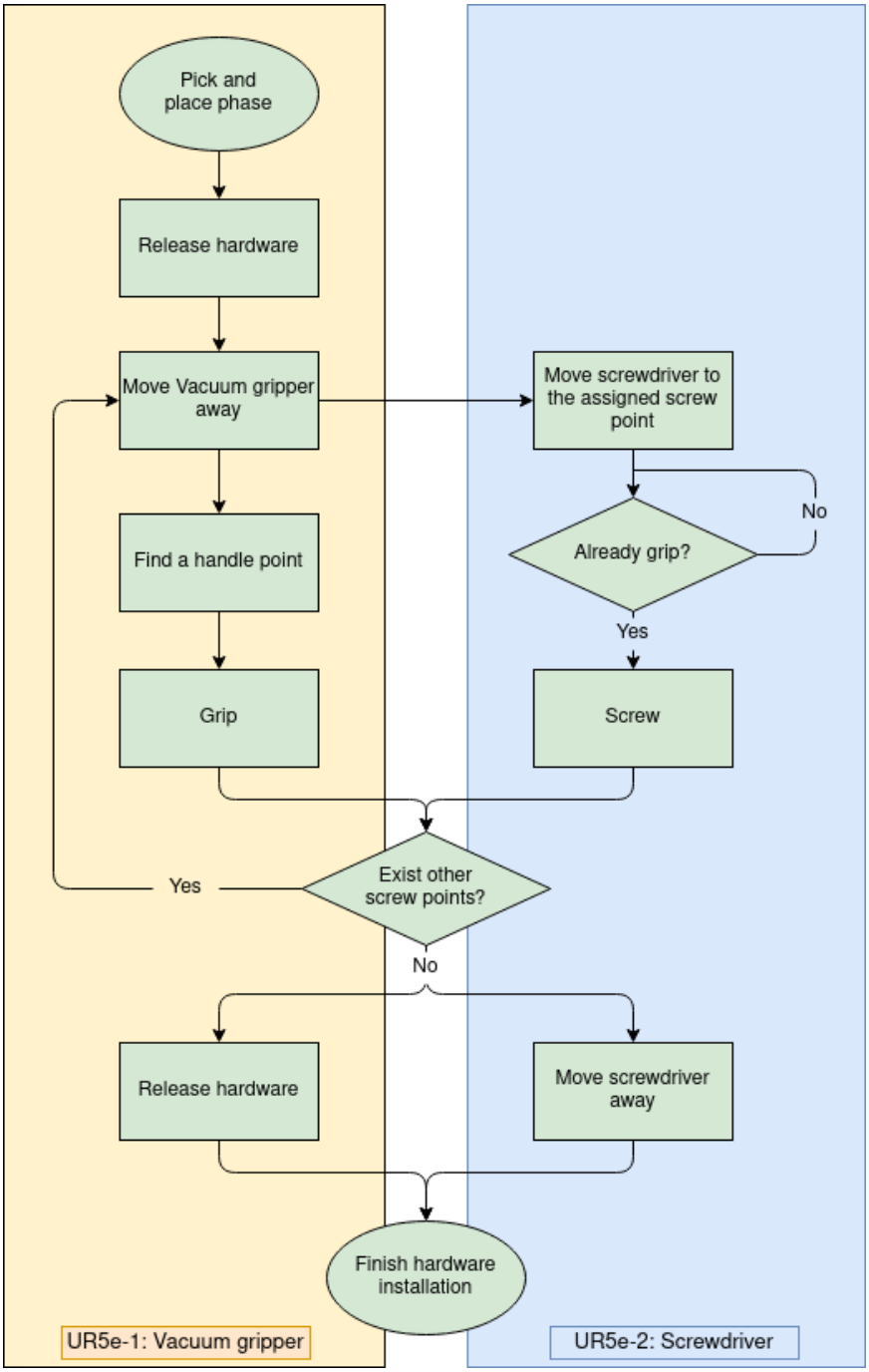


Figure 8. Screw and handle process

3.2 BIM model parameterization and import

The industry partner provides the customer customizable window types and sizes with an enormous amount of choices. The BIM model served not only as a geometric 3D model but also as a source for positional data for the hardware on the frame or sash. For training purposes, a variety of configurations can also provide a solid training environment sample to make the model more robust. In order to meet these demands, a window parameterized configuration adjusting system was designed that is a pre-stage preparation before the simulation. A Revit group was built that includes the host profiles (frame or sash) and the corresponding hardware. By inputting the desired window width, length, and specific hardware usage, the Revit group will automatically adjust the hardware type and position based on the parameterized equation. The mathematical model translation processes for the three chosen hardware components are listed below as Equations (3-1) to (3-4).

To begin, the ramp was determined to be the simplest hardware because it has only one single screw point, and it is only applied to the casement window type. The distance D_{cr} from the inside of the frame to the casement ramp is 4 inches (101.6mm). Equation (3-1) and Figure 9 demonstrate the hinge track screw position calculation for the three kinds of hinge. D_{ht_si} (i = 0-4) is the distance from the i screw center to the frame edge. Equation (3-2) explains the rule of operator selection based on the window width w_w . Equations (3-3) and (3-4) illustrate the operator screw position rule of each type of operator. D_{OCT_six} (i = 0-6) represents a set of screw locations which is the distance between screw center and the side frame edge. D_{OCT_sy} stands for the distance from operator screw center to the frame's outer vertical edge. Lastly, the three types of operator are shown in Figure 10.

In this way, the Fbx. file was able to be exported into the simulated environment depending on the order's requirements. The system was also able to generate different configurations of the product by inputting variables such as length and width since the parameterization done by the BIM model. In this research, the configuration with window length 850 mm and width 600 mm was selected as the target model. The information pertaining to the final hardware position and all obstacles was also able to be imported into the simulation environment with data provided by the window parameterized configuration adjusting system.

$$
\begin{aligned}
D_{ht\_s1} &= 31.8 \\
D_{ht\_s2} &= D_{ha\_s1} + 101.6 \\
D_{ht\_s3} &= \begin{cases} D_{ha\_s1} + 212.7\ (10"\ standard\ hinge) \\ D_{ha\_s1} + 203.2\ (13"\ standard\ hinge) \\ D_{ha\_s1} + 212.7\ (10"\ egress\ hinge) \end{cases} \\
D_{ht\_s4} &= D_{ha\_s1} + 288.9\ (13"\ standard\ hinge)
\end{aligned}
\tag{3-1}
$$

Figure 9. Hinge track screw position on frame

$$Operator\ type = \begin{cases} Reversed\ Dyad\ Operator\ (350 \leq w_w < 500) \\ Single\ Arm\ Operator\ (500 \leq w_w < 600) \\ Dual\ Arm\ Operator\ (600 \leq w_w \leq 950) \end{cases} \quad (3\text{-}2)$$

$$D_{OCT\_s1x} = \begin{cases} 151\ (Standard, Single\ Arm\ Operator) \\ 101\ (Egress, Single\ Arm\ Operator) \\ 101\ (Reversed\ Dyad\ Operator) \\ 216\ (Dual\ Arm\ Operator) \end{cases} \quad (3\text{-}3)$$

$$D_{OCT\_s2x} = D_{OCT\_s1x} + 15.9$$

$$D_{OCT\_s3x} = D_{OCT\_s1x} + 31.8$$

$$D_{OCT\_s4x} = D_{OCT\_s1x} + 76.3$$

$$D_{OCT\_s5x} = D_{OCT\_s1x} + 92.1$$

$$D_{OCT\_s6x} = D_{OCT\_s1x} + 108$$

$$D_{OCT\_sy} = \begin{cases} 24.7\ (Single\ Arm\ Operator) \\ 24.6\ (Reversed\ Dyad\ Operator) \\ 26.6\ (Dual\ Arm\ Operator) \end{cases} \quad (3\text{-}4)$$

Figure 10. Reversed dyad operator (left), Single arm operator (centre), Dual arm operator (right)

3.3 Simulation environment

3.3.1 Building the simulation environment

As mentioned in Section 2.1, all training and experiments were done in the simulation tool Webots, which provides robot nodes and universal robot series for automation development. Universal robot node was utilized to connect the end-effector mentioned in Section 3.1.3. The graphical training process was unstable as the imported end-effector model required a large amount of computational resources. In order to stabilize the simulation environment, the concept of effective shapes was introduced to represent the complex model surface; this concept states that a simplified shape is effective as long as the box collider of the simplified shape covers more than the original shape. For the vacuum gripper, the true shape of the sucker was retained but the connection frame between the sucker and the manipulator endpoint was replaced by a cylinder. The screwdriver system was also simplified into two cylinders with a rectangular box to represent the screwdriver and connection plates, respectively.

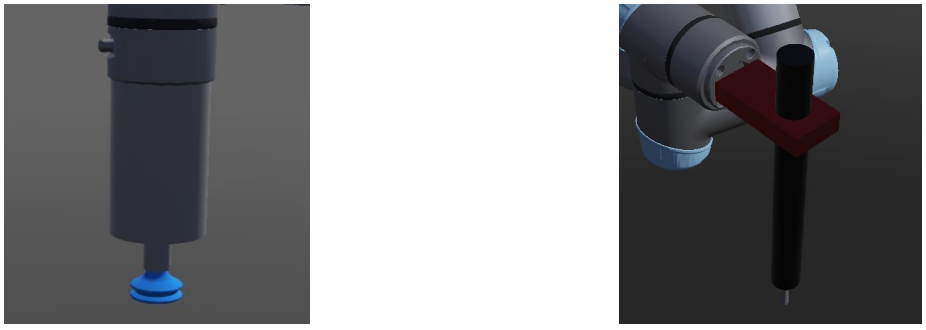


Figure 11. Vacuum gripper simplified model(left), Screwdriver simplified model (right)

Based on the method described in Section 3.2, the completed window model, including its frame profiles with hardware components secured in their final positions, was imported to Webots from Revit. Each hardware component was imported separately for further connect points treatment, which will be described further in Section 3.3.4. After all the models were imported, the hardware installation workstation was configured. Figure 10 shows a screenshot of the configuration. In order to free up computational resources for accelerated reinforcement learning, the environment was simplified by eliminating unessential elements such as manipulator table bases, clamping system, and hardware storage.

The grid plane served as the same plane of the manipulator base table and the height of the clamping system. Both manipulators' end-effectors were able to reach all hardware with a distance of 1 meter between them.

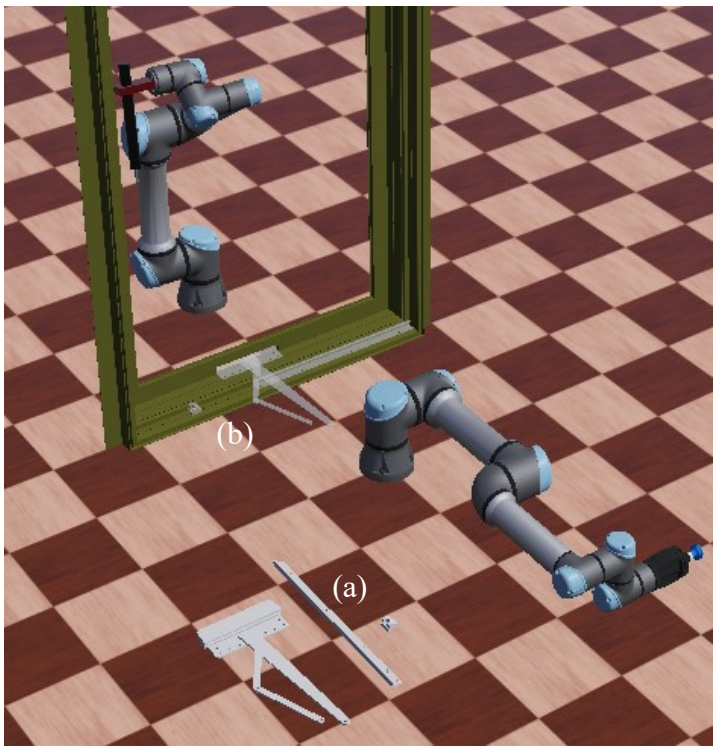


Figure 12. Hardware installation station: (a) initial storage area, (b) installation area

3.3.2 Kinematics of manipulators

Forward kinematics (FK) and inverse kinematics (IK) need to be solved for a six degree-of-freedom (DOF) manipulator to perform specific movements. FK was used to compute the position of the end-effector by changing the six joint parameters to specific values. On the other hand, the IK were used to compute the joint angle from the specified position and orientation of the end-effector. Based on the universal robot configuration document and the summary in the user guide, the FK and IK of UR5e were solved with the support configuration, and IK control was further utilized to generate different motion types and velocity control.

Two basic motion types, point-to-point (PTP) motion and linear (LIN) motion, were implemented in this research. PTP motion moves from one point to another point along the quickest path. In other words, PTP motion makes the least total motor movement, and thus has an advantage of high energy efficiency. LIN motion moves the end-effector to the end point along a straight line, which is able to perform a specific trajectory. The most common practice was adopted in the pick-and-place simulation, which is to combine PTP motion and LIN motion in a certain order to complete the tasks, as shown in Figure 13.

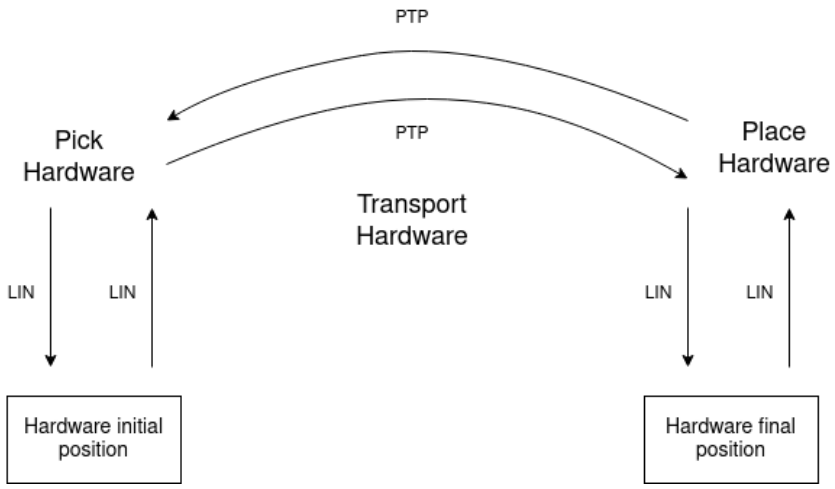


Figure 13. End-effector motion in pick and place phase

Velocity control is a specific method employed to assign specific linear and angular velocity to the end-effector to change the end-effector's position by an arbitrary time step. The main purpose of implementing velocity control was to limit the end-effector's action space for easier training. Velocity control was not necessary in the pick-and-place phase as all the positions are known, and thus can be automated by direct instructions. However, velocity control was needed for the screw-and-handle phase as it requires the end-effectors to remain in their action space. The concept of action space will be discussed further in Section 3.4.4.

3.3.3 Hardware connect simulation

The physical properties of the hardware components played a vital role in the present research. A specific node called "connector" in Webots was used to simulate a real-world condition that any surface can be gripped. Connector nodes were used to simulate mechanical docking systems that can dynamically create a physical connection with another device. As shown in Figure 14, when

the vacuum gripper's z-axis (blue line) is aligned with any target device's connector's z-axis, the hardware can be gripped. If the rotation tolerance is set to 0, it means that not only the z-axis but also the y-axis has to be aligned. The connector was evenly distributed to every surface and confirmed that the hardware can be gripped even if the vacuum gripper stays at the middle point between two connectors.

For single model training purposes, all three hardware components were placed into the same observation space. which means the amount of connector had to be the same across different hardware no matter their type or size, as they all had to share the same observation space dimension.

In this research, 29 connectors were used for each hardware component. The distribution of connectors of the hardware is shown in Figures 15, 16, and 17. The distance tolerance was set to 15 mm with a full tolerance on rotation to allow a more dynamic connection.

