博士論文

# Efficient Reinforcement Learning Using Simulation for Robot Control: Application to Logistics Cart Transportation System

（シミュレーションを活用した実ロボット制御のための効率的な強化学習:
台車搬送システムへの適用）

東京大学大学院工学系研究科航空宇宙工学専攻
松尾　凌輔

# Abstract

Robots are becoming more accessible and need to become highly autonomous. We believe that reinforcement learning is one of the promising methods to realize autonomous systems in dynamic and complex environments. Reinforcement learning can be expected to learn how to behave in order to achieve the goals set by the designer, even if it does not have sufficient knowledge of the problem, and this may enable it to solve complex control problems that could not be solved before. On the other hand, the agent needs to solve the complex task by trial and error, therefore, large amount of data is required. When considered in the context of the robot control, additional challenges arise. For example, if the agent tries to collect data in the real world, a large amount of time is required to execute the robot control and reset the environment, which adds to the learning cost, and unstable behavior in the early stages of learning may endanger the robot and its surroundings, which increases the cost of safety management. In this context, recent research was conducted to improve the learning efficiency and reduce the risk by using a physical simulation. However, in previous works, useful information from the simulation and knowledge of control is not utilized enough. Also, there are some issues that arise by using a physical simulation. Specifically, the reinforcement learning controller learned in a simulation environment may not work well in a real-world environment due to the difference in behavior between the simulation environment and the real-world environment. Therefore, we develop methods for efficient reinforcement learning by using some information and obtaining the robust controller in a simulation environment.

In Chapter 2, we propose a method to improve the learning efficiency by residual reinforcement learning using the control knowledge in a physical simulation. Using the control knowledge improve the learning efficiency and make learning process stable.

In Chapter 3, we introduce a method to improve the learning efficiency and the performance of a reinforcement learning controller in a delayed feedback environment by using undelayed feedback information.

In Chapter 4, we propose two-stage learning algorithm to learn the policy efficiently in a domain randomized environment. Domain randomization is a method that randomly samples textual noise of an image or/and environment parameters (friction, weight, etc.) and helps to obtain a robust controller. Two-stage learning algorithms includes learning feature extractor and reinforcement learning.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Toward the realization of autonomous robots

The control of robots has become a widespread and indispensable part of the manufacturing process due to their ability to perform predetermined movements with high precision. On the other hand, as machines and robots have become more familiar to people in recent years, there has been an increasing demand for autonomous systems that can function in dynamic and complex environments. We believe that one of the most promising method for realizing autonomous robot system is reinforcement learning.

With the advent of deep learning, various machine learning algorithms using neural networks have been proposed, and reinforcement learning is one of the areas being studied by many researchers. In reinforcement learning, the environment is unknown, and an agent collects data by trial and error and learns a controller in the environment. This property has the potential to learn a controller for complex tasks that are difficult to model simply.

In recent years, the success of deep reinforcement learning in game area has attracted much attention. Starting with the Deep Q Network (DQN) [1], which plays the Atari game at first, various methods have been proposed, and some of them can compete with humans or better in various games, for example, go and Shogi.

One important development is the emergence of the algorithms for continuous control tasks [2, 3, 4, 5, 6, 7], which have been applied to a variety of robotic applications. For example, self-driving cars [8], object grasping [9], object throwing [10], quadrotor [11] , quadrupedal walking [12], and navigation [13]. On the other hand, many challenges remain

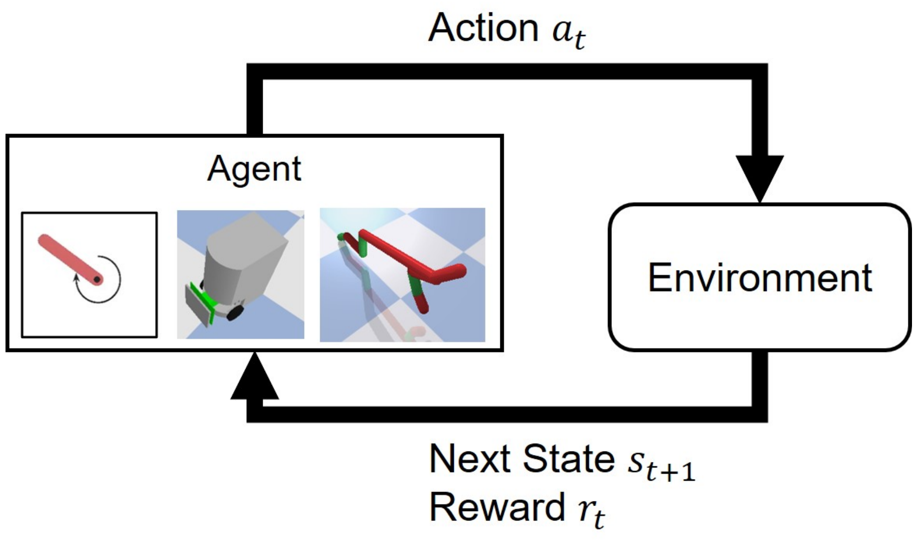


**Figure.1.1:** Problem setting of reinforcement learning

when we try to apply reinforcement learning to robot control. One of them is that the cost of performing trial and error for learning in the real world is very high. The example of the solution to this problem is to improve the sampling efficiency (the degree of improvement of the controller in relation to the amount of experiences) to obtain a good controller efficiently with a small amount of experiences, or to reduce the cost of gathering experiences by using a simulation. There have been many studies on reinforcement learning for robots using a simulation. However, there is a problem that remains from the use of the simulation: the gap between the simulation environment and the real-world environment. Physical simulations model the real world and represent its behavior numerically. Therefore, it is not possible to reproduce the real world perfectly. The controller that works in the simulation may not work well in the real world because of the difference between the behavior in the simulation and the behavior in the real world.

While reinforcement learning has the potential to build highly autonomous systems, it faces many challenges as mentioned above, and a number of major developments are needed before it can be widely adopted.