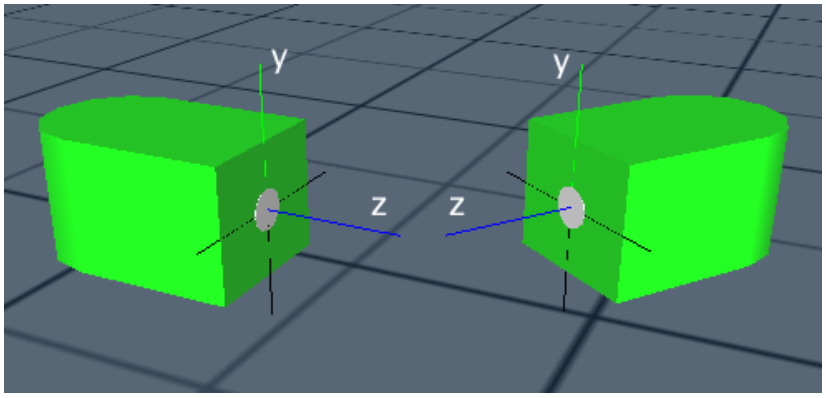


Figure 14. Connector axis

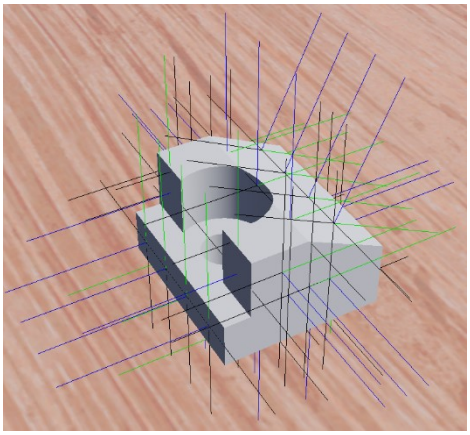


Figure 15. Ramp connector distribution

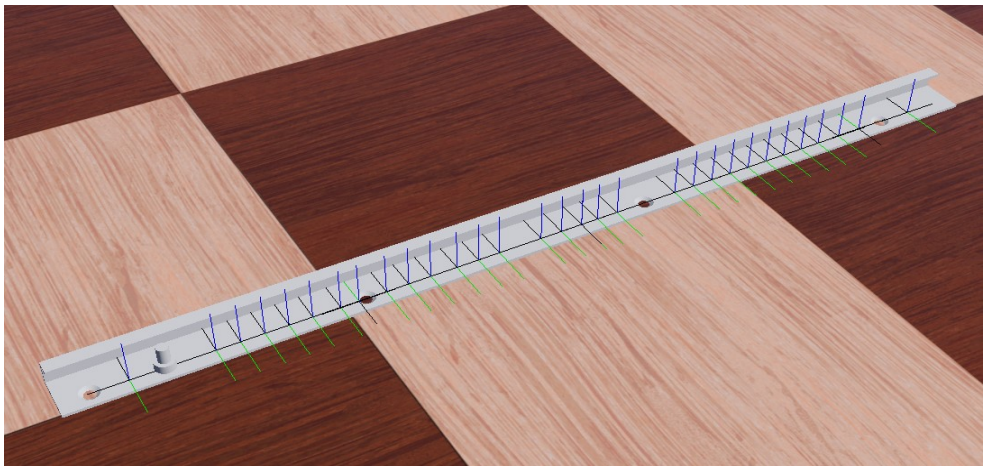


Figure 16. Hinge track connector distribution

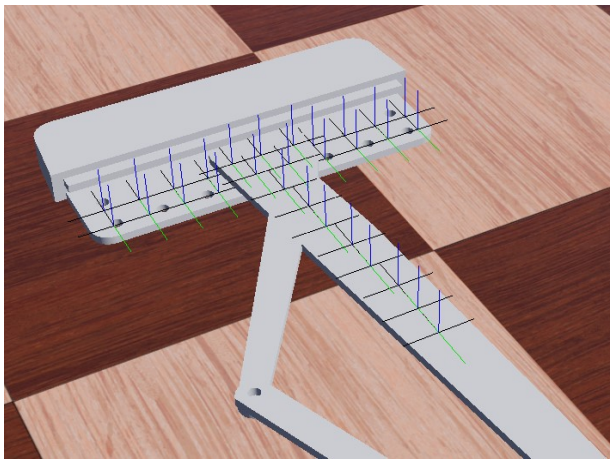


Figure 17. Operator connector distribution

3.4 Task specifications

3.4.1 Agent overview

As described in Section 3.1.5, the objective was to find a collision-free path to reach a proper handling point, which will be primarily implemented by UR5e-1. UR5e-1 was chosen as the agent to decide the action based on its observation and reward. UR5e-1 used a supervisor and robot combined controller, while UR5e-2 used the only robot controller mentioned in Section 2.1. The agent on UR5e-1 was able to gain positional and orientational information of the two end-effectors and hardware components. Figure 18 shows a detailed environment data flowchart for the communications between UR5e-1 and UR5e-2. Both manipulators had their own emitter and receiver nodes that were able to send string data to each other. During the simulation, two manipulators were allowed to collect their motor position and collision situation and send those data to the supervisor through their position sensors and touch sensors, respectively. The supervisor can also send motion commands to control both UR5e-1 and UR5e-2. Since the hardware is not a robot node, the position and orientation information can only be observed by the supervisor. In other words, the hardware data message was directly obtained by the supervisor controller.

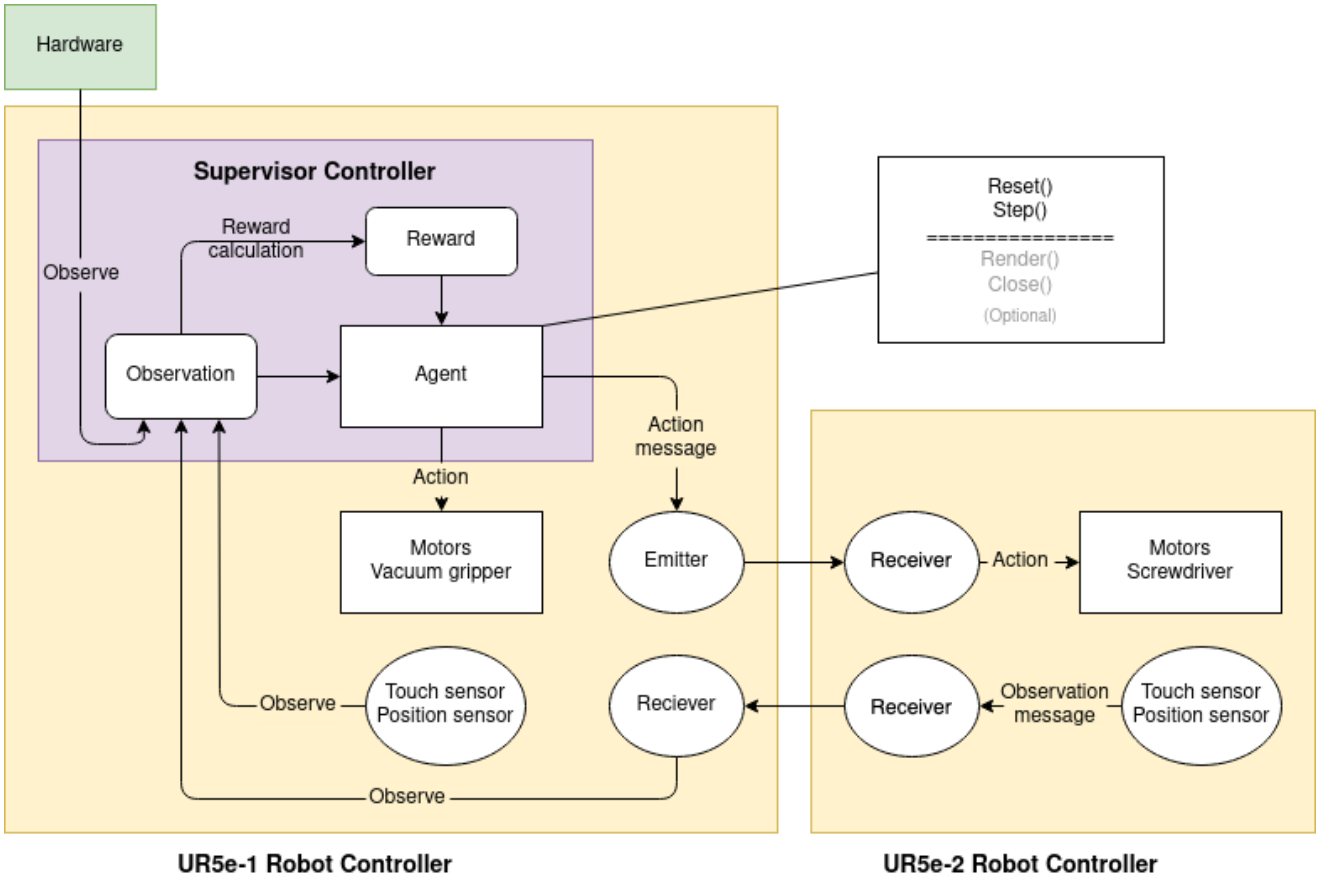


Figure 18. Environment data flowchart

In the supervisor script, the class structure followed the same agent-environment loop as the OpenAI Gym framework. In this research, the maximum number of steps ep_n in a single episode was set to 100 during the custom environment creation. The agent will end an episode when failure happens or an ep_n is reached, which can avoid useless steps in simulation. In the environment, the agent must contain the reset method (Reset()) and the step method (Step()) to execute reinforcement learning. After a custom environment was created, the stable baselines3 package (Raffin, 2019) was utilized to start the training with the SAC algorithm. The detailed content will be demonstrated in the following sections.

3.4.2 Reset Method

The reset method, (Reset()), was called at the beginning of each episode. It initializes the environment to be prepared for the upcoming episode, then returns an initial state of the observation to the agent. Figure 19 contains the pseudo code of the reset function. In the supervisor controller in Webots, the variable of episode number ep_n, step number n, and reward were first defined and initialized. After that, the agent will reset the simulation to zero seconds and ensure that all objects and the physics engine are turned back to the initial state. Next, the hardware will be randomly set to a position within the possible configuration to make sure the environment has enough complexity during the training process. Finally, $S_{final\_hardware}$ will be the final state on the frame profile.

In the present research, different curriculum tasks were designed to be conducted by different initial states for the two manipulators. Basically, as shown previously in Figure 8, all possible scenarios were categorized into two categories: installation of the hardware with one screw point, and installation of the hardware with more than one screw point.

In the cases with only one screw point, S_{ini_ur1} stands for the end-effector initial states of UR5e-1, which represents information such as its location, rotation, etc.. Although the distance from the end-effector and the hardware will vary depending on the curriculum difficulty, the initial state generally stays at a point near the hardware. Then, the S_{final_ur2} represents the final state of the end-effector of UR5e-2, as the target screw point is its only destination. However, in the second

category, the S_{ini_ur2} is needed to represent the initial state of the end-effector of UR5e-2, as it will need to move from a random finished screw point to the target screw point. This scenario happens when the last screw point is secured and the UR5e-2 end-effector is ready to move to the next screw point, which then will be the location of the S_{final_ur2}. Step (1 timestep) was to define the simulation graphic output timeframe to a frame per 4-11 milliseconds. This function is necessary to implement robot motions in Webots.

With every reset called, the inverse kinematics (IK) solution was first calculated to make sure the point is reachable, then they were let go to the assigned position to check if there is any collision at the assigned position and the path between the start position and assigned position. Starting with a perfect situation is important, or else the training result would be imprecise due to the error caused by the beginning position. In the end, the supervisor controller will collect the observation and start looping the step function.

```
function Reset ()
        Input: The episode number ep_n , step number n, reward.
        ep_n = ep_n + 1
        n = 0,  reward = 0
        Step (a)
        Reset simulation
        Reset physics
        Set hardware to its final state S_final_hardware by random
        while no IK solution of UR5e-1 and UR5e-2 do
                if screw point > 1
                        Get the S_ini_ur1 , S_ini_ur2 by random
                        Set the end-effector of UR5e-1 and UR5e-2 to S_ini_ur1 , S_ini_ur2
                else
                        Get the S_ini_ur1 by random
                        Set the end-effector of UR5e-1 and UR5e-2 to S_ini_ur1 , S_final_ur2
                end
                repeat
                        Step (1 timestep)
                        if detect collision from UR5e-1 or UR5e-2
                                go back to Step (a)
                        end if
                until the end-effector of UR5e-1 and UR5e-2 reach S_ini_ur1 , S_ini_ur2
        end while
        Get observation Obs
        return  Obs
end function
```

Figure 19. Pseudo code of reset function

3.4.3 Observation space

The observation space is the format of valid observations for an environment. The common space includes discrete space and box space. The discrete space allows a positive integer to represent its related value such as true/false or a value from known choices such as North/East/South/West or a colour code within the RGB spectrum. The box space consists of an n-dimensional continuous space. The box space is bounded by an upper and lower limit. The

value can be any value within the corresponding dimension's limit; in other words, each dimension gives a continuous variable.

In this research, the observation space was set as a 2-dimensional box space with the dimension 39 × 6. The first dimension with 39 arrays was composed of the states of the two manipulators, screw points, and their connectors. All arrays shared the same definition: a second dimension containing 6 continuous values. The first three elements represent the cartesian coordinates, x, y, z, while the latter three represent the euler angles denoted as α, β, γ. The end-effectors were able to be oriented by the known target object position and these 6 positional and orientational values.

Table 3-3 shows a detailed allocation observed by the agent, calculated using relative position instead of absolute position except for the observation of the UR5e end-effector end point of its own. The reason for this design was to increase the richness of the observation, which was beneficial to the agent's cognitive ability to the environment.

The bound varied within the range [-1,1] after normalization using Equation (3-5). $Midpoint_i$ is the center point of the $Maxrange_i$, which is the combined reachable range of two manipulators, along with the i axis (i = x , y , z). Because observation space was used as an input to a neural network, normalizing the input to neural networks is useful for increasing convergence speed, improving computer precision, and preventing divergence of parameters, which then leads to easier hyperparameter tuning.

Table 3-3. Content of observation space

Amount	Target object point
1	The end point of UR5e-1 end-effector
1	The end point of UR5e-2 end-effector relative to the end point of UR5e-1 end-effector
6	Each motor centre point relative to the end point of UR5e-1 end-effector
1	Centre point of the UR5e-2 end-effector connect plate relative to the end point of UR5e-1 end-effector
1	Screw point relative to the end point of UR5e-1 end-effector
29	Each hardware connector relative to the end point of UR5e-1 end-effector

$$Pos\_i_{\,normalization} = \frac{Pos\_i_{\,original} - Midpoint_{\,i}}{Maxrange_{\,i} \div 2} \qquad i = x, y, z \qquad (3\text{-}5)$$

3.4.4 Action space

The action space defines the range of valid actions. As for the observation space, this could be a discrete number of options, such as open or close for a gripper, or a continuous range such as the target velocities for seven joints. In this research, multi-step motion (up to 100 steps) was implemented in the whole process. By syncopating the motion into small time periods, the path can be optimized in a smooth way and the agent can be sensitive enough to adjust when urgent conditions happen.

There are three kinds of control methods used to perform manipulators' actions: motor control, IK control, and velocity control. Using motor control, the agent will send commands directly to the six motors to rotate assigned distances. Motor control is the least efficient way to navigate the end-effector to reach the target objects. Based on the knowledge of IK, the agent can assign a random location in the reachable range for the end-effector. The end-effector can reach it by moving motors to achieve a calculated distance. This method faces a crucial problem in that the interval between steps is unstable. If the location is too far away from the previous one, the motors require a large movement and the end-effector may jump. Moreover, the relevance between steps is low resulting in inefficient training. Velocity control is an improved version of IK control. As mentioned in Section 3.3.2, velocity control assigns a specific linear and angular velocity to the end-effector. The end-effector position will then move with the velocity assigned in each step, resulting in a stable and smooth motion of the end-effector.

After several trials, the velocity control method was chosen as the path planning rule. Note that all parameters in the action space will also need to be normalized to the [-1,1] range as observation space for training purposes. The action spaces for the three abovementioned control methods are demonstrated in Equations (3-6), (3-7), and (3-8).

$$Motor\ control : Space.Box\ [Motor_1, Motor_2, \ldots, Motor_n] \quad n = number\ of\ DOF \tag{3-6}$$

$$IK\ control : Space.Box\ [Pos_x, Pos_y, Pos_z,\ \alpha,\ \beta,\ \gamma] \tag{3-7}$$

$$Velocity\ control : Space.Box\ [V_x, V_y,\ V_z,\ R_x,\ R_y,\ R_z] \tag{3-8}$$

3.4.5 Step method

The step method takes an action as input and utilizes the action to interact with the environment at each step. At the end of the step function, it has to return observation, reward, done, and info. Infos is a dictionary that can be used to return additional data if needed. Reward is a huge part of this research which is related to reward shaping technique and will be further introduced in Section 3.5.1.

In the observation part, it should return a state that matches the pre-defined observation space as mentioned in Section 3.4.3. In the step method, the order of the 29 connector observations was shuffled in each step to avoid having the agent only memorize certain gripping motions. This design was for the purpose of robust agent training and the ability to recognize the irregularity of the hardware component's shape.

Done is a Boolean value. When the value is True, the episode will be ended and immediately call the Reset function. Otherwise, it will keep looping the step function until the step number reaches the max episode step (100), which wastes a lot of time in task failure situations.

To accelerate the training progress, four different scenarios will return True to the done function and halt the step function: 1) reach any connect point; 2) any collision happens; 3) one of the manipulator end-effectors gives no IK solution; and 4) exceed max episode step. With the aforementioned setup, the training process yielded episodes of various lengths, which saved an enormous amount of time and provided a useful method to validate the training result.

Figure 20 shows the pseudo code of the step function. As mentioned in Section 3.1.5, when the number of hardware screw points is larger than 1, the agent will send commands to UR5e-2 to make its end-effector move to the assigned screw position S_{final_ur2} at the first step. Action space A is automatically generated by the agent and is used for calculation of the next state.

```
function Step ()
    Input: A set of action space A (Vx,Vy, Vz, Rx, Ry, Rz) , max episode step ep_max = 100,
    step number n = 0, reward = 0.
    n = n + 1
    if screw point > 1
        if n = 1
            Set the end-effector of UR5e-2 to S_final_ur2
        end if
    end if
    Set the end-effector of UR5e-1 to move A X timestep
    Obs = get_observation ()
    reward = get_reward ()
    done = get_done ()
    info = None
    return Obs, reward, done, info
end function
```

Figure 20. Pseudo code of step function

3.5 Reward shaping and model architecture

3.5.1 Reward shaping

Reward shaping is a technique to transfer domain knowledge into reinforcement learning. As mentioned in Section 2.2, reward will make the agent define the quality of certain performance. Moreover, effective reward shaping can lead to high sample efficiency and speed up the agent's training process. In this section, we will define the reward function used in this research and discuss the derivation. With respect to the modification of the reward function, the experiment results will be discussed further in Section 4.1.

According to the step method, the agent should return a value of reward with the function get_reward () as shown in Figure 18. It is defined as successful if the end-effector got closer to any hardware connector in each step to provide an incentive for the agent to locate a handle point at the end. Equations (3-9) and (3-10) show the first version of the reward function. R_1 is the reward function of the first version. In the first row of Equation (3-10), a penalty is given on collision and reaching a point out of range. Moreover, in order to encourage the agent to reach the goal, 10 points were given whenever it succeeded. The agent was allowed to earn more points when it connected the hardware closer to the force point. However, if the agent was unlucky did not succeed once, this led to sparse reward situations and was hard to converge. Some bonus points were thus added to gain more points when getting closer to the hardware during exploration. D_ec_i is the distance between UR5-1 end-effector and one of the 29 connectors $connector_i$. The variable $eularD\_ec_m$ represents the eular distance between UR5-1

end-effector and the connector that has the smallest distance between it and the end-effector. *D_cs* represents the distance between the connection point and the screw point.

$$m = argmin(D\_ec_1, D\_ec_2, ..., D\_ec_i) \qquad i = number\ of\ connector \tag{3-9}$$

$$\begin{aligned} R_1 = {} & -Failure \times 10 && if\ collide\ or\ no\ IK\ solution: Failure = 1\ or\ Failure = 0 \\ & + Success \times (10 + D\_cs) && if\ gripped: Success = 1\ or\ Success = 0 \\ & + 2 - normalize(min(D\_ec_1, D\_ec_2, ..., D\_ec_i) - normalize(eularD\_ec_m)) \end{aligned} \tag{3-10}$$

This reward function turned out to be ineffective, as most episodes ended with either a collision or no IK solution. Most of the time, the end-effector chose to wander around until it reached the episode step limitation. The equations were then modified to Equations (3-11) and (3-12) to flatten the reward to ensure the agent will not be heavily punished on failures. With the modified reward function, the sum of reward was more regularly distributed and had less jump value. The variable r_t represents the current reward. When failure happens, it will first give a penalty ep_{max} which is equal to the max episode step, and then set the reward for the remaining steps to the current reward. This design avoided the extremely low reward while remaining a lower sum of reward than the situation in which the end-effector was closer to the handle point. Exponential normalization was also used for the distance-related reward to make the agent more sensitive to the distance between the end-effector and the handle point.

$$\begin{aligned} r_t = 2 - exp\_normalize(min(D\_ec_1, D\_ec_2, ..., D\_ec_i) - exp\_normalize(eularD\_ec_m)) \\ m = argmin(D\_ec_1, D\_ec_2, ..., D\_ec_i),\ i = number\ of\ connector \end{aligned} \tag{3-11}$$

$$\begin{aligned} R_2 = {} & Failure \times (-ep_{max} + r_t \times ep_{remain}) && if\ collide\ or\ no\ IK\ solution: Failure = 1\ or\ Failure = 0 \\ & + Success \times (2 \times ep_{max} \times (1 - normalize(D\_cs))) && if\ gripped: Success = 1\ or\ Success = 0 \\ & + r_t \end{aligned} \tag{3-12}$$

The second version strongly improved the performance compared to the first. However, there were two different problems which led to frequent failure. Firstly, when the end-effector was close to entering the connection range, the next step movement was often too fast such that the sucker crushed into the hardware. Secondly, due to the nature velocity control being written by IK, the manipulator sometimes chose to move motors in an inefficient way. For example, a motor's rotation range is from -6.28 degrees to 6.28 degrees. A path was created which required the motor to move from -6 degrees to 6 degrees. The moving distance was small but it took a long time to make an entire motor revolution. To solve the problem, a velocity factor (when an episode is terminated) and a penalty of the motor rotate distance $MotorD_j$ were added to the equation shown in Equation (3-13). The result showed an increased success rate and will be discussed further in the later section 4.1.

$$\begin{aligned} R_3 &= R_2 + Success \times (1 - normalize\,(\,max\,(\,|V_x|\,,\,|V_y|\,,\,|V_z|\,)) \\ &\quad - normalize\,(max\,(MotorD_1, MotorD_2, \ldots, MotorD_j)) \quad j = number\ of\ DOF \end{aligned} \tag{3-13}$$

3.5.2 Parameter tuning

Parameter tuning can have significant effects on the training result. A proper neural network size is vital for machine learning: a model with low capacity will not be able to learn. On the contrary, a model with too much capacity can learn the training dataset too well, resulting in an overfit phenomenon. Big architecture also takes more time to update the networks which increases the training time. Two fully connected layers of 400 × 300 neurons were used and the

network architecture was shared with both actor and critic networks. MlpPolicies, which are made for other types of features such as robot joints, sets of vectors, was chosen. The minibatch size for updating the network each time was set to 256. Replay buffer size was set to 10^6. A discount factor of 0.99 was used, in which case the future reward will be discounted by a coefficient of 0.99. The coefficient τ, which relates to the soft update rate, was set to 0.02.

Learning rate is an important variable in deep network training that represents the update weight in the stochastic gradient descent. It was determined a higher learning rate in the early training stage can boost the network updating speed, while a smaller learning rate in the late stage can further improve the success probability.

Most of the time spent on this part of the research was to find an optimal learning rate—it was dynamically tuned during the training period. At first, a common practice was adopted to schedule the learning rate to decrease linearly from training step. Because the training step varied from 10,000 steps to 1,000,000 steps, the change of learning rate became negligible in terms of affecting the training process when the number of training step got too long. Figure 21 shows the approach taken to tune the model's learning rate (LR).

Different learning rates will be set under the approach illustrated in Figure 21 to start training according to the task difficulty at the beginning. If the success rate increases, the learning rate will then be minimized when the success rate is over 0.75 and 0.95 based on the experience.

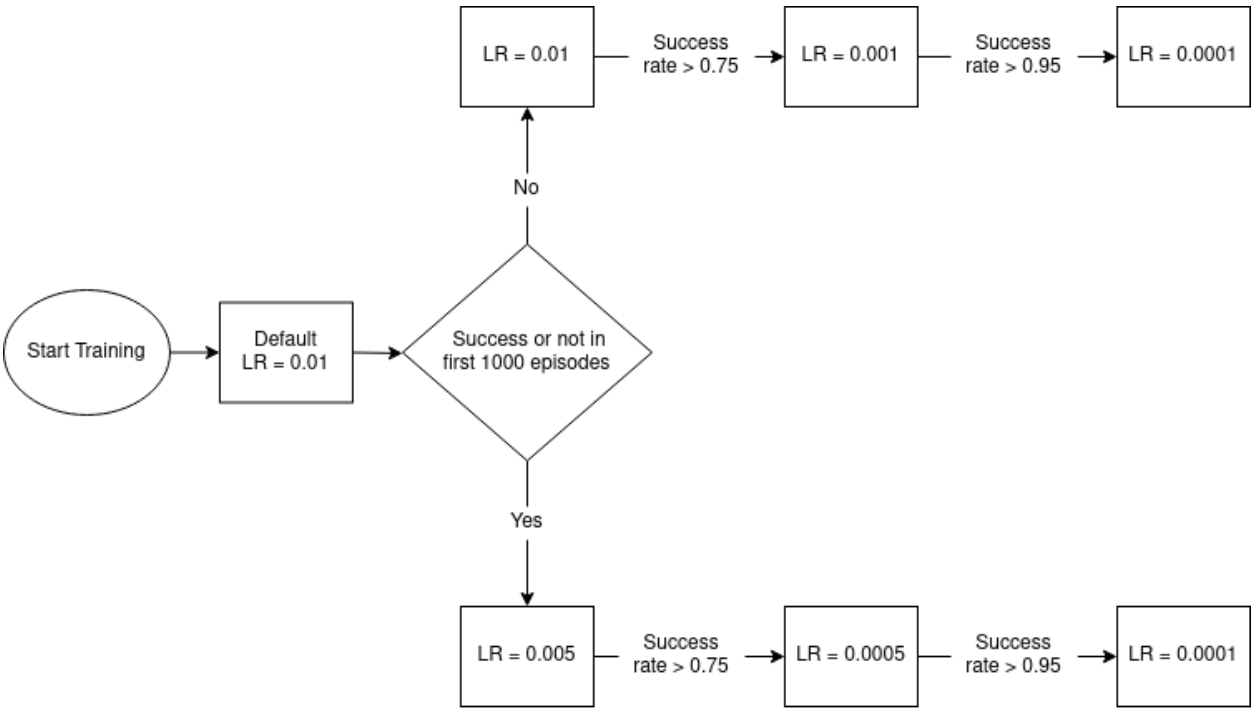


Figure 21. Learning rate tuning strategy

3.6 Curricular learning and task design

3.6.1 Curriculum guided reinforcement learning structure (CGRLS)

One of the most critical challenges machine learning faces in real-world application is the time resources required for training. That is why curricular learning is an essential tool as it reduces training time. Curriculum learning involves a gradual ascending learning procedure. The training would start from a simple setting and task, then the difficulty and complexity will increase after a successful trial. In this way, the agent can apply its experience from the basics to a more advanced setup to ensure a smooth learning curve. A curriculum guided reinforcement learning

structure (CGRLS) was proposed and implemented into the application. The following concepts are the main guideline the CGRLS followed: 1) The scale of objectives matters in the curriculum design. 2) The training starts from a simple environment. 3) The model increases the training environment's complexity and difficulty gradually. 4) The model avoids catastrophic forgetting by providing previous experience. Figure 22 shows an overview of the proposed method.

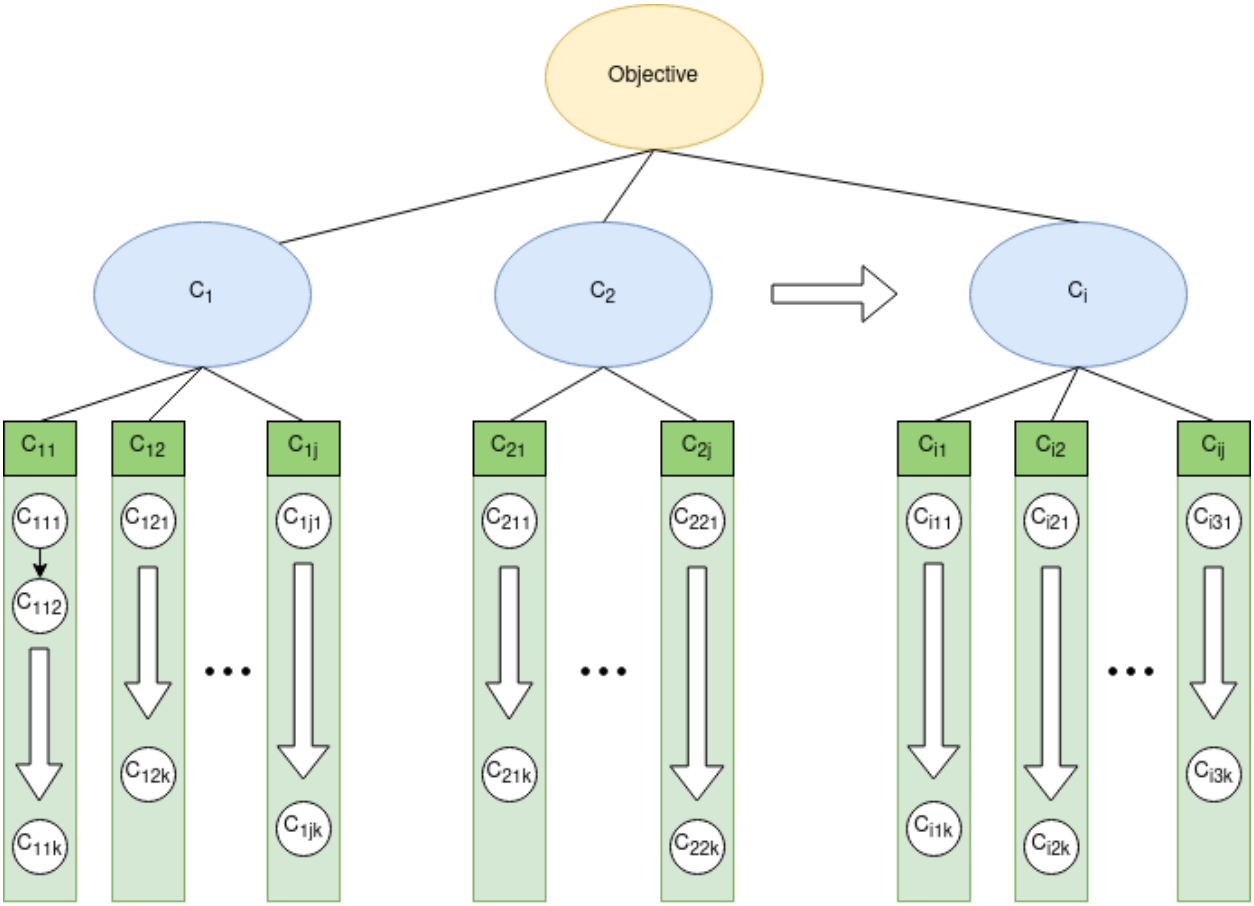


Figure 22. Curriculum tree with two layers

First, the scale of the objectives was defined. The agent will be able to solve a problem only if the objectives are pre-defined. The next step was to build a curriculum tree based on the objectives' definitions. The vertical tree layers represent the hierarchy of the sub-curriculum. In

the first layer, the objective can be a single task or can include multiple different but similar tasks horizontally. If there is only one objective, then it leads to its vertical sub-layer. If not, the task curriculum will be listed and ordered by their difficulty from C_1 (task with the easier difficulty) to C_n (task with the hardest difficulty) horizontally, while n represents the total number of tasks. This process was repeated in every horizontal layer until it reached the bottom. In the bottom layer, depending on the program complexity, a number of influential parameters C_{ij} will be set. For example, it could be the pole length in the Cart-pole environment or x-axis space of the Atari game. After defining each parameter, the sub-curriculum C_{ijk} of each parameter is then arranged from easy to hard, again horizontally, to finalize the curriculum tree. The amount of sub-curriculum needed for each parameter can be different, but the increases in difficulty should be as even as possible. The more sub-curriculum is in the parameter, the flatter the difficulty growth rate is.

The final step was to group sub-curricula according to the selection rule and combine them to the final curriculum stage. The curriculum tree stage design was then completed by the following procedures. First, at least two top-curriculums were selected to focus on the top layer starting from C_1. Among the focused curricula, all sub-curricula start from the difficulty level C_{xy1} (the easiest curriculum), which becomes the curriculum stage 1 for training. After that, at least two parameters of each top-curriculum were picked to increase difficulty for the next stage. The rate of difficulty growth depends on the number of the final curriculum stage, as the intent was to spread the knowledge growth among parameters as evenly as possible.

Note that all parameters had to be updated at least once before any parameter's sub-curriculum was completed. This ensures a similar rate of growth between parameters and to avoid the agent

forgetting an early finalized parameter experience. The selected sub-curriculum then went to the next final curriculum stage, and the previous final curriculum stage then became the starting point. These procedures were repeated until every sub-curriculum was selected. During the process, additional top-tasks can be added to the pool to provide a flexible learning model.