## 1.2 Reinforcement learning for continuous control

The problem setup for reinforcement learning is shown in Fig. 1.1. The agent acts in the environment through its behavior $a_t$ and receives the reward $r_t$ and the next state $s_{t+1}$ from the environment. Most reinforcement learning assumes that the process is a Markov decision process. A Markov decision process means that the state transitions satisfy Markovianity (Markovianity is the property that the posterior probability of the next state is determined from the current state and behavior and does not depend on the past). In this interaction, the reinforcement learning problem can be formulated as reward maximization. Here, we consider time-discounted cumulative reward as the objective function of reinforcement learning.

$$\mathbb{E}_{s\sim p_\pi, a\sim\pi}\left[\sum_{i=0}^{T}\gamma^i r_i\right]. \tag{1.1}$$

In this case, the time-discounted cumulative reward sum over the future for taking action $a$ in state $s$ is defined as the state action value function:

$$Q(s,a) = \mathbb{E}\left[\sum_{i=t}^{T}\gamma^{i-t} r_i | s_t = s, a_t = a\right], \tag{1.2}$$

which is called Q-function. In this case, from Markov property, we can be written as

$$Q(s,a) = r + \gamma\mathbb{E}\left[\sum_{i=t+1}^{T}\gamma^{i-t-1} r_i\right], \tag{1.3}$$

where $r$ is the reward as the result of action $a$ in state $s$. Focusing on the expectation operation, we can be defined as

$$Q(s,a) = r + \gamma\mathbb{E}_{s'\sim p(s'|s,a), a'\sim\pi(\cdot|s')}\left[Q(s',a')\right]. \tag{1.4}$$

where $p(s'|s,a)$ is the state transition probability, which is conditioned only on the current state and action due to Markov property.

We first introduce Q-learning [14] to understand the continuous control by reinforcement learning. We assume that the states and actions are discrete values. That is, there are $|S| \times |A|$ combinations of $(s,a)$ (where $|S|$ is the number of states and $|A|$ is the number of actions), and each combination can be considered in the form of a table with Q-values. When an agent

**Algorithm 1** DQN [1]

```
1: Initialize parameter $\theta$ and target network parameter $\bar{\theta} \leftarrow \theta$
2: Initialize replay memory $\mathcal{D}$
3: $s_0 \leftarrow$ env.reset()
4: for $t = 0, 1, \ldots, T$ do
5:     With probability $\epsilon$ select random action $a_t$
6:     otherwise $a_t = \arg\max_a Q(s_t, a)$
7:     $s_{t+1}, r_t, done$ = env.step($a_t$)
8:     $D$.add($s_t, a_t, r_t, s_{t+1}, done$)
9:     Sample mini-batch $(s_i, a_i, r_i, s_{i+1}, done_i)$ from $\mathcal{D}$ (size $N$)
10:    $y_i = r_i + done_i \cdot \gamma \max_a Q_{\bar{\theta}}(s_{i+1}, a)$
11:    $L_\theta = \sum_i (y_i - Q(s_i, a_i))^2 / N$
12:    if done then $s_{t+1} \leftarrow$ env.reset() end if
13:    if $t$ mod $C = 0$ then $\bar{\theta} \leftarrow \theta$
14: end for
```

observes the state $s$, it selects the action from the Q table to maximizes the Q-value. In other words, the action $a$ is defined as

$$a = \arg\max_b Q(s, b), \tag{1.5}$$

Then, the state-action value function is

$$Q(s, a) = r + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} \left[ \max_{a'} Q(s', a') \right]. \tag{1.6}$$

In Q-learning, the state-action value function is updated each time step the experience $(s_n, a_n, r_n, s_{n+1})$ is obtained; the equation for the $n$-th update is

$$Q_n(s_n, a_n) = (1 - \alpha_n) Q_{n-1}(s_n, a_n) + \alpha_n \left[ r_n + \gamma \max_{a'} Q(s_{n+1}, a') \right]. \tag{1.7}$$

where $\alpha_n$ is the learning rate. This update equation brings the Q-value closer to $[r_n + \gamma \max_{a'} Q(s_{n+1}, a')]$. Also, only the element of $(s_n, a_n)$ sampled at this time step are updated in the Q table.

Deep Q-learning, which uses deep learning and is designed for learning stably, has been proposed by Mnih et al [1]. One of the innovations in DQN is the use of experience replay.

Experience replay is a method of learning by storing a certain amount of experiences in memory and then sampling experiences from it randomly. This weakens the correlation between the sampled experiences to stabilizes the learning process. Another idea is the target Q-network. The target $[r_n + \max_{a'} Q(s_{n+1}, a')]$ in naive Q-learning is always changing, and if this change is too severe, the learning becomes unstable. Therefore, in the method of the target Q-network, the target is updated at a certain interval and is fixed at other times. Unlike Q-learning, in DQN, the states take continuous values, therefore, the state-action value is approximated by a neural network with the parameter $\theta$. Also, fixing the target means fixing the parameter. Specifically, this method prepares a parameter $\bar{\theta}$ for the state-action value function in the target, which copies $\theta$ at every $C$ step. Then, the loss function for learning the parameter $\theta$ is

$$L_\theta = \frac{1}{N}\sum_i^N \left(r_i + \gamma \arg\max_a Q_{\bar{\theta}}(s_{i+1}, a) - Q_\theta(s_i, a_i)\right)^2 . \tag{1.8}$$

The algorithm for DQN is shown in Algorithm 1.

Next, we introduce the reinforcement learning algorithm for continuous control. The structure used in most of the reinforcement learning for continuous control is actor critic. Actor critic architecture has two models: actor, which outputs the action, and critic, which evaluates the action. One of the advantages of having two models is that continuous action can be easily introduced. In the following, we describe Deep Deterministic Policy Gradient (DDPG) [2], which has the actor critic architecture.

If the action is represented by a neural network with parameter $\phi$, the action $a$ to be taken in state $s$ is written as

$$a = \pi_\phi(s). \tag{1.9}$$

In DQN, when calculating the target of the state-action value, the action is selected by argmax operator, which is replaced by the policy $\pi_\phi$. In DDPG, the target is defined as

$$r_t + \gamma Q_{\bar{\theta}}(s_{t+1}, \pi_{\bar{\phi}}(s_{t+1})), \tag{1.10}$$

where $\bar{\phi}$ is a parameter of the target network of the policy. The state-action value function is learned to minimize the mean least square as in DQN.

$$L_\theta = \frac{1}{N}\sum_i^N \left(r_i + \gamma Q_{\bar{\theta}}(s_{i+1}, \pi_{\bar{\phi}}(s_{i+1})) - Q_\theta(s_i, a_i)\right)^2 . \tag{1.11}$$