The training results indicated that if a certain final curriculum stage was not able to achieve its first success within its first 5,000 episodes, its success rate for the upcoming curriculum stages was near to zero. This phenomenon indicates that the difficulty growth rate has to be flattened if the agent is not capable of completing a given curriculum stage within a certain amount of time. In summary, a flattened difficulty growth rate with a smoother path can provide a more efficient training experience for the agent. However, there is no hard rule to determine the optimal timeframe and the number of curriculum stages that can yield the most efficient results: A smoother curriculum can provide the agent an easier learning pathway; however, it will require a much more complicated model processing procedure, resulting in a more time-consuming task for the developer.

Figure 22 shows a curriculum tree sample of 2 vertical sub-layers with *i* proposed tasks. The white arrow shows the increase in the degree of difficulty of the tasks within parameters. The white dot represents the sub-curriculum of a particular parameter. In the parameter variable C_{ijk} , *i* is the number related to its top tasks; *j* represents the number related to the parameters of each top task; k stands for the total number of sub-curriculums of each parameter. This variable is used to further illustrate the change in difficulty of each parameter. Note that parameter C_{11} consists of a curriculum stage with an increased amount of curriculum. This implies that the

agent failed to succeed once with its first 5,000 trials during the direct path from C_{111} to C_{11k}, so that the curriculum stage C_{112} needs to be introduced to smoothen the learning difficulty growth rate. Figure 22 illustrates an optimal progress with evenly distributed parameter knowledge growth represented by a similar vertical distance travelled by the final curriculum stages.

3.6.2 Curriculum task design

This section will illustrate how the CGRLS technique was used to design the curriculum stage of the two tasks. The objective of Task 1 was to train an agent who was able to guide UR5e-1 to locate a ramp handling point. Table 3-4 shows the curriculum tree of this objective. Because there was only one single task, the top layer number was set to 1. The set of parameters is represented as $C_{i,}$ and the sub-curriculum will be named C_{ij} . The geometry display of the parameters of Task 1 is shown in Figure 23. The final curriculum stage is introduced in Table 3-5. The bold font represents the update by picking sub-curriculums of each parameters when making a new curriculum stage.

Table 3-4. The curriculum tree of Task 1

Objective	Locate ramp handling point		
Parameter	C_1 = UR5e-1 initial X distance from ramp	C_2 = UR5e-1 initial Y distance from ramp	C_3 = UR5e-1 initial Z distance from ramp
Sub-curriculum	C_{11} = 0.03	C_{21} = 0	C_{31} = 0
	C_{12} = [0, 0.01, 0.02, 0.03]	C_{22} = [-0.02, -0.01, ..., 0.04]	C_{132} = [-0.03, -0.02, ..., 0.03]
	C_{13} = [0, 0.01, 0.02, ..., 0.1]	C_{23} = [0, 0.01, 0.02, ..., 0.16]	C_{33} = [-0.08, -0.07, ..., 0.08]
	C_{14} = [0, 0.01, 0.02, ..., 0.2]	C_{24} = range(0, 0.3)	C_{34} = range(-0.15, 0.15)
	C_{15} = range(0, 0.3)		
Parameter	C_4 = UR5e-1 initial Rz rotation	C_5 = UR5e-1 initial Ry rotation	C_6 = Ramp initial Z position
Sub-curriculum	C_{41} = 0	C_{51} = 0	C_{61} = 0
	C_{42} =[-10°, 0, 10°]	C_{52} = [0, 10°, 20°]	C_{62} = [-0.2, -0.1, ..., 0.02]
	C_{42} = [-30°, -20°, ..., 30°]	C_{52} = [0, 10°,..., 50°]	C_{63} = range(-0.2, 0.2)
	C_{43} = [-60°, -50°, ..., 60°]	C_{53} = range(0, 90°)	
	C_{44} = range(-90°, 90°)		
Parameter	C_7 = UR5e-2 moving requirement		
Sub-curriculum	C_{71} = No movement required		
	C_{72} = From offset [-0.1, 0, 0.2] move to the screw point		

Table 3-5. The final curriculum stage of Task 1

Final curriculum stage of Task 1	Combination of sub-curriculum
Stage 1	C_{11} , C_{21} , C_{31} , C_{41} , C_{51} , C_{61} , C_{71}
Stage 2	$\boldsymbol{C_{12}}$, C_{21} , C_{31} , C_{41} , C_{51} , $\boldsymbol{C_{62}}$, C_{71}
Stage 3	$\boldsymbol{C_{13}}$, $\boldsymbol{C_{22}}$, $\boldsymbol{C_{4\text{-}11}}$, C_{41} , C_{51} , C_{62} , C_{71}
Stage 4	$\boldsymbol{C_{14}}$, $\boldsymbol{C_{23}}$, $\boldsymbol{C_{33}}$, C_{41} , C_{51} , C_{62} , C_{71}
Stage 5	C_{14} , C_{23} , C_{33} , $\boldsymbol{C_{42}}$, C_{51} , C_{62} , $\boldsymbol{C_{72}}$
Stage 6	C_{14} , C_{23} , C_{33} , $\boldsymbol{C_{43}}$, $\boldsymbol{C_{52}}$, C_{62} , C_{72}
Stage 7	$\boldsymbol{C_{15}}$, $\boldsymbol{C_{24}}$, $\boldsymbol{C_{34}}$, C_{43} , C_{52} , C_{63} , C_{72}
Stage 8	C_{15} , C_{24} , C_{34} , $\boldsymbol{C_{44}}$, $\boldsymbol{C_{53}}$, C_{63} , C_{72}

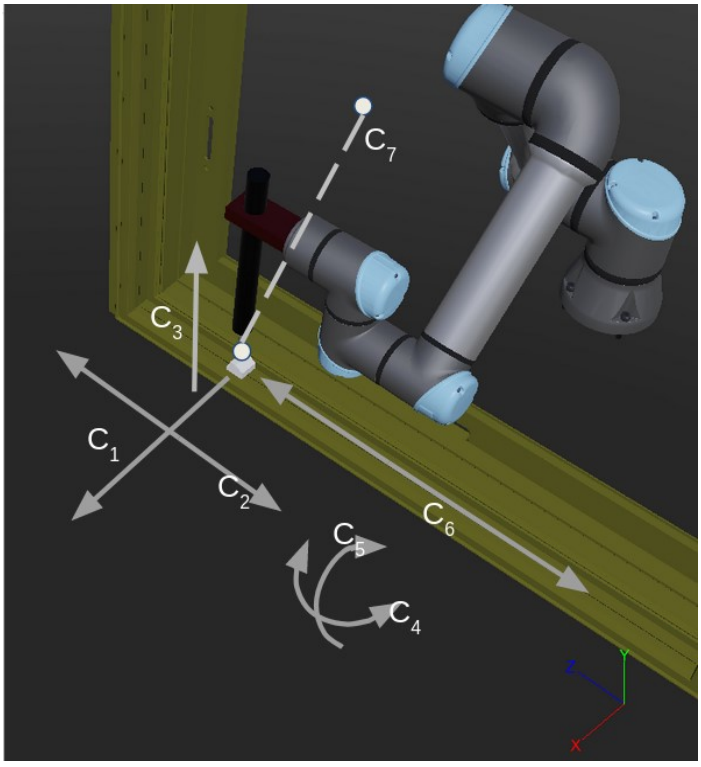


Figure 23. Parameters of Task 1

After the experience of a single hardware task, an attempt was made to implement the learning from multiple hardware tasks to a single agent. Task 2's objective was to train an agent to guide UR5e-1 to locate a ramp, a hinge track, and an operator handling point, respectively. The top task layer number became 3 as there were three different hardware components as the top tasks in Task 2. Tables 3-6 and 3-7 show the curriculum sub-tree of the top tasks, hinge track and operator. These sub-trees were combined with the top task ramp to form a complete curriculum tree. With one more layer added, one more dimension of parameters is created as $C_{ij,}$ and the sub-curriculum is presented as C_{ijk}. A geometric illustration of the parameters of Task 2 is shown in Figures 24 and 25. The final curriculum stage is introduced in Table 3-6.

Table 3-6. The combined curriculum tree for Task 2

Objective	Find hinge track, operator, and ramp handle point		
Top layer task	C_1 = Find hinge track handling point		
Parameter	C_{11} = UR5e-1 initial Y position from hinge track	C_{12} = UR5e-1 initial X position from hinge track	C_{13} = UR5e-1 initial Z position from hinge track
Sub-curriculum	C_{111} = 0.03	C_{121} = 0	C_{131} = 0
	C_{112} = [0, 0.01, 0.02, 0.03]	C_{122} = [-0.03, -0.02, ..., 0.03]	C_{132} = [-0.03, -0.02, ..., 0.03]
	C_{113} = [0, 0.01, 0.02, ..., 0.1]	C_{123} = [-0.08, -0.07, ..., 0.08]	C_{133} = [-0.08, -0.07, ..., 0.08]
	C_{114} = [0, 0.01, 0.02, ..., 0.2]	C_{124} = range (-0.15, 0.15)	C_{134} = range (-0.15, 0.15)
	C_{115} = range (0, 0.3)		

Parameter	C_{14} = UR5e-1 initial Rx rotation	C_{15} = UR5e-1 initial Rz rotation	C_{16} = UR5e-2 initial screw point
Sub-curriculum	C_{141} = 0	C_{151} = 0	C_{161} = 3
	C_{142} =[-10º, 0, 10º]	C_{152} =[0, 10º, 20º]	C_{162} = 3, 4
	C_{143} = [-30º, -20º, ..., 30º]	C_{153} = [0, 10º,..., 50º]	C_{163} = 1, 2, 3, 4
	C_{144} = [-60º, -50º, ..., 60º]	C_{154} = range (0, 90º)	C_{164} = 1, 2, 3, 4, 5, 6
	C_{145} = range (-90º, 90º)		C_{165} = random (1, 2, 3)→random(1, 2, 3), random(4, 5, 6)→random(4, 5, 6)
			C_{166} = random (1, 2, 3, 4, 5, 6)→random (1, 2, 3, 4, 5, 6)
Parameter	C_{17} = UR5e-2 pass outer space or not		
Sub-curriculum	C_{171} = No		
	C_{172} = Stay at a point with offset [range (-0.1,-0.1), range (0.05,0.35), range (-0.1,0.1)]		
	C_{173} = Move from a point with offset [range (-0.1,-0.1), range (0.05,0.35), range (-0.1,0.1)]		
Top layer task	C_2 = Find operator handling point		
Parameter	C_{21} = UR5e-1 initial Y position from hinge track	C_{22} = UR5e-1 initial X position from hinge track	C_{23} = UR5e-1 initial Z position from hinge track

Sub-curriculum	C_{211} = 0.03	C_{221} = 0	C_{231} = 0
	C_{212} = [0, 0.01, 0.02, 0.03]	C_{222} = [-0.03, -0.02, ..., 0.03]	$C_{24\text{-}11}$ = [-0.03, -0.02, ..., 0.03]
	C_{213} = [0, 0.01, 0.02, ..., 0.1]	C_{223} = [-0.08, -0.07, ..., 0.08]	C_{233} = [-0.08, -0.07, ..., 0.08]
	C_{214} = [0, 0.01, 0.02, ..., 0.2]	C_{224} = range (-0.15, 0.15)	C_{234} = range (-0.15, 0.15)
	C_{215} = range (0, 0.3)		
Parameter	C_{24} = UR5e-1 initial Rx rotation	C_{25} = UR5e-1 initial Rz rotation	C_{26} = UR5e-2 initial screw point
Sub-curriculum	C_{241} = 0	C_{251} = 0	C_{261} = 2,3
	C_{242} =[-10°, 0, 10°]	C_{252} =[0, 10°, 20°]	C_{262} = 1, 2, 3
	C_{243} = [-30°, -20°, ..., 30°]	C_{253} = [0, 10°,..., 50°]	C_{263} = 1, 2, 3, 4
	C_{244} = [-60°, -50°, ..., 60°]	C_{254} = range (0, 90°)	
	C_{245} = range (-90°, 90°)		
Parameter	C_{27} = UR5e-2 initial screw point Z offset	C_{28} = Hinge track initial Z position	C_{29} = UR5e-2 pass outer space or not
Sub-curriculum	C_{271} = 0	C_{281} = 0	C_{291} = No
	C_{272} = [-0.01, 0, 0.01]	C_{282} = [-0.02, -0.01, ..., 0.01]	C_{293} = Stay at a point with offset [-0.1,0.1,0]
	C_{273} = [-0.03, -0.02, ..., 0.03]	C_{283} = range (-0.02, 0.01)	C_{293} = Stay at a point with offset [range (-0.1,-0.1), range (0.05,0.35), range (-0.1,0.1)]
	C_{282} = range (-0.03, 0.03)		C_{294} = Move from a point with offset [range (-0.1,-0.1),

			range (0.05,0.35), range (-0.1,0.1)]
Top layer task	C_3 = Locate ramp handling point		
Parameter	C_{31} = UR5e-1 initial X distance from ramp	$C_{4\text{-}11}$ = UR5e-1 initial Y distance from ramp	C_{33} = UR5e-1 initial Z distance from ramp
Sub-curriculum	C_{311} = 0.03	$C_{4\text{-}111}$ = 0	C_{331} = 0
	C_{312} = [0, 0.01, 0.02, 0.03]	$C_{4\text{-}112}$ = [-0.02, -0.01, ..., 0.04]	$C_{34\text{-}11}$ = [-0.03, -0.02, ..., 0.03]
	C_{313} = [0, 0.01, 0.02, ..., 0.1]	$C_{4\text{-}113}$ = [0, 0.01, 0.02, ..., 0.16]	C_{333} = [-0.08, -0.07, ..., 0.08]
	C_{314} = [0, 0.01, 0.02, ..., 0.2]	$C_{4\text{-}114}$ = range (0, 0.3)	C_{334} = range (-0.15, 0.15)
	C_{315} = range (0, 0.3)		
Parameter	C_{34} = UR5e-1 initial Rz rotation	C_{35} = UR5e-1 initial Ry rotation	C_{36} = Ramp initial Z position
Sub-curriculum	C_{341} = 0	C_{351} = 0	C_{361} = 0
	C_{342} = [-10°, 0, 10°]	C_{352} = [0, 10°, 20°]	C_{362} = [-0.2, -0.1, ..., 0.02]
	C_{342} = [-30°, -20°, ..., 30°]	C_{352} = [0, 10°,..., 50°]	C_{363} = range(-0.2, 0.2)
	C_{343} = [-60°, -50°, ..., 60°]	C_{353} = range(0, 90°)	
	C_{344} = range(-90°, 90°)		
Parameter	C_{37} = UR5e-2 moving requirement		
Sub-curriculum	C_{371} = No movement required		
	C_{372} = From offset		

	[-0.1, 0, 0.2] move to the screw point		

Table 3-7. The final curriculum stage of Task 2

Final curriculum stage of Task 1	Combination of sub-curriculum
Stage 1	C_{111} , C_{121} , C_{131} , C_{141} , C_{151} , C_{161} , C_{171} , C_{211} , C_{221} , C_{231} , C_{241} , C_{251} , C_{261} , C_{271} , C_{281} , C_{291}
Stage 2	C_{111} , $\mathbf{C_{122}}$, $\mathbf{C_{14\text{-}11}}$, C_{141} , C_{151} , $\mathbf{C_{162}}$, C_{171} , C_{211} , $\mathbf{C_{222}}$, $\mathbf{C_{24\text{-}11}}$, C_{241} , C_{251} , C_{261} , $\mathbf{C_{272}}$, $\mathbf{C_{282}}$, C_{291}
Stage 3	$\mathbf{C_{112}}$, $\mathbf{C_{123}}$, $C_{14\text{-}11}$, C_{141} , C_{151} , $\mathbf{C_{163}}$, $\mathbf{C_{172}}$, $\mathbf{C_{212}}$, $\mathbf{C_{223}}$, $C_{24\text{-}11}$, C_{241} , C_{251} , $\mathbf{C_{262}}$, C_{272} , C_{282} , $\mathbf{C_{292}}$
Stage 4	$\mathbf{C_{113}}$, C_{123} , $\mathbf{C_{133}}$, $\mathbf{C_{142}}$, $\mathbf{C_{152}}$, $\mathbf{C_{164}}$, C_{172} , $\mathbf{C_{213}}$, C_{223} , $\mathbf{C_{233}}$, $\mathbf{C_{242}}$, $\mathbf{C_{252}}$, C_{262} , $\mathbf{C_{273}}$, C_{282} , C_{291}
Stage 5	$\mathbf{C_{114}}$, C_{123} , C_{133} , $\mathbf{C_{143}}$, C_{152} , $\mathbf{C_{165}}$, C_{172} , $\mathbf{C_{214}}$, C_{223} , C_{233} , $\mathbf{C_{243}}$, C_{252} , C_{262} , C_{273} , C_{282} , $\mathbf{C_{293}}$
Stage 6	C_{114} , C_{123} , C_{133} , C_{143} , C_{152} , C_{165} , C_{172} , C_{214} , C_{223} , C_{233} , C_{243} , C_{252} , C_{262} , C_{273} , C_{282} , C_{293} $\mathbf{C_{311}}$, $\mathbf{C_{4\text{-}111}}$, $\mathbf{C_{331}}$, $\mathbf{C_{341}}$, $\mathbf{C_{351}}$, $\mathbf{C_{361}}$, $\mathbf{C_{371}}$
Stage 7	C_{114} , C_{123} , C_{133} , C_{143} , C_{152} , C_{165} , C_{172} , C_{214} , C_{223} , C_{233} , C_{243} , C_{252} , C_{262} , C_{273} , C_{282} , C_{293} $\mathbf{C_{312}}$, $C_{4\text{-}111}$, $\mathbf{C_{34\text{-}11}}$, C_{341} , C_{351} , C_{361} , C_{371} Failed to merge harder ramp sub-curriculums from here, so the ramp task was abandoned in favour of returning to Stage 6.
Stage 8 (modified stage 6)	$\mathbf{C_{115}}$, C_{123} , C_{133} , C_{143} , C_{152} , C_{165} , C_{172} , $\mathbf{C_{215}}$, C_{223} , C_{233} , C_{243} , C_{252} , $\mathbf{C_{263}}$, $\mathbf{C_{274}}$, C_{282} , C_{293}
Stage 9	C_{115} , $\mathbf{C_{124}}$, $\mathbf{C_{134}}$, $\mathbf{C_{144}}$, C_{153} , C_{165} , C_{172} , C_{215} , $\mathbf{C_{224}}$, $\mathbf{C_{234}}$, $\mathbf{C_{244}}$, C_{253} , C_{263} , C_{274} , C_{282} , C_{293}
Stage 10	C_{115} , C_{124} , C_{134} , C_{144} , $\mathbf{C_{153}}$, $\mathbf{C_{166}}$, C_{172} , C_{215} , C_{224} , C_{234} , C_{244} , $\mathbf{C_{253}}$, C_{263} , C_{274} , $\mathbf{C_{283}}$, C_{293}
Stage 11	C_{115} , C_{124} , C_{134} , C_{144} , C_{152} , C_{166} , $\mathbf{C_{173}}$, C_{215} , C_{224} , C_{234} , C_{244} , C_{252} , C_{263} , C_{274} , C_{283} , $\mathbf{C_{294}}$

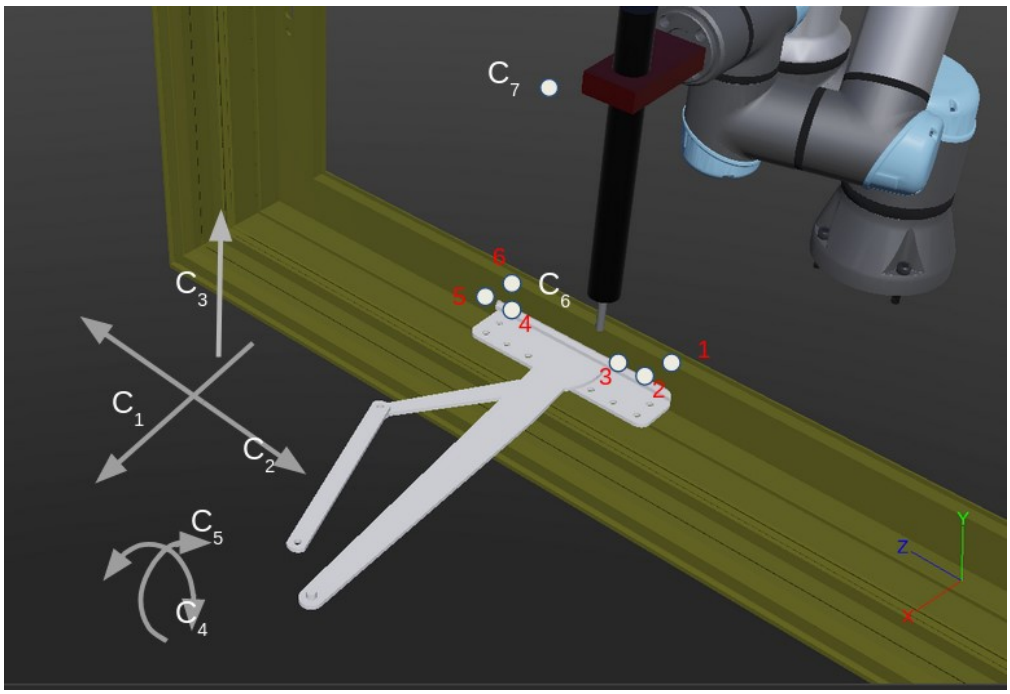


Figure 24. Parameters of the top task C_1 for Task 2

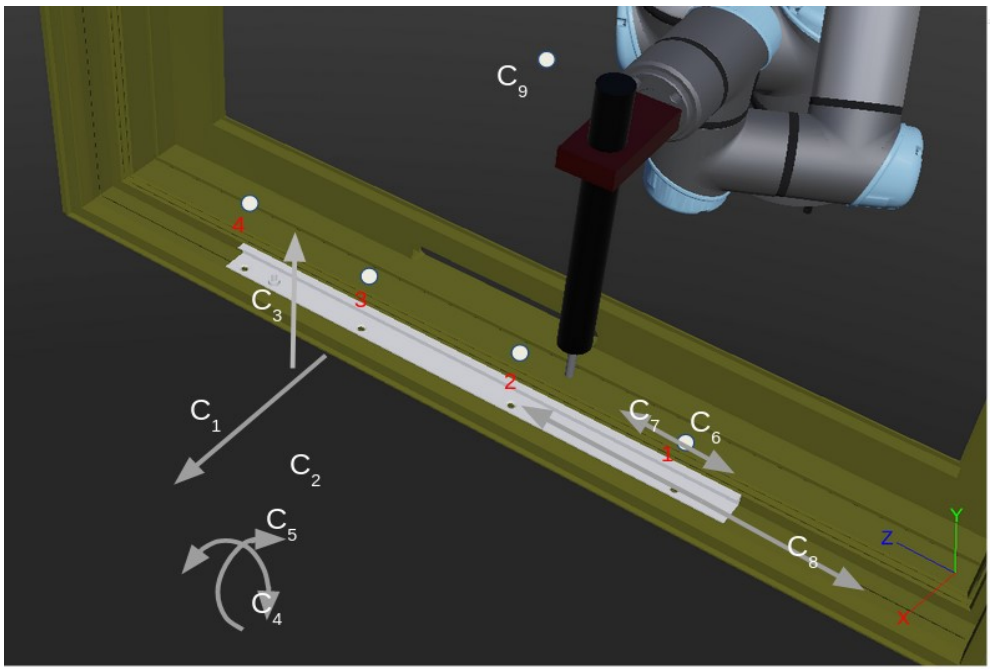


Figure 25. Parameters of the top task C_2 for Task 2

4 EXPERIMENTS

4.1 Reward shaping results

Successful model training requires a proper incentive for the agent to explore possibilities for a solution. After defining the environment, a reward shaping technique was introduced to improve the performance and success rate. As described in Section 3.5.2, three versions of the reward function were designed within the same environment.

- The first version of reward function R_1: The basic hardware approach version
- The second version of reward function R_2: Fail and success reward flattened and distance emphasized version
- The third version of reward function R_3: Velocity slow down and motor efficiency function added version).

The environment was defined such that the end-effector of UR5e-1 was set to the location from the hardware [0.06, 0.03, 0] and the sucker pointed at the hardware surface perpendicularly without rotation difference. Figure 26 shows the success rate every 100 episodes using the first version reward function R_1. Although the agent was able to reach its goal, it was not stable in terms of a gradual success rate improvement, which indicated a poor training flow. The graph shows no positive relation between the y-axis (success rate) and x-axis (episodes) under the first reward version, illustrating that the extreme success rate values from the first reward function prevented the agent from finding a suitable approach. The “jumping” extreme values were caused by the large difference in reward given by success and failure trials. The reward difference was evident over a broad range of episodes. The agent also lost incentive in terms of

trying to reach the hardware if no penalty was given at the maximum episode step because it was punished by collision more often than getting rewarded by successful trial.

Figure 27 shows the success rate every 100 episodes using the second version reward function R_2. The more extreme success rate after a successful trial was able to be solved by flattening the sum of reward. The success rate improved steadily after the first success; however, the model failed to surpass the 0.55 success rate even after the learning rate was reduced. Further modification was needed in order to reach the expected 0.95 success rate.

In the third version of the reward function R_3, the velocity of the end-effectors and the motor efficiency were adjusted as described in Section 3.5.1. The modified model reduced the end-effectors' velocity when it started to reach a proximate distance to the hardware. Moreover, the end-effector took a smoother path with less rotational time, which could reduce energy costs in a real-world manipulator implementation. Figure 28 shows an early increase in the success rate with a more continuous uptrend towards the 1.0 success rate. This result proved that the reward shaping modification did not only improve the performance of the agent, but also implemented a predictable change in the agent's behaviour by specific reward modification given by the developer.

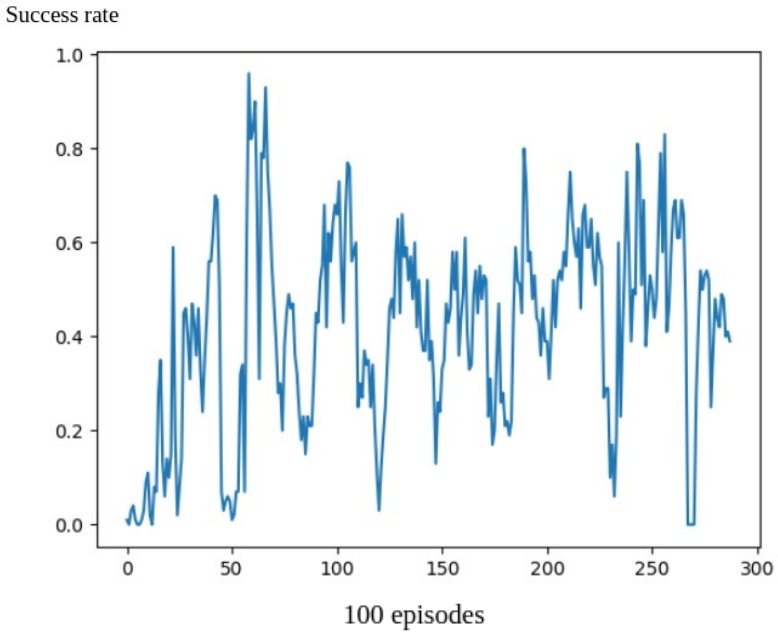


Figure 26. Training result for the first version of reward function R_1

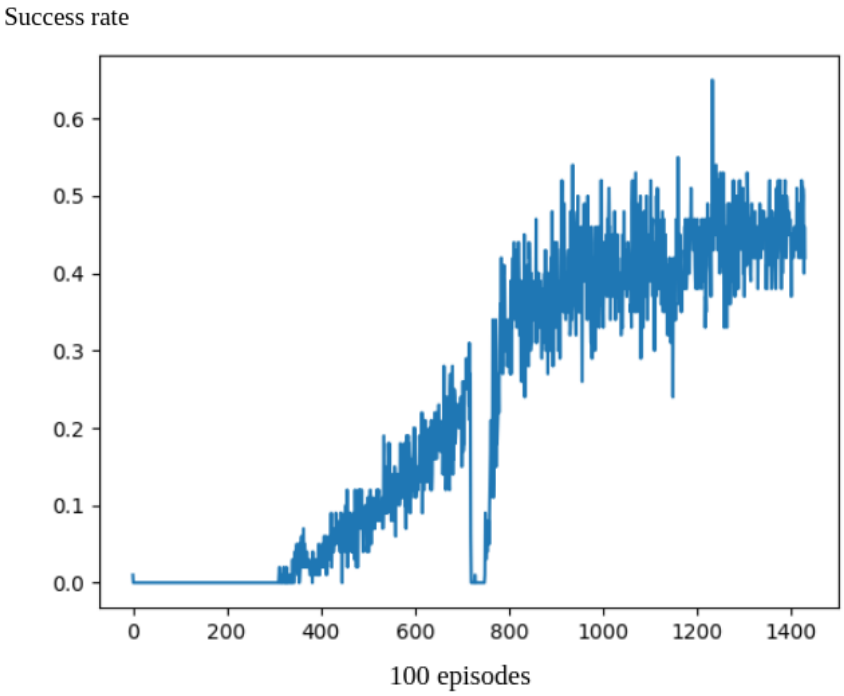


Figure 27. Training result for the second version of reward function R_2

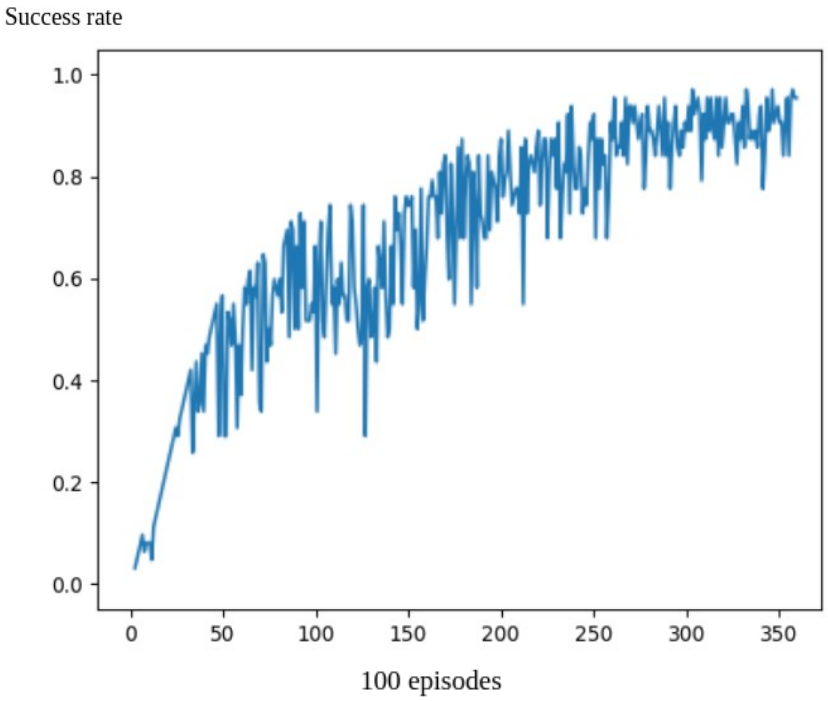


Figure 28. Training result by the third version of reward function R_3

4.2 Parameter tuning experiment

The parameters were then tuned with the intent to find a hyperparameter for this case. The parameter tuning trial was completed with six different setups in terms of model size under the same environment described in Section 4.1 for two layers and three layers of neurons, i.e., 64 × 64, 256 × 256, 400 × 300, 64 × 64 × 64, and 256 × 256 × 256. Table 4-1 shows the results of all setups on which the experiments were conducted. Based on the results, it was determined that the 400 × 300 network was the optimal model size for this case because it contained a sufficient amount of neurons to reach the goal without costing too much in terms of time to update. This balance provided a favorable environment for the agent to train efficiently without overfitting to the dataset.

The learning rate can be adjusted during the training; however, the effect of modifying the coefficient tau was negligible. The approach we took was to instead modify the learning rate as mentioned in Section 3.5.2. The results suggested that an effective improvement can be achieved at the end of the training process by modifying the learning rate.

Table 4-1. Experiment results under different model size setup

Model size (neurons)	Number of episodes required to achieve 30% success rate	Time needed to achieve 30% success rate (mins)	Episode-to-time needed ratio
64 × 64	36394±2912	183.12±10.12	198.74
256 × 256	10982±1117	12.17±1.61	902.38
400 × 300	2429±205	2.38 ±0.19	1020.59
512 × 512	2121±181	2.88±0.29	736.91
64 × 64 × 64	1512±185	3.09±0.35	488.27
256 × 256 × 256	1269±149	2.25±0.15	563.14

4.3 Task 1: Single hardware curriculum learning

With respect to curriculum task design, as described in Section 3.6.2, the final goal of Task 1 was to guide the UR5e-1 end-effector from its initial state within a range of 30 cm × 30 cm × 30 cm box range and arbitrary rotation to reach the target hardware, while the agent had to determine whether to move UR5e-2 or not according to its initial state. First, Task 1 is trained without curriculum learning. If the model was trained directly with the final goal of Task 1, the model would have had a dramatically lower chance of achieving its first success. Even if the

model were able to perform a successful episode, the experience would not be able to be replicated. The performance even worsened in some scenarios.

From the success rates of Task 1 without curriculum training as shown in Figure 29, the success rate was close to zero even after 80,000 episodes of training. The reason for this was explored through performance measurement by the sum of reward. The reward flow in Figure 30 shows that the agent failed to find a successful path.