**Algorithm 2** DDPG [2]

```
Output: optimal policy $\pi^*$
 1: Initialize critic networks $Q_\theta$ and policy network $\pi_\phi$
 2: Initialize target networks $\bar{\theta}, \bar{\phi} \leftarrow \theta,, \phi$
 3: Initialize replay buffer $\mathcal{D}$
 4: Initialize environment env
 5: $s_0 \leftarrow$ env.reset()
 6: for $t = 0, 1, \ldots, T$ do
 7:     $a_t \leftarrow \pi_\phi(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$
 8:     $s_{t+1}, r_t, done_t \leftarrow$ env.step$(a_t)$
 9:     $\mathcal{D}$.add$(s_t, a_t, r_t, s_{t+1}, done_t)$
10:     Sample mini-batch $(s_i, a_i, r_i, s_{i+1}, done_i)$ from $\mathcal{D}$ (size $N$)
11:     $a' \leftarrow \pi_{\bar{\phi}}(s_{i+1})$
12:     $y_i \leftarrow r_i + done_i \cdot \gamma Q_{\bar{\theta}}(s_{i+1}, a')$
13:     Update critics by (1.11)
14:     Update policy by (1.13) using $Q_\theta$
15:     Update target networks:
        $\bar{\theta} \leftarrow \tau\bar{\theta} + (1 - \tau)\theta$
        $\bar{\phi} \leftarrow \tau\bar{\phi} + (1 - \tau)\phi$
16:     if done then $s_{t+1} = env.reset()$ end if
17: end for
```

Also, unlike in DQN, the target network is updated slowly, following the parameters, rather than periodically in DDPG. The update equations for the target network of the Q-function and the policy are

$$\bar{\theta} \leftarrow \tau\theta + (1 - \tau)\bar{\theta}, \bar{\phi} \leftarrow \tau\phi + (1 - \tau)\bar{\phi}. \tag{1.12}$$

On the other hand, the policy $\pi_\phi$ is learned to maximize the following objective function.

$$J_\phi = \mathbb{E}\left[Q_\theta(s, \pi_\phi(s))\right]. \tag{1.13}$$

The state-action value represents the expected value of how much reward will be obtained in the future if a certain action $a$ is taken in a certain state $s$, which shows how appropriate the action is in a certain state $s$. Therefore, the agent can update the policy parameters by the gradient decent of (1.13). The algorithm for DDPG is shown in Algorithm 2.

## 1.3 Physical simulation for reinforcement learning in robotics

In recent years, there are a variety of physical simulators available, and users need to select a suitable simulator for the control task. For example, AirSim [15] is one of the suitable simulator for solving the control task of UAVs, and CARLA [16] is one of the reasonable simulator for auto driving. When we would like to solve the problem of navigation or manipulation by reinforcement learning, MuJoCo [17], Pybullet [18], V-Rep [19] and Gazebo [20] are often used.

In terms of what is required for physical simulation, for example, Erez et al. [21] focused on the relationship between computation time and trajectory error. MuJoCo has shown superior performance in this study. Also, Collins et al. [22] compared various simulations from a functional perspective (sensor abundance, rendering, etc.).

On the other hand, there are some works on differentiable physical simulations. In [23], automatic differentiation was introduced to allow computing fast and accurate approximations of the derivative in physical simulations. Residual physics models are also proposed to learn residual elements from analytical physical simulations. In [12], the input-output mapping of the actuator is learned from real data using supervised learning, which is then used to correct the behavior in the simulation environment to learn highly accurate real-world control in the simulation. Heiden et al. [24] proposed a hybrid simulator, which is a combination of an analytical physical model that is differentiable by automatic differentiation like [23] and a residual physical model that is defined by a neural network and is also differentiable. Inspired by Sparse Input Neural Networks (SPINN) [25], this work shows that it is possible to learn a sparse residual physical model that corrects only the necessary physical quantities.

Improvements in the accuracy, computational speed, and differentiability of physical simulations will enable the collection of large amounts of experiences in a short period of time in a simulation environment similar to the real-world environment. Therefore, these improvements will accelerate research and development on simulation-based reinforcement learning for robot control.

## 1.4 Summary of remaining chapters

In Chapter 2, we propose a method to efficiently learn a controller for logistics cart transportation by residual and distributed reinforcement learning in a physical simulation. We show that the proposed method make the improvement of a controller efficient in a simulation environment, and the policy learned in the simulation realizes a logistics cart transportation in a real-world environment.

In Chapter 3, we propose a method to deal with a delayed feedback that occurs in the real-world environment. We use the undelayed observation and reward from the simulation to learn the policy that works in a delayed feedback environment efficiently.

In Chapter 4, we focus on the study of domain randomization in order to solve the problem of misalignment caused by using a simulation, and aim to obtain favorable controller in a domain randomized environment. In this chapter, we propose a method that incorporates privileged information from simulation and state representation learning to extract essential information for making the problem setting into simple MDP.

In Chapter 5, we summarize the proposed method and discuss future issues.

# Chapter 2

# Residual Reinforcement Learning for Logistics Cart Transportation

## 2.1 Introduction

Object transportation is increasingly being automated through the use of automated guided vehicles (AGVs) in large warehouses. However, this is less common in smaller warehouses, where objects are typically conveyed by human workers with logistics carts because existing automation systems by AGVs are not supported to transport existing logistics carts. For example, a space below a logistics cart is too small to move under and lift it. To address this, an automated object transportation system for these warehouses [26] was proposed. In this system, the robot's position is estimated by utilizing images from a camera on the ceiling, and two robots grasp a logistics cart and transport it as shown in Fig. 2.1. The strategy of having two robots hold a logistics cart makes it possible to automate the transportation without additional equipment. However, control for transporting a logistics cart remains a difficult problem because robots need to keep holding the cart. There is currently no method for making a logistics cart track a trajectory.

In the context of robot control by RL, unique methods such as residual reinforcement learning [27, 28, 10] and control structured policy learning [29], which include a feedback control structure, have been proposed. These methods improve the learning efficiency by utilizing prior knowledge.