Loss flow were logged to compare with the reward flow in order to ensure the training progress is normal. Actor loss represents the model's inability to perform a rewardable action. Critic loss stands for the model's inability for the agent to distinguish between good performance and bad. Figure 31 shows a downtrend of critic loss while the actor loss and reward procedure did not improve. Such a phenomenon illustrates that the critic network was unable to define a good action or approach. The model's size and learning rate were adjusted, and, unfortunately, it resulted in a similar failing outcome. It was concluded, from the loss function logs, that it was highly possible that it was due to a low success frequency, resulting in a low learning efficiency. Hence, curricular reinforcement learning was necessary to aid the agent to learn from milestones in order to reach the final goal. In this case, we implement curricular learning concepts and use CGRLS to train the agent from simple to hard.

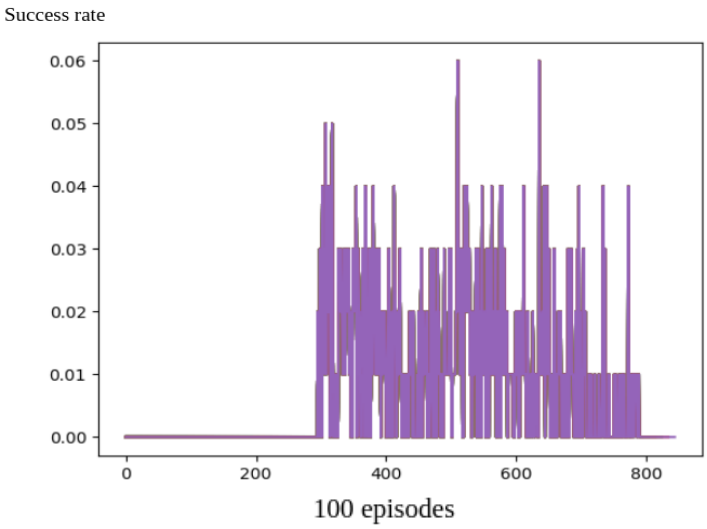


Figure 29. Success rate of Task 1 without curriculum learning

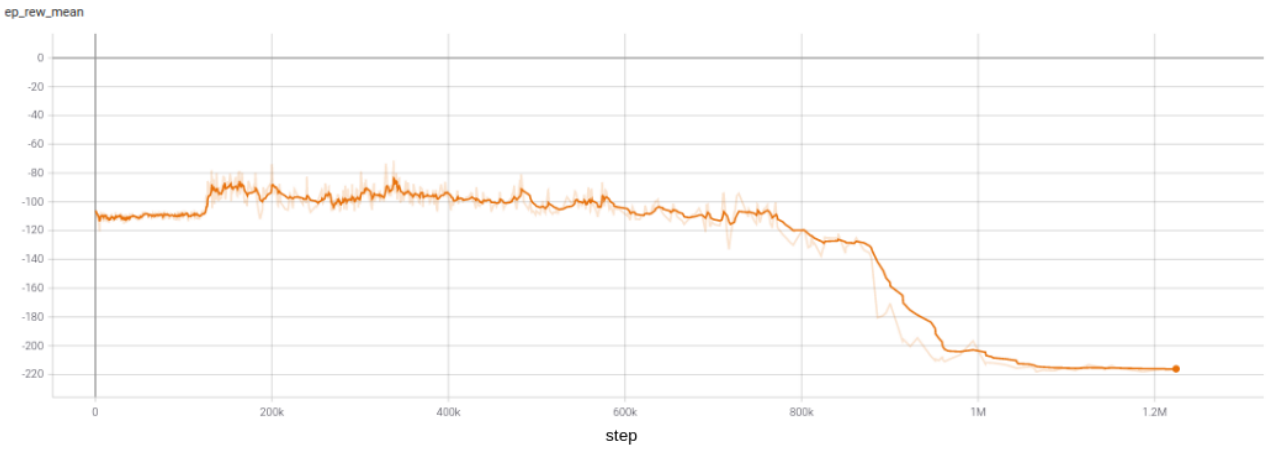


Figure 30. Mean reward of Task 1 without curriculum learning

Figure 31. Actor and critic loss of Task 1 without curriculum learning

Under the curriculum tree shown in Tables 3-4 and 3-5, which were created using the CGRLS methodology mentioned in Section 3.6.1, curriculum Stage 1 was designed to be a stage where the UR5e-1 will be able to point at the ramp hardware without rotation variation. Then curriculum Stage 2 was designed to be a little bit more complexed: the initial distance along the x-axis from the UR5e-1 to the ramp was changed from a fixated $C_{11} = 0.03$ to a randomized pick from $C_{12} = [0, 0.01, 0.02, 0.03]$, while the ramp's initial Z position was changed from a constant $C_{61} = 0$ to a pick from $C_{62} = [-0.2, -0.1, ..., 0.02]$. After that, the distance between UR5e-1 and the ramp along all three axes was increased, which expanded the range of distance along the x-axis throughout Stages 3 and 4. Stage 5 introduced a movement requirement for the UR5e-2 to locate the ramp within an offset randomly picked from [-0.1, 0, 0.2] to the screw point. Then, the difficulty was further increased in Stage 6 by randomizing the initial Rz and Ry rotation of UR5e-1 so that they were randomly picked from $C_{42} = [-10^o, 0, 10^o]$ and $C_{52} = [0, 10^o, 20^o]$, respectively. Note that all parameters had been updated once; hence, the requirement for any parameter to advance to its final curriculum was fulfilled at the end of this stage. At Stage 7, we advanced C_1 to its final curriculum C_{15}, which means that the initial distance along the x-axis started to be a randomized value from the continuous range (0, 0.3). The same was done to $\boldsymbol{C_2}$, $\boldsymbol{C_3}$, and C_6 at the same stage. Finally, C_4 and C_5 were advanced to their final curriculum, finalizing the final goal of Task 1 such that the model was able to install the ramp hardware to the frame within a 30 × 30 × 30 box range and arbitrary rotation, no matter at what distance they initially were.

Figure 32 and Figure 33 show the episode reward averages as the model training proceeded under curriculum learning. It can be concluded that the agent was able to determine a suitable approach to tackle the assigned curriculum, as the reward average shows a steadily upward trend.

The actor loss and critic loss functions shown in Figure 34 states that the models' inability to perform rewardable and critical action was decreasing. In other words, the model learned to tackle the given task with critical decisions.

Finally, the success rate of Task 1 had a strong upward learning trend with a peak of around 98% success rate, as shown in Figure 35.

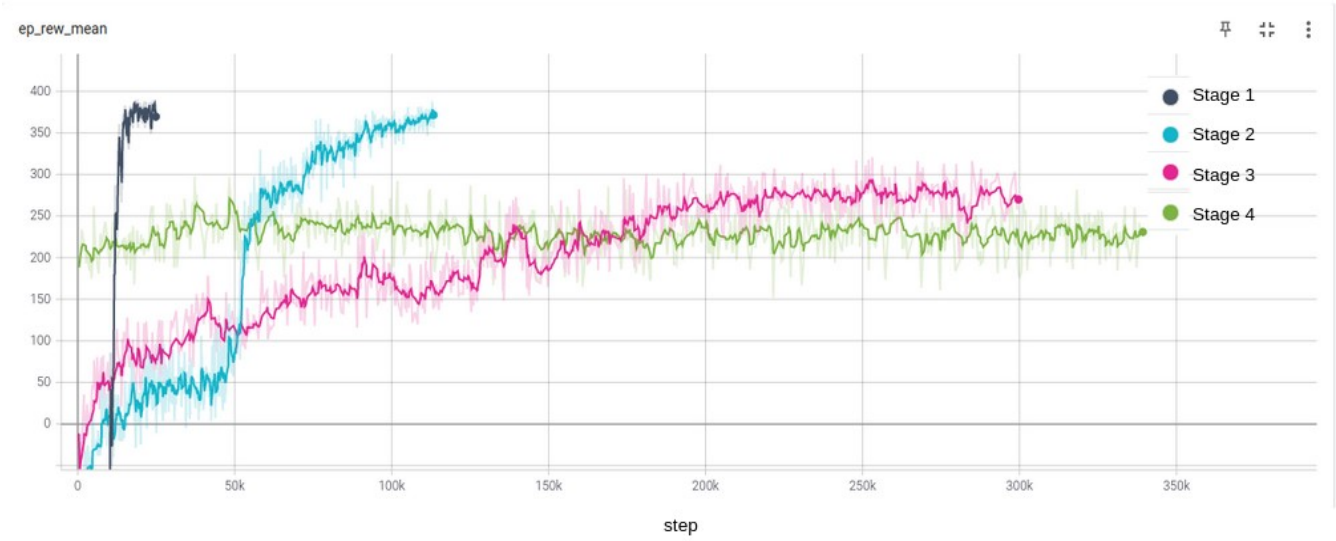


Figure 32. Mean reward of Task 1 with curriculum Stages 1–4

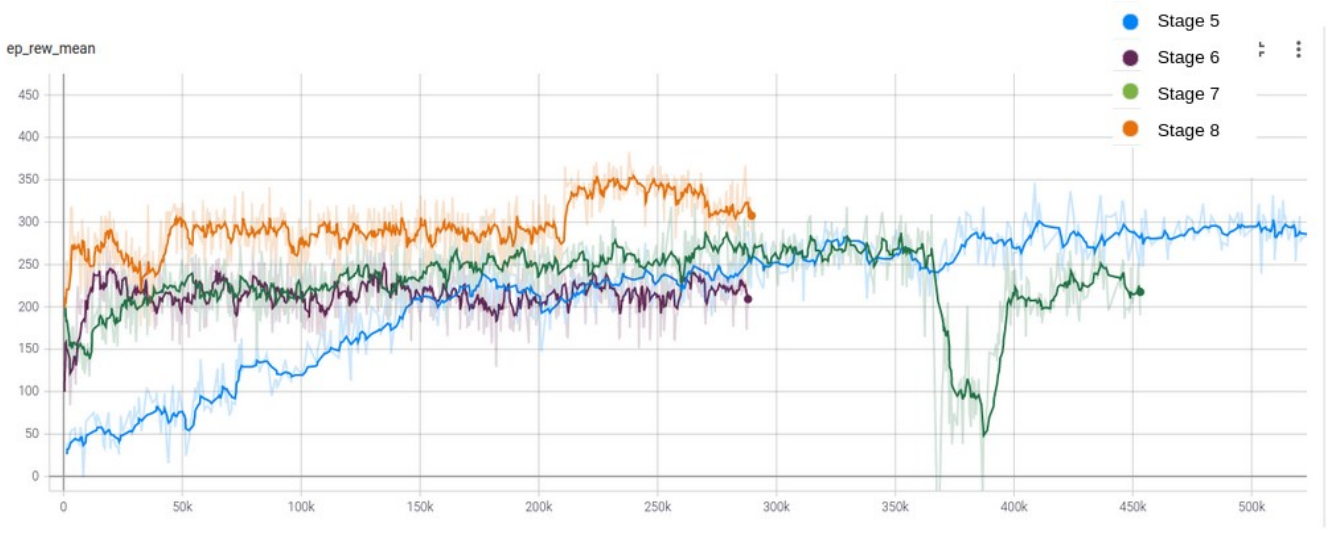


Figure 33. Mean reward of Task 1 with curriculum Stages 5–8

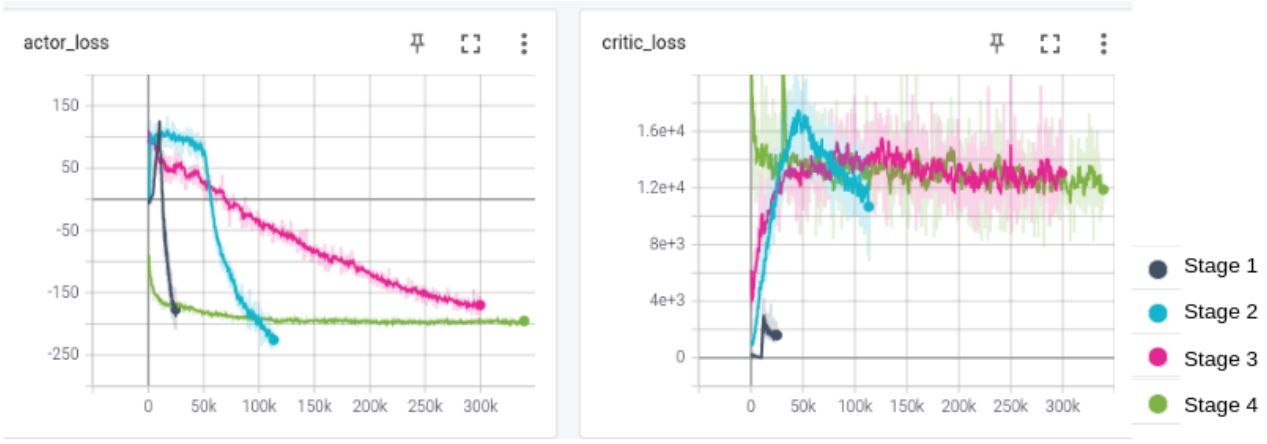


Figure 34. Actor and critic loss of Task 1 with curriculum Stages 1–4

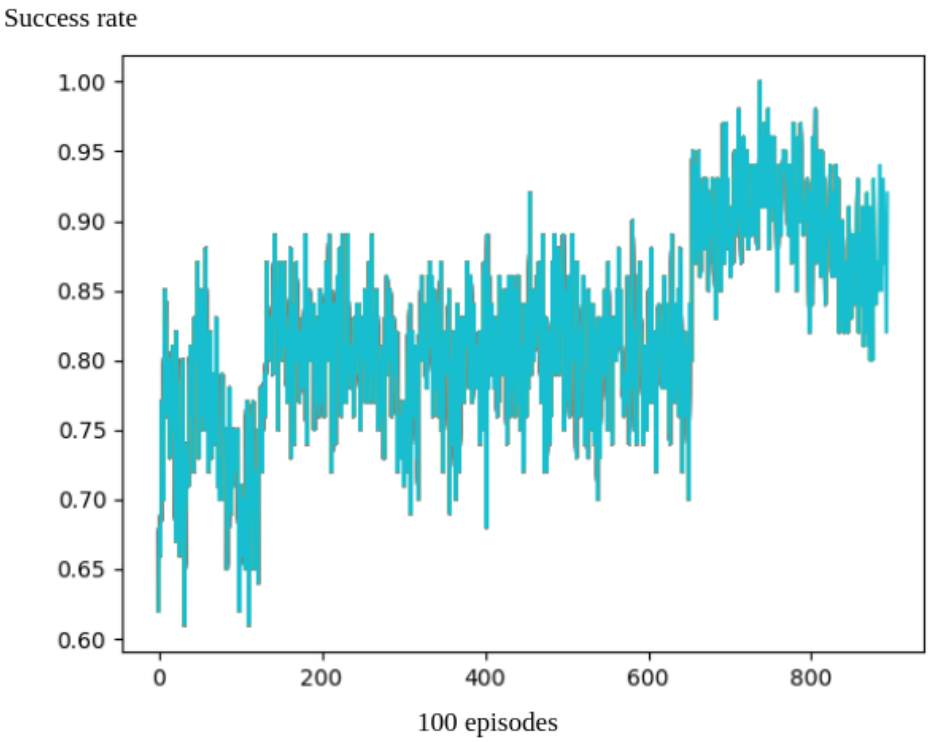


Figure 35. Success rate of Task 1 with curriculum Stage 8 (final)

4.4 Task 2: Multiple hardware curriculum learning

The successful Task 1 model was then used as the training agent for the multiple top task environments. The downward episode reward mean trend shown in Figure 36 indicates that the model had a low learning performance with the multiple top tasks environments. The results remained the same even after the sub-curriculum of the new tasks were set to their easiest difficulties. It could be that the agent overfit into the pre-existing experience, and the brand new curriculum forced the agent to change its parameter abruptly, causing poor performance on both the previous curriculum and newly-introduced curriculum. Thus, it was necessary to define all the objectives at the beginning and train every top task before any task was completed.

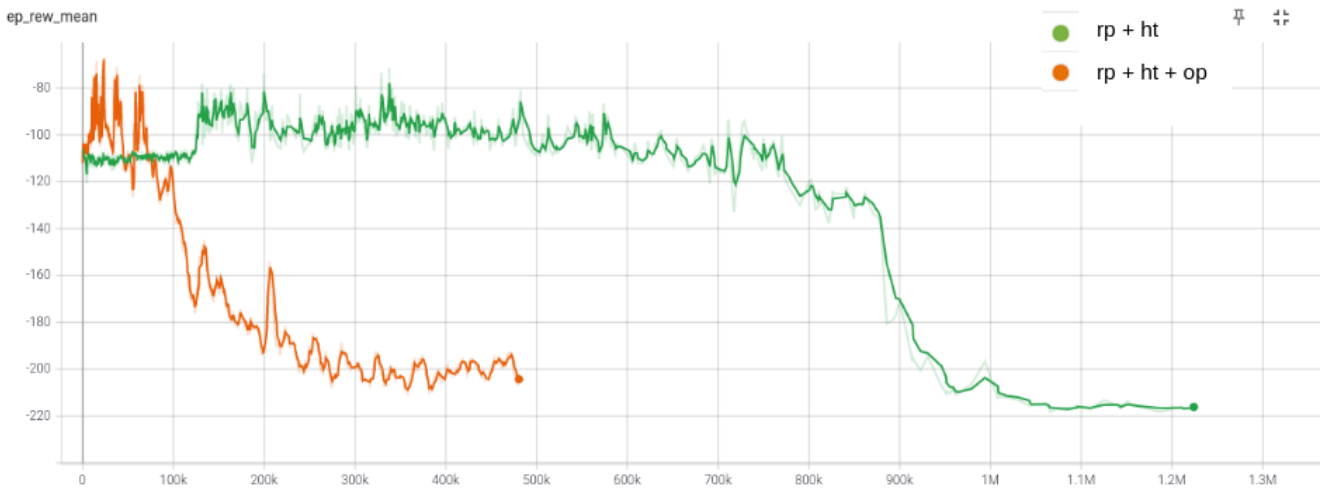


Figure 36. Mean reward of combined hardware environment using Task 1 model

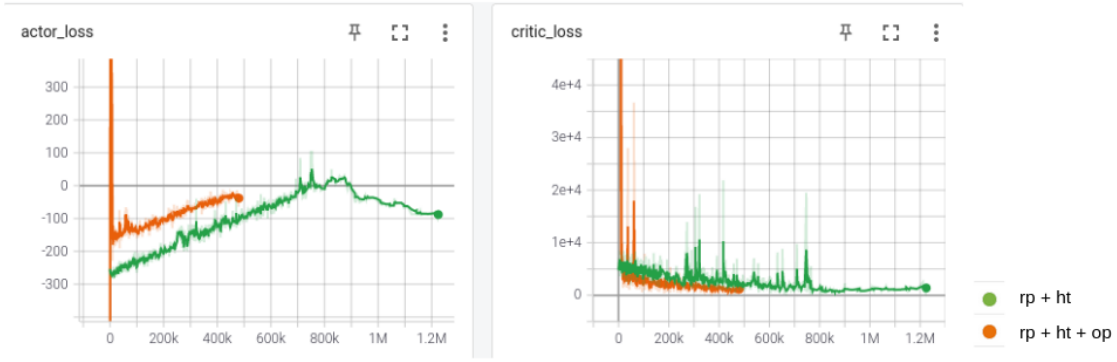


Figure 37. Actor and critic loss of combined hardware environment using Task 1 model

Based on the reason mentioned above, all the training of multiple top tasks must be started simultaneously under the rule outlined in Section 3.6.1. A new curriculum tree was then constructed with three tasks at the top layer as mentioned in Section 3.6.2. More curriculum stages were needed as more sub-curriculums were involved. However, the success rate shown in Figure 38 indicates a performance divergence between the hinge track/operator/ramp (HT/OP/RP) curricula. The divergence illustrates the difficulty the model experienced in terms of learning hardware components with different connector directions. The hinge track and

operator shared a similar connector direction, so that their corresponding curricula had identical trends of high success rate and learning pattern. Unfortunately, the RP curriculum was not able to be learned efficiently by the agent in the beginning.

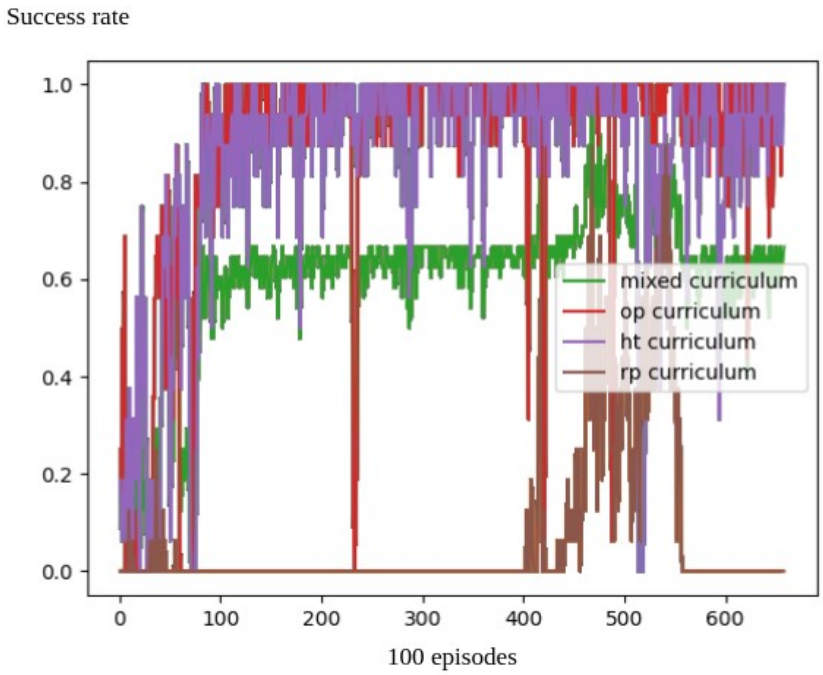


Figure 38. Success rate selecting all hardware appearing 1:1:1 using their first sub-curriculum

The approach used to solve this problem was to rearrange the HT and OP curricula to start training in early stages, then to introduce the RP curricula in later stages. Figure 39 and 41 show the success rate of the Stage 1, and Stages 1–5, respectively. Figure 40 shows the mean reward of curriculum Stage 1–5 with the HT and OP curricula being the focused tasks. The agent performed as well as expected in the early stages. Note that at the end of Stage 5, all parameters had reached their penultimate stages. The RP curriculum needed to be merged in order to continue the learning process; therefore, the RP curriculum was introduced into the agent's training at that point.

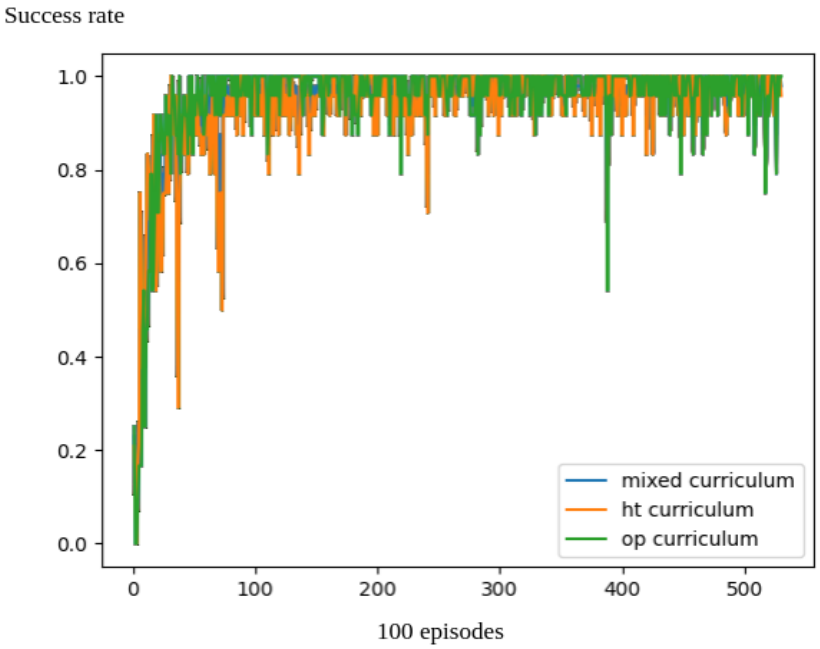


Figure 39. Curriculum Stage 1 of Task 2

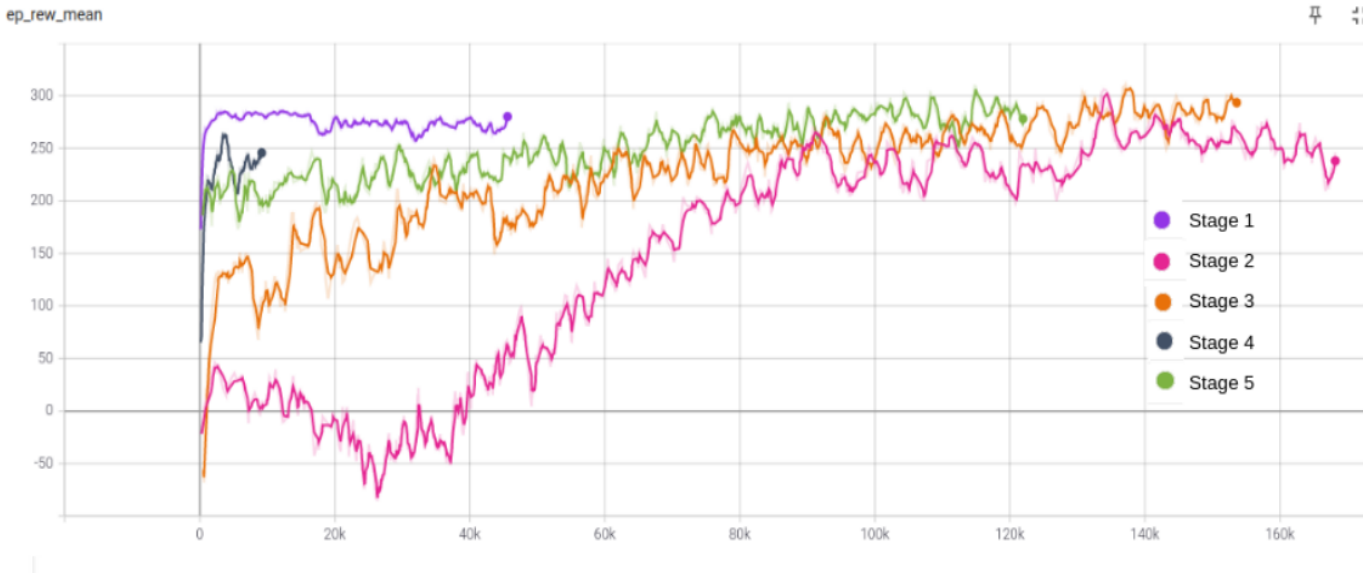


Figure 40. Mean reward of Task 2 with curriculum Stages 1–5

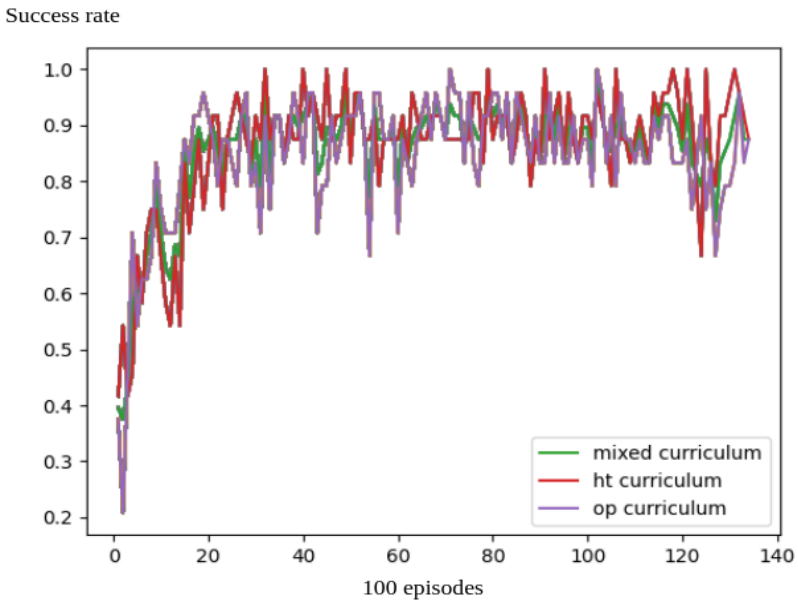


Figure 41. Curriculum Stage 5 of Task 2

The newly merged RP curriculum followed the rules mentioned in Section 3.6.1, which were for all its sub-curriculum to start from the easiest difficulties. The learning process was unstable and time-consuming.

Next, variations were experimented with in terms of the proportions between the HT/OP curricula that the agent had already started training and the RP curriculum that the agent had never seen before. Figure 42 shows results of different proportions of the RP curriculum being introduced in Stage 6.

Experiment results indicate that a large amount of the RP curriculum disturbed existing HT/OP curricula's knowledge, while a small amount of RP curriculum did not provide sufficient learning experience to the agent, causing sample inefficiency. It was determined that a 60% new-

curriculum-introduction proportion is optimal for the agent to focus on learning a slightly different curriculum before putting its original curricula into their final curriculum stage.

The performance measurements shown in Figure 43 show that the agent struggled to emerge from pre-existing knowledge to the newly-introduced curriculum at the beginning. The performance decreased during the implementation of Stage 7, where the distance was changed along the x-axis and the initial z-position were changed from constants to variables. It was concluded that the agent failed to adapt to the RP curriculum, and a decision was made to remove the RP curriculum from the top curriculum layer. Stage 8 was a modified version of Stage 6 with the absence of RP curriculum.

As shown in Figure 44, the agent's learning rate then improved rapidly at Stage 8 after the removal of the RP curriculum. Finally, the agent's success rate on Task 2 shown in Figure 42 reached nearly 100% on mixed curriculum at the end of its final curriculum stage.

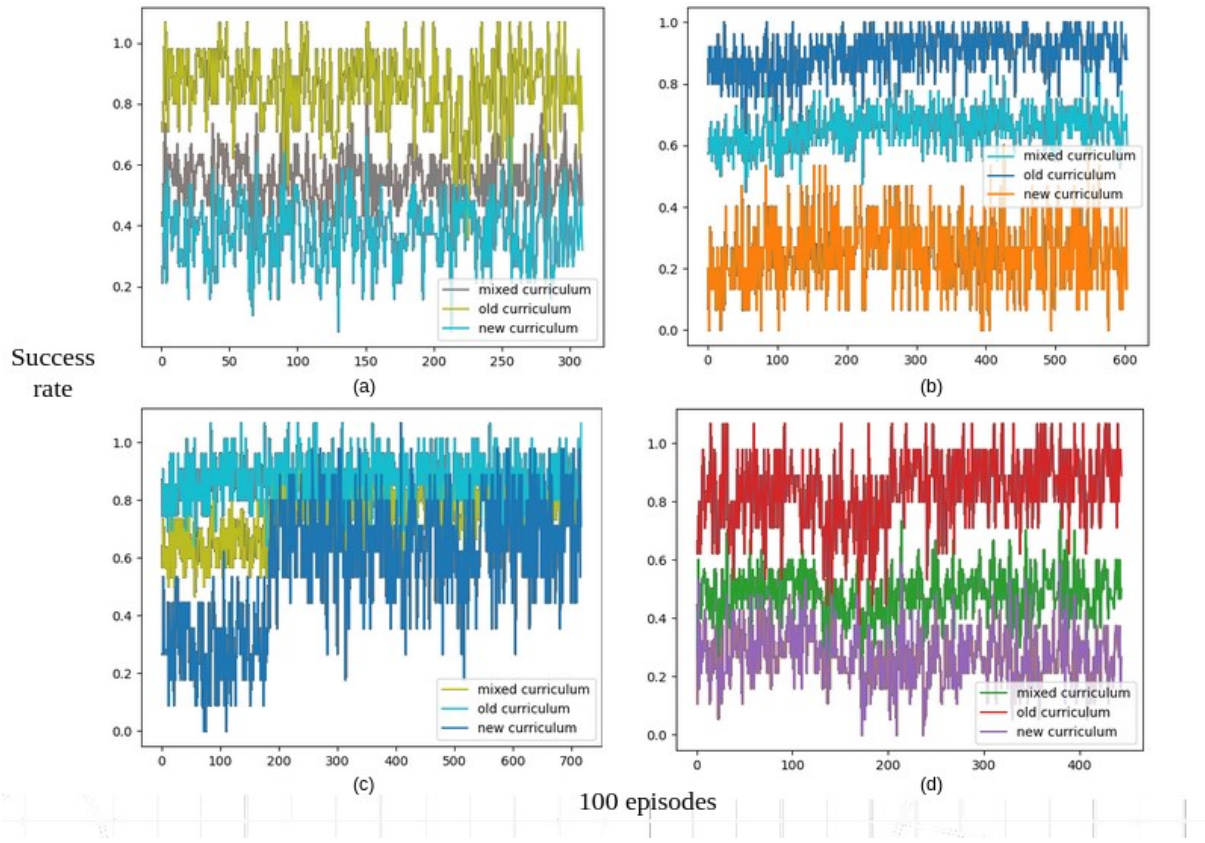


Figure 43. Success rate of Curriculum Stage 6 for Task 2 with new-curriculum-introduction proportion of (a) 20%, (b) 40%, (c) 60%, (d) 80%