In this chapter, we propose a method for constructing a residual reinforcement learning controller that modifies the base controller by reinforcement learning for the logistics cart

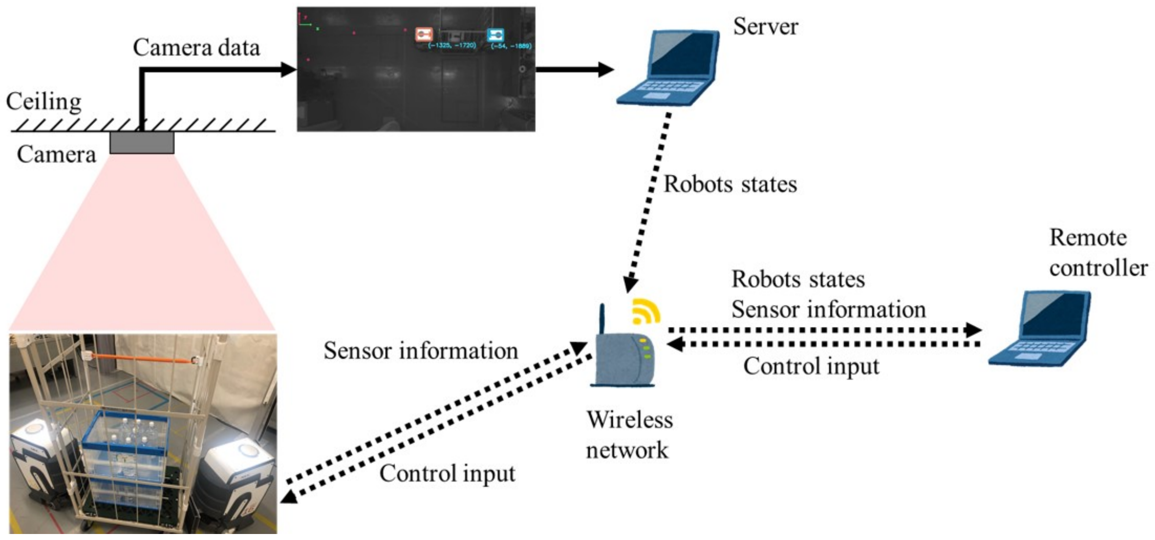


**Figure.2.1:** System overview.

transportation along a given trajectory. We do not consider trajectory planning. Our main contributions are as follows.

- We achieve cooperative logistics cart transportation by a reinforcement learning controller.
- We utilize the physical simulator and knowledge of the feedback controller to make the controller learn the policy efficiently.
- Our proposed controller learned by physical simulator can be transferred to real-world robots without additional parameter tuning.

## 2.2 Related works

### 2.2.1 Deep reinforcement learning for cooperative multi-agent control

Deep reinforcement learning methods that deal with high-dimensional observation for multi-agent control have been extensively researched (see [30] for a survey). Much research has focused on discrete action space, with less attention paid to continuous action space.

Lowe et al. [31] implemented an actor-critic model in a cooperative multi-agent system where each agent has an independent Q-function whose inputs include its own observation, other agents' observations, and the policy. In the training phase, each agent learns the Q-function with the other agents' information and the policy learned using this Q-function. Then, in the execution phase, each agent utilizes the policy with only local information.

Gupta et al. [32] compared three strategies and learning algorithms in various simulations that request discrete action or continuous action at first. The three strategies are (i) Centralized, where one model includes joint observation and joint action, (ii) Concurrent, where each agent has an independent observation and model, and (iii) Parameter Sharing, where each agent has an independent observation and sharing model parameter. The authors found that Parameter Sharing was the most scalable to the number of agents and proposed Parameter Sharing TRPO (PS-TRPO). TRPO is a method of deep reinforcement learning for continuous control [3].

In our system, two robots are controlled as a single agent, however, it is related to centralized strategy in the context of multi-agent reinforcement learning.

### 2.2.2 Cooperative object transportation

Cooperative object transportation has been researched since the 90s. The strategies for solving this problem are mainly divided into three groups: the pushing strategy, the grasping strategy (where the object is fixed between robots by some equipment or is placed directly on the robots), and the caging strategy [33]. The pushing-only strategy appears simple strategy, however, robots can only push an object, therefore, the system requires delicate control. The grasping strategy has physical connection between robots and an object. Thus, it is easy to realize stable transportation and robots can exert pulling power on an object. However, physical connection is required. The caging strategy can realize stable transportation by caging an object by robots, however, a controller is required to keep caging an object and caging an object by a few robots is difficult. We adopt the caging strategy because of two reasons: (i) a logistics cart has casters, therefore, it is easy to keep moving due to inertia, and (ii) our system is introduced to existing warehouses and transports existing logistics carts, thus, an additional attachment for physical connection is undesirable.

Research by Ohsaki et al. [34] utilizes the pushing strategy with the object on casters. Two cooperating robots arrange small dollies under an object and then one robot pushes the object for transportation. This research focused on arranging the dollies and stabilizing the pushing robot, and did not discuss the details of the object transportation. Our research differs in that we focus specifically on logistics cart transportation. Another difference is that the weight of the object in [34] was 35 kg, whereas ours is considerably heavier.

As for the grasping strategy, some studies have examined leader-follower systems with two mobile robots [35, 36, 37]. In a leader-follower system, the leader tracks a given trajectory and the follower follows the leader for transporting the object. Extended methods where the object is regarded as a virtual leader and the robots are followers have been proposed [38].

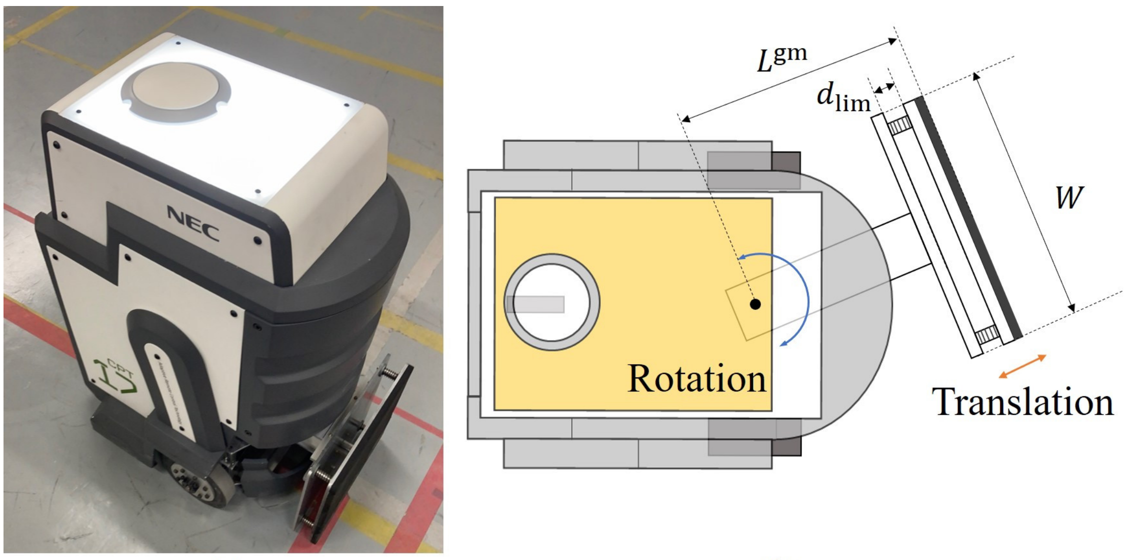
Brown et al. [39] proposed a simple controller for cooperative object transportation by using the caging strategy with two robots. One of the robots guides the object movement, and the other pushes the object. Their research assumes that the contact point between the robots and the object slides for making the guidance robot track an arc trajectory, while in contrast, our system assumes that the contact point between the logistics cart and the robots does not slide. Moreover, the object property in [39] is different from that in our research.