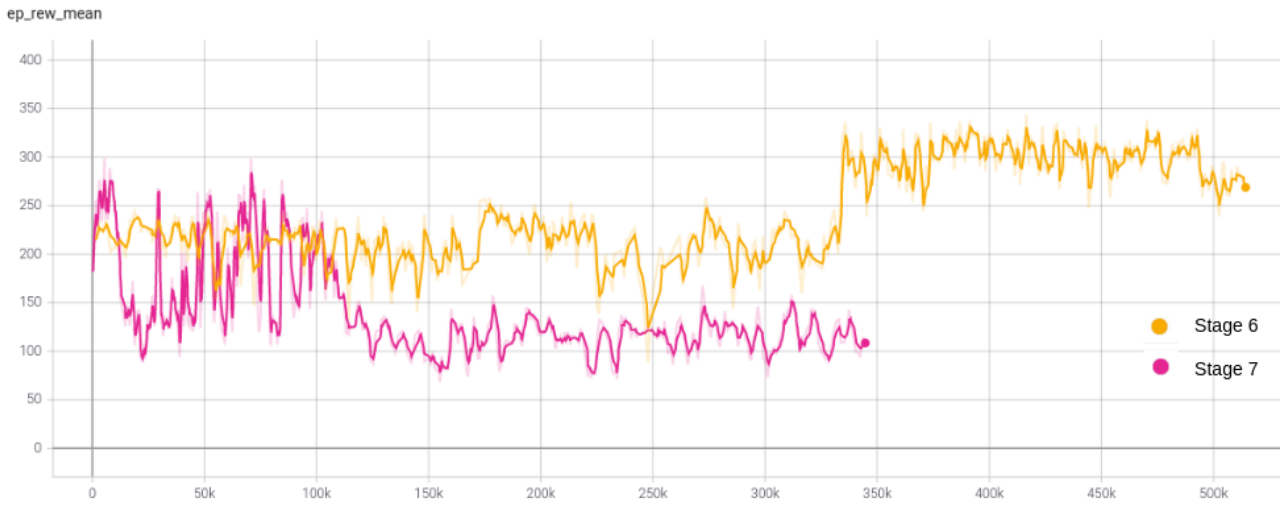


Figure 44. Mean reward of Task 2 with curriculum Stages 6, 7

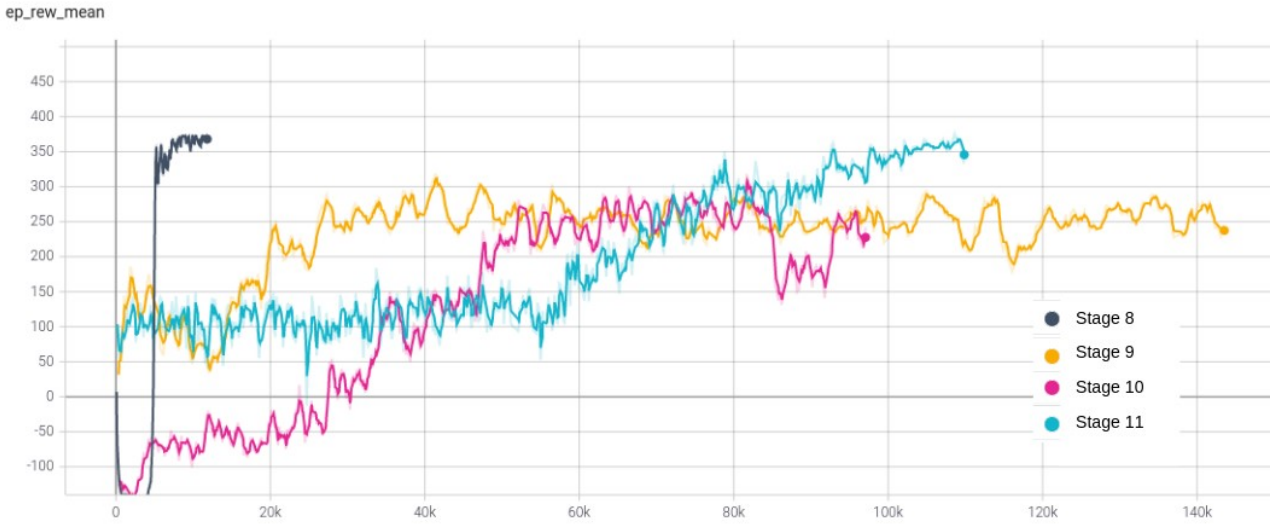


Figure 45. Mean reward of Task 2 with curriculum Stage 8–11

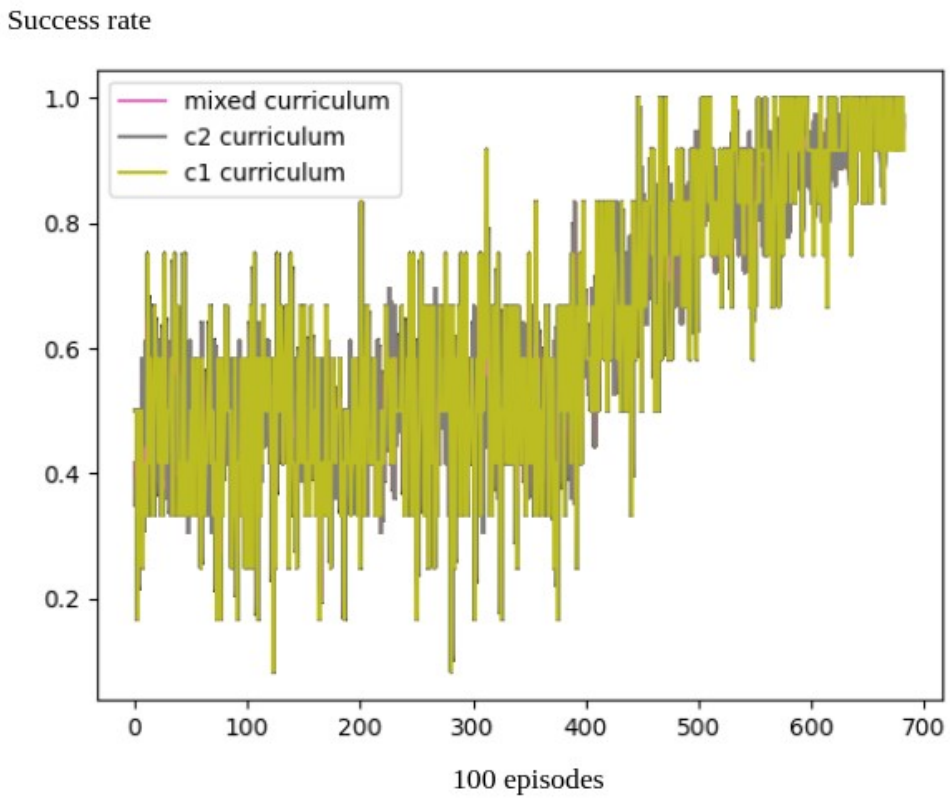


Figure 46. Success rate of Task 2 with curriculum Stage 11 (final)

Although the RP curriculum was absent in the final curriculum stage, the agent successfully learned to adapt to the change of target hardware and environment. In the final stage, the agent was able to guide the UR5e-2 from one screw point to another while guiding the UR5e-1 to go around if it was blocked by the UR5e-2. The stable and efficient performance presented by the model shows that complex hardware installation is able to be automated by robots using AI. With the application of reinforcement learning with CGRLS techniques, real-time adjustment of the model training is also possible, which indicates that robotic automation has huge potential in the manufacturing industry.

5 CONCLUSION

The purpose of the present research was to propose an automated hardware installation solution. The present research provided a case study which explored the automation of the hardware assembly process for window production. The “screw and handle” motion was identified as the most frequent action through task and hardware selection. However, the automation of such motions experienced a bottleneck due to the complexity of problems that could not be simply solved by rule-based robot motion planning. This was the motivation behind the adoption of reinforcement learning to accomplish the research objectives, and it was found to have great potential for a possible solution to the automation bottleneck.

5.1 Summary

The research was accomplished through simulating the window hardware installation process by exporting the processed parameters according to the order in the BIM model. An agent was created with the reinforcement learning OpenAI Gym structure to define an environment for the AI’s observation, action, and interaction.

Past experience provided justification in terms of applying techniques like reward function shaping and parameter tuning to improve the model’s performance. However, the performance was poor due to a common phenomenon in reinforcement learning called low sample efficiency. The low sample efficiency caused uncertainties and long learning time at the early research stage.

To tackle this issue, a powerful strategy was proposed, herein referred as CGRLS, based on the concept of curricular learning. The CGRLS is a curriculum system, a tree system that can train multiple tasks with the same nature. The proposed curriculum system improved the agent's efficiency and ability to learn.

In the experiment, two trials were executed, Task 1 and Task 2, where main tasks were assigned to the model to learn. The first task consisted of a single ramp installation curriculum. The second task consisted of a combined ramp, hinge track, and operator installation curricula. The first task was fully completed with great success, while the second task achieved only partial completion.

5.2 Research limitation

One limitation of the proposed method is that all task features must be trained evenly for an effective learning process. This condition greatly limits the method's flexibility and scalability. Furthermore, as the failure in Task 2, if the observation of one task is much different than others, the agent may need lots of training time and even fail to succeed.

5.3 Research contribution

This research provides a specific example using the case study of window hardware installation by AI in simulation. The definition and solution to the problem are defined, and a guided framework is proposed.

More specific contributions are as follows:

- Provided an automated robotic solution to automate the pick-and-place and handle-and-screw motion to offer manufacturers and researchers with a framework in terms of approaching production line automation.
- Addressed the design of reward function shaping and analyzed performance parameters of the reinforcement learning agent.
- Proposed a powerful curriculum guided reinforcement learning structure (CGRLS) to design a smoother learning pathway with curriculum stages for the agent in reinforcement learning.

5.4 Future work

Although the model's current outcome is satisfactory, the framework has not reached its full potential yet. Future efforts could create a more efficient model for full automation of the window hardware installation process. Future expansion of the model to other hardware parts and even to the sash of a window is expected as the model was designed to be scalable.

Although the proposed CGRLS method can effectively boost the agent's training efficiency and yield a better training result, it lacks the ability to transfer the model's knowledge to another similar task with slightly different properties under a multi-task condition. In the future, a more robust automatic hardware installation agent model could be provided via the combination of the method and the theory of policy distillation and policy reuse.

6 REFERENCES

Antonin, R., Ashley, H., Maximilian, E., Adam, G., Anssi, K., & Noah, D. Stable Baselines3. (2019). *GitHub repository.* https://github.com/DLR-RM/stable-baselines3

Ayala, A., Cruz, F., Campos, D., Rubio, R., Fernandes, B., & Dazeley, R. (2020). A comparison of humanoid robot simulators: A quantitative approach. *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob), Chile,* 6-8. doi:10.1109/icdl-epirob48136.2020.9278116

Bengio, Y., Louradour, J., Collobert, R., Weston, J. (2009). Curriculum learning. *26th Annual International Conference on Machine Learning, ICML 2009, Canada*.

Botteghi, N., Sirmacek, B., Mustafa, K. A. A., Poel, M., & Stramigioli, S. (2020). On reward shaping for mobile ROBOT Navigation: A reinforcement learning and SLAM based approach. arXiv.org. https://arxiv.org/abs/2002.04109.

Burduk, A. (2014). Stability analysis of the production system using simulation models. *Process Simulation and Optimization in Sustainable Logistics and Manufacturing*, 69–83. https://doi.org/10.1007/978-3-319-07347-7_5

Diankov, R. (2010). *Automated construction of robotic manipulation programs*. Carnegie Mellon University, The Robotics Institute. http://www.programmingvision.com/rosen_diankov_thesis.pdf

Elman, J. L. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, *48*(1), 71–99. https://doi.org/10.1016/0010-0277(93)90058-4

Evjemo, L. D., Gjerstad, T., Grøtli, E. I., & Sziebig, G. (2020). Trends in smart manufacturing: Role of humans and industrial robots in Smart Factories. *Current Robotics Reports*, *1*(2), 35–41. https://doi.org/10.1007/s43154-020-00006-5

Franceschetti, A., Tosello, E., Castaman, N., & Ghidoni, S. (2020). Robotic Arm Control and Task Training through Deep Reinforcement Learning. *Intelligent Autonomous Systems Laboratory, University of Padova, adova, Italy*, Retrieved from https://arxiv.org/abs/2005.02632

Frankfurt. (2021). Top 5 robot trends 2021. https://ifr.org/ifr-press-releases/news/top-5-robot-trends-2021

Ghafil, H. N., & Jármai, K. (2018). Research and application of industrial robot manipulators in vehicle and Automotive Engineering, a survey. *Lecture Notes in Mechanical Engineering*, 611–623. https://doi.org/10.1007/978-3-319-75677-6_53

Grau, A., Indri, M., Bello, L. L. & Sauter, T. (2017). Industrial robotics in factory automation: From the early stage to the Internet of Things. *IECON 2017 - 43rd Annual Conference of the*

IEEE Industrial Electronics Society, China, 6159-6164.
https://ieeexplore.ieee.org/document/8217070

Gu, S., Holly, E., Lillicrap, T., Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *In Proceedings - IEEE International Conference on Robotics and Automation, 3389-3396.*

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous Deep Q-Learning with Model-based Acceleration. ArXiv Preprint ArXiv:1603.00748.

Hawkins, K. P. (2013). Analytic inverse kinematics for universal robots UR-5/UR-10 arms. https://smartech.gatech.edu/bitstream/handle/1853/50782/ur_kin_tech_report_1.pdf

Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the knowledge in a neural network*. [1503.02531] Distilling the Knowledge in a Neural Network. https://export.arxiv.org/abs/1503.02531.

Huang, J. B., A Study of Reinforcement Learning for Robotics Applications. (2017). *National Chiao Tung University, Taiwan.* https://ir.nctu.edu.tw/handle/11536/142790

Lei, T., Paolo, G., & Liu, M. (2017). Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Canada*.

Liu, S., & Liu, P. (2021). A review of motion planning algorithms for Robotic Arm Systems. *Lecture Notes in Mechanical Engineering*, 56–66. https://doi.org/10.1007/978-981-16-4803-8_7

Michel, O. (2004). Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems, 39-42*. http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *Journal of machine learning research 21 (2020) 1-50.* https://www.semanticscholar.org/paper/Curriculum-Learning-for-Reinforcement-Learning-A-Narvekar-Peng/e102cd42c402026a3862d2e60a75eed7c78860a2/figure/0.

Peters, J. (2010). Policy gradient methods. *Scholarpedia*, *5*(11), 3698. *Max-Planck Institute, Germany & University of Southern California, USC.* https://doi.org/10.4249/scholarpedia.3698

Radosław Wachol, (2021). Robotic Arms and Digital Twins in Manufacturing. Retrieved April, 18, https://www.moicon.net/blog/robotic-arms-and-digital-twins-in-manufacturing

Schulman, J., Levine, S., Abbeel, P., Jordan, M.I., & Moritz, P..(2015), Trust Region Policy Optimization. ArXiv, abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv.org. https://arxiv.org/abs/1707.06347?source=post_page.

Sutton, R. S., McAllester, D. A., Singh, S. P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems 12, pp. 1057–1063, MIT Press, 2000*

Valle, A. (2021). Bilby Rover autonomous mobile robot in the context of Industry 4.0 - Survey on sensor-enabled applications through Webots simulations and implementation of Hector-SLAM algorithm for autonomous navigation. *Rel. Giovanni Belingardi, Maria Pia Cavatorta.*

Varin, P., Grossman, L., & Kuindersma, S. (2019). A comparison of action spaces for learning manipulation tasks. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. https://doi.org/10.1109/iros40897.2019.8967946 *Politecnico di Torino, Corso di laurea magistrale in Automotive Engineering (Ingegneria Dell'Autoveicolo)*, 51. https://webthesis.biblio.polito.it/17504/

Wang, M. (2021). Windows Hardware Installation Production Line Improvement, University of Alberta.

Webots. (2004) http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software. Cyberbotics Ltd. http://www.cyberbotics.com

Weinshall, D., Cohen, G., & Amir, D. (2018). Curriculum learning by transfer learning: Theory and experiments with deep networks. arXiv.org. https://arxiv.org/abs/1802.03796.

Weng, L. (2018). A (long) peek into reinforcement learning. *Lil'Log.* https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html.

Zhou, D., Jia, R., & Yao, H. (2021). Robotic Arm Motion Planning Based on Curriculum Reinforcement Learning. *2021 6th International Conference on Control and Robotics Engineering (ICCRE)*, 44-49.