With regard to the caging strategy, leader-follower systems that utilize three or more robots have been proposed. Wang et al. proposed a system featuring one leader and some followers [40]. Wan et al. [41] proposed a system where the object is regarded as the leader and robots are followers. Wan et al. developed a leader-follower system featuring a multi-fingered mechanism [42].

Methods for caging a concave object by means of a two-fingered mechanism have also been proposed [43, 44, 45]. See the work by Makita et al. for a detailed survey of the methods utilizing the caging strategy [46].

A reinforcement learning controller for cooperative object transportation by the grasping strategy has been proposed [47]. In this method, the controller is constructed by a deep Q network (DQN) that outputs a discrete action. Two robots and an object are arranged in a fixed environment and their purpose is to arrive at an exit. This research differs from our own in many respects, including the property of the object, the method of transportation, the action space, and the concept underlying the processing observation.

While much research has examined cooperative object transportation, there has been no research on the transportation of a logistics cart by two robots using the caging strategy, to the best of our knowledge.



(a) (b)

**Figure.2.2:** Hardware architecture: (a) prototype and (b) holding mechanism.

## 2.3 Methodology

### 2.3.1 Hardware architecture

The hardware architecture is shown in Fig. 2.2. The plate that contacts a logistics cart has a spring mechanism to enable flexible holding. This plate is covered by a high-friction material to avoid slippage between the robot and the cart. The holding mechanism has a spring-damper characteristic that rotates to track a circular trajectory. It stays at the center when it is in the free state, as shown in Fig. 2.2 (a).

This system utilizes two robots: a supporting robot that supports a change in the direction of the logistics cart, and a pushing robot that pushes the cart. These roles are fixed when the cooperative logistics cart transportation is started and do not change during transportation.

### 2.3.2 Problem statements

#### Observation

The position and orientation of the robots are estimated by the image from the ceiling camera, which utilizes rectangular recognition. The robot position corresponds to the center of rotation and the installation point of the holding mechanism. The position and orientation of the logistics cart, in contrast, cannot be estimated by the image because of the complexity of the logistics cart and the loading object.

If the controller utilizes the robots' position and orientation in global coordinates, it does not work when the scale of the environment changes. Therefore, we utilize local coordinates calculated by the estimated position and orientation of the logistics cart. Fig. 2.3 shows the ideal arrangement of the robots and logistics cart in local coordinates. The orange line is the ideal trajectory of the logistics cart and is given in advance, i.e., we do not consider trajectory planning. The true position and orientation of the logistics cart can be calculated by the pushing robot position, orientation, and rotation angle of the holding mechanism. The $y$-axis of local coordinates corresponds to the line from the logistics cart position to the center of the circular trajectory and the $x$-axis is the tangent of the circular trajectory. The position and orientation of the robots in these local coordinates is the observation of the reinforcement learning controller. However, in real situations, robots often cannot maintain the ideal holding, so the position and orientation of the logistics cart become estimated values and local coordinates are calculated by the estimated position and orientation. The reason we only use the pushing robot for calculating the logistics cart position and orientation is that the pushing robot is required to contact the logistics cart for transporting it, which means the estimated logistics of the cart position and orientation are calculated more accurately.

Additionally, the variation of each robot's touch plate and the rotation angle of the holding mechanism are included in the observation, and time differentials of these observations are added. Finally, an action before one step, the trajectory curvature, and the size of the logistics cart are appended to the observation.

#### Action

The continuous action space is defined as each robot's linear velocity and rotational velocity. Therefore, the action space has four dimensions. The range of the linear velocity of each robot is $[-0.5, 0.5]$ m/s and the range of the rotational velocity of each robot is $[-\pi/6, \pi/6]$ rad/s. The observation variables and action variables are listed in Table 2.1.

### 2.3.3 Reward shaping

The reinforcement learning controller is required to make the logistics cart follow the trajectory as accurately and speedily as possible. Therefore, the reward is shaped by three

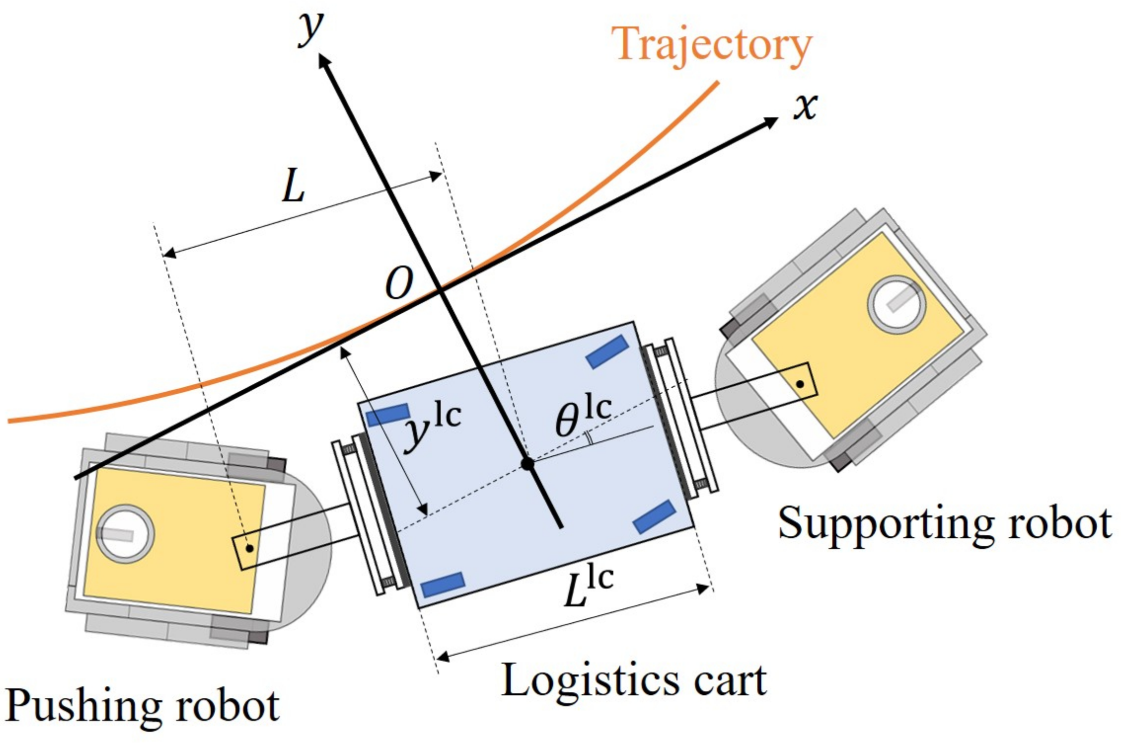


**Figure.2.3:** Local coordinates for calculating observation.

elements: a) logistics cart's position and orientation errors from the trajectory, b) logistics cart's velocity, and c) each robot's position and orientation relative to the logistics cart's position and orientation. The reward element a) is based on [48, 49] and b) is based on [49], which are studies on trajectory tracking using reinforcement learning. In the real world, the true position, orientation, and velocity of the logistics cart cannot be obtained, but in simulation, the reward calculator can refer to the ground truth and then calculate the reward by utilizing it.

### Logistics cart's position and orientation errors

First, we introduce the reward for precision of tracking the trajectory. This is divided into two elements: the position deviation and the orientation deviation. The reward for position deviation $r_{\mathrm{pd}}$ is defined as

$$r_{\mathrm{pd}} = \exp\left(-k_{\mathrm{pd}}|y^{\mathrm{lc}}|\right), \tag{2.1}$$

**Table.2.1:** Observation variables and action variables. FB obs. means the observations for the feedback controller.

| Observation variables | Symbols | FB obs. |
|---|---|---|
| Supporting robot position | $x^{\mathrm{sr}}, y^{\mathrm{sr}}$ | ✓ |
| Supporting robot orientation | $\cos\theta^{\mathrm{sr}}, \sin\theta^{\mathrm{sr}}$ | ✓ |
| Pushing distance of grasping mechanism of supporting robot | $d^{\mathrm{sr}}$ | |
| Rotation angle of rotation mechanism of supporting robot | $\phi^{\mathrm{sr}}$ | |
| Pushing robot position | $x^{\mathrm{pr}}, y^{\mathrm{pr}}$ | ✓ |
| Pushing robot orientation | $\cos\theta^{\mathrm{pr}}, \sin\theta^{\mathrm{pr}}$ | ✓ |
| Pushing distance of grasping mechanism of pushing robot | $d^{\mathrm{pr}}$ | |
| Rotation angle of grasping mechanism of pushing robot | $\phi^{\mathrm{pr}}$ | |
| Estimated position of logistics cart | $\hat{y}^{\mathrm{lc}}$ | |
| Estimated orientation of logistics cart | $\cos\hat{\theta}^{\mathrm{lc}}, \sin\hat{\theta}^{\mathrm{lc}}$ | |
| Time derivatives of above variables | 15 dimensions | |
| Logistics cart size | $L^{\mathrm{lc}}$ | ✓ |
| Curvature | $\rho$ | ✓ |
| Action before one step | $v^{\mathrm{sr}}, w^{\mathrm{sr}}, v^{\mathrm{pr}}, w^{\mathrm{pr}}$ | |

| Action variables | |
|---|---|
| Each robot's velocity and angular velocity | $v^{\mathrm{sr}}, w^{\mathrm{sr}}, v^{\mathrm{pr}}, w^{\mathrm{pr}}$ |

where $y^{\mathrm{lc}}$ is the logistics cart's position in the local $y$-coordinate and $k_{\mathrm{pd}}$ is a coefficient set to 7.5.

The reward for orientation deviation $r_{\mathrm{od}}$ is defined as

$$r_{\mathrm{od}} = \exp\left(-k_{\mathrm{od}}|\theta^{\mathrm{lc}}|\right), \tag{2.2}$$

where $|\theta^{\mathrm{lc}}|$ is the logistics cart's orientation in the local coordinates and $k_{\mathrm{od}}$ is a coefficient set to $13.5/\pi$.

### Logistics cart's velocity

Ideally, the logistics cart should be transported as speedily as possible. Therefore, the reward for the logistics cart's velocity $r_{\mathrm{dv}}$ is defined as

$$r_{\mathrm{dv}} = k_{\mathrm{dv}} d_{\mathrm{d}}, \tag{2.3}$$

where $d_{\mathrm{d}}$ is the amount of the logistics cart's advancement along a given trajectory and $k_{\mathrm{dv}}$ is a coefficient set to 10. To calculate $d_{\mathrm{d}}$, first, polar coordinates whose origin is the center of a circular trajectory are defined and the moving angle of the logistics cart in one step is defined as $\Delta\theta_{\mathrm{d}}$. Then, if the radius of curvature is $R$, $d_{\mathrm{d}}$ is calculated as $d_{\mathrm{d}} = R\Delta\theta_{\mathrm{d}}$.

### Each robot's position and orientation relative to logistics cart's position and orientation

Stable holding of a logistics cart is important for users' sense of security. Thus, we introduce the reward for holding it as long as possible. For holding the logistics cart, each robot's position and orientation relative to the logistics cart's position and orientation are required to stay in a certain area. This area is decided by the range of motion of the touch mechanism $d_{\mathrm{lim}}$, the width of the touch mechanism $W$, and the angle range of the rotation mechanism $\phi_{\mathrm{lim}}$. Therefore, the area is defined as

$$\begin{aligned} &x\text{-axis} : \tfrac{L^{\mathrm{lc}}}{2} + L^{\mathrm{gm}} - d_{\mathrm{lim}} \leq |r_x^{\{\mathrm{sr},\mathrm{pr}\}}| \leq \tfrac{L^{\mathrm{lc}}}{2} + L^{\mathrm{gm}}, \\ &y\text{-axis} : |r_y^{\{\mathrm{sr},\mathrm{pr}\}}| \leq \tfrac{W}{5}, \\ &\text{orientation} : |r_\theta^{\{\mathrm{sr},\mathrm{pr}\}}| \leq \phi_{\mathrm{lim}}, \end{aligned} \tag{2.4}$$

where $r_x^{\{\mathrm{sr},\mathrm{pr}\}}, r_y^{\{\mathrm{sr},\mathrm{pr}\}}, r_\theta^{\{\mathrm{sr},\mathrm{pr}\}}$ are each robot's position and orientation relative to the logistics cart's position and orientation and $L^{\mathrm{gm}}$ is the length from the robot's origin to the touch plate of the holding mechanism. $L^{\mathrm{lc}}$ is the length of the logistics cart (see Fig. 2.3). The reward for relative position and orientation is defined as

$$r_{\mathrm{rp}} = \begin{cases} 1.0 & \text{(both robots are in the area defined by (4))} \\ 0.1 & \text{(otherwise)} \end{cases} . \tag{2.5}$$

Finally, the reward is defined by combining all elements:

$$r = \begin{cases} r_{\mathrm{dv}} r_{\mathrm{rp}} \left( r_{\mathrm{pd}} + r_{\mathrm{od}} \right) & (0 \leq r_{\mathrm{dv}}) \\ r_{\mathrm{dv}} \left( 4 - r_{\mathrm{pd}} - r_{\mathrm{od}} \right) & (otherwise) \end{cases} . \tag{2.6}$$

This equation means that not only that the logistics cart tracks a given trajectory but also that an amount of the logistics cart advancement along a given trajectory is needed for obtaining the reward. If the logistics cart moves backwards along a given trajectory, the system given a minus reward even if it tracks a given trajectory perfectly. We calculate the reward shaping by the product of the error elements and the velocity element as in [49], which is research on high-speed trajectory tracking control for autonomous driving. This product makes elements improve for obtaining higher reward. As our research scenario allows a logistics cart to move backwards, we have added the reward for moving backwards.

### 2.3.4 Formation-based feedback controller

We utilize the feedback controller proposed in [38] for cooperative logistics cart transportation by formation-based control, which corresponds to the grasping strategy as the base controller. In this method, the system regards an object as a virtual leader (VL) and the robots as followers, and the leader-follower system controls the robots by feedback control for maintaining formation. We show later that the robots' target orientation can be written simply.

The robots' target is calculated as the relative state of VL. VL's target velocity, position, and orientation in local coordinates at time $k$ are $\boldsymbol{v}^{\mathrm{VL}}_{\mathrm{tgt}}(k) = \left[v^{\mathrm{VL}}_{\mathrm{tgt}}, \omega^{\mathrm{VL}}_{\mathrm{tgt}}\right]^\top$, $\boldsymbol{x}^{\mathrm{VL}}_{\mathrm{tgt}}(k) = [0, 0]^\top, \theta^{\mathrm{VL}}_{\mathrm{tgt}}(k) = 0$. Local coordinates are on a given trajectory and $x$-axis corresponds to the tangent of a given trajectory, so the logistics cart's target position and orientation are both zero. Also, VL's target position at time $k + 1$ in local coordinates at time $k$ is

$$\boldsymbol{x}^{\mathrm{VL}}_{\mathrm{tgt}}(k+1) = \begin{cases} \left[R \sin \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}, R(1 - \cos \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}})\right]^\top & \left(\omega^{\mathrm{VL}}_{\mathrm{tgt}} \neq 0\right) \\ \left[v^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}, 0\right]^\top & \left(\omega^{\mathrm{VL}}_{\mathrm{tgt}} = 0\right) \end{cases}, \tag{2.7}$$

where $R$ is the radius of the curvature of a given trajectory. In addition, $t_{\mathrm{s}}$ [s] is the time elapsed in one step, which we set to 0.1 s. The supporting robot's target position at time $k$ in local coordinates at time $k$ is $\boldsymbol{x}^{\mathrm{sr}}_{\mathrm{tgt}}(k) = [L, 0]^\top$, and the supporting robot's target position at time $k+1$ in local coordinates at time $k$ is written as $\boldsymbol{x}^{\mathrm{sr}}_{\mathrm{tgt}}(k+1) = \boldsymbol{x}^{\mathrm{VL}}_{\mathrm{tgt}}(k+1) + \left[L \cos \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}, L \sin \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}\right]^\top$.

The supporting robot's target velocity is $||\boldsymbol{x}^{\mathrm{sr}}_{\mathrm{tgt}}(k+1) - \boldsymbol{x}^{\mathrm{sr}}_{\mathrm{tgt}}(k)||/t_{\mathrm{s}}$. The result of calculating this equation is

$$v^{\mathrm{sr}}_{\mathrm{tgt}}(k) = \sqrt{{v^{\mathrm{VL}}_{\mathrm{tgt}}}^2 + 2L^2 \left(1 - \cos \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}\right) / t_{\mathrm{s}}^2}. \tag{2.8}$$

Additionally, the supporting robot's target orientation in local coordinates at time $k$ is calculated from its positions at times $k-1$ and $k$ as

$$\theta^{\mathrm{sr}}_{\mathrm{tgt}}(k) = \pi + \tan^{-1}\frac{y^{\mathrm{sr}}_{\mathrm{tgt}}(k) - y^{\mathrm{sr}}_{\mathrm{tgt}}(k-1)}{x^{\mathrm{sr}}_{\mathrm{tgt}}(k) - x^{\mathrm{sr}}_{\mathrm{tgt}}(k-1)} = \pi + \rho L - \frac{\omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}}{2}, \tag{2.9}$$

where $\rho$ is the curvature of a given trajectory, which means the target orientation can be calculated simply. The details of the calculation process are shown in the Appendix. The first item $\pi$ is added so that the supporting robot moves backwards. From the above, the supporting robot's targets in local coordinates at time $k$ are

$$\begin{aligned}\boldsymbol{v}^{\mathrm{sr}}_{\mathrm{tgt}}(k) &= \left[v^{\mathrm{sr}}_{\mathrm{tgt}}, \omega^{\mathrm{sr}}_{\mathrm{tgt}}\right]^{\top} = \left[\sqrt{{v^{\mathrm{VL}}_{\mathrm{tgt}}}^{2} + 2L^{2}\left(1-\cos\omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}\right)/t_{\mathrm{s}}^{2}},\ \omega^{\mathrm{VL}}_{\mathrm{tgt}}\right]^{\top},\\ \boldsymbol{x}^{\mathrm{sr}}_{\mathrm{tgt}}(k) &= [L, 0]^{\top}, \theta^{\mathrm{sr}}_{\mathrm{tgt}}(k) = \pi + \rho L - \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}/2.\end{aligned} \tag{2.10}$$

The pushing robot's targets are calculated in the same way.

$$\begin{aligned}\boldsymbol{v}^{\mathrm{pr}}_{\mathrm{tgt}}(k) &= \left[v^{\mathrm{pr}}_{\mathrm{tgt}}, \omega^{\mathrm{pr}}_{\mathrm{tgt}}\right]^{\top} = \left[\sqrt{{v^{\mathrm{VL}}_{\mathrm{tgt}}}^{2} + 2L^{2}\left(1-\cos\omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}\right)/t_{\mathrm{s}}^{2}},\ \omega^{\mathrm{VL}}_{\mathrm{tgt}}\right]^{\top},\\ \boldsymbol{x}^{\mathrm{pr}}_{\mathrm{tgt}}(k) &= [-L, 0]^{\top}, \theta^{\mathrm{pr}}_{\mathrm{tgt}}(k) = -\rho L - \omega^{\mathrm{VL}}_{\mathrm{tgt}} t_{\mathrm{s}}/2.\end{aligned} \tag{2.11}$$

After calculating each robot's targets, each robot's position and orientation errors between the targets and the observations are calculated as

$$\begin{aligned}\boldsymbol{x}_e^{\{\mathrm{sr,pr}\}} &= \left[x_e^{\{\mathrm{sr,pr}\}}\ y_e^{\{\mathrm{sr,pr}\}}\right]^{\top} = {}^{\{\mathrm{sr,pr}\}}T_{\mathrm{trj}}\left(\boldsymbol{x}_{\mathrm{tgt}}^{\{\mathrm{sr,pr}\}} - \boldsymbol{x}_{\mathrm{obs}}^{\{\mathrm{sr,pr}\}}\right)\\ \theta_e^{\{\mathrm{sr,pr}\}} &= \theta_{\mathrm{tgt}}^{\{\mathrm{sr,pr}\}} - \theta_{\mathrm{obs}}^{\{\mathrm{sr,pr}\}}\end{aligned}, \tag{2.12}$$

where ${}^{\{\mathrm{sr,pr}\}}T_{\mathrm{trj}}$ is the transformation matrix from local coordinates to each robot's coordinates. $\boldsymbol{x}_{\mathrm{obs}}^{\{\mathrm{sr,pr}\}}, \theta_{\mathrm{obs}}^{\{\mathrm{sr,pr}\}}$ are the observation value of each robot's position and orientation. Following [50], each robot's velocity is calculated as

$$\pi_{\mathrm{fb}}^{\{\mathrm{sr,pr}\}}(\boldsymbol{o}_{\mathrm{fb}}) = \begin{bmatrix} v_{\mathrm{fb}}^{\{\mathrm{sr,pr}\}} \\ \omega_{\mathrm{fb}}^{\{\mathrm{sr,pr}\}} \end{bmatrix} = \begin{bmatrix} v_{\mathrm{tgt}}^{\{\mathrm{sr,pr}\}}\cos\theta_e^{\{\mathrm{sr,pr}\}} + K_x x_e^{\{\mathrm{sr,pr}\}} \\ \omega_{\mathrm{tgt}}^{\mathrm{VL}} + v_{\mathrm{tgt}}^{\{\mathrm{sr,pr}\}}\left(K_y y_e^{\{\mathrm{sr,pr}\}} + K_\theta \sin\theta_e^{\{\mathrm{sr,pr}\}}\right) \end{bmatrix}, \tag{2.13}$$

where $K_x, K_y, K_\theta$ are the feedback gains and $\boldsymbol{o}_{\mathrm{fb}}$ is the observation for calculating the feedback controller output (refer to Table 2.1). In our work, the feedback gains are decided by the Tree-structured Parzen Estimator Approach (TPE) [51] (a Bayesian optimization method) so that the reward for one episode is maximized. For the implementation of TPE, we utilize Optuna [52].

This feedback controller is locally stable from the perspective of Lyapunov function, but it is not enough for transporting the logistics cart, as discussed later in section 2.4.

### 2.3.5 Residual reinforcement learning

Here, we introduce our cooperative logistics cart transportation controller using residual reinforcement learning.

In residual reinforcement learning, the control output is calculated as the sum of the base controller output and the reinforcement learning controller output. The output of the reinforcement learning controller is defined as each robot's linear velocity $v_{\mathrm{res}}^{\{\mathrm{sr},\mathrm{pr}\}}$ and rotation velocity $\omega_{\mathrm{res}}^{\{\mathrm{sr},\mathrm{pr}\}}$; thus, the reinforcement learning controller outputs four dimensions of action:

$$\pi_{\mathrm{res}}\left(\boldsymbol{o}_{\mathrm{fb}}, \boldsymbol{o}_{\mathrm{rl}}\right) \rightarrow \left[v_{\mathrm{res}}^{\mathrm{sr}},\ \omega_{\mathrm{res}}^{\mathrm{sr}},\ v_{\mathrm{res}}^{\mathrm{pr}},\ \omega_{\mathrm{res}}^{\mathrm{pr}},\right]^{\top}, \tag{2.14}$$

where $\boldsymbol{o}_{\mathrm{rl}}$ is the observation that is used for reinforcement learning. Finally, each robot's control output is calculated as

$$\pi\left(\boldsymbol{o}_{\mathrm{fb}}, \boldsymbol{o}_{\mathrm{rl}}\right) = \left[v^{\mathrm{sr}},\ \omega^{\mathrm{sr}},\ v^{\mathrm{pr}},\ \omega^{\mathrm{pr}}\right]^{\top} = \pi_{\mathrm{fb}}\left(\boldsymbol{o}_{\mathrm{fb}}\right) + \pi_{\mathrm{res}}\left(\boldsymbol{o}_{\mathrm{fb}}, \boldsymbol{o}_{\mathrm{rl}}\right), \tag{2.15}$$

where $\pi_{\mathrm{fb}}\left(\boldsymbol{o}_{\mathrm{fb}}\right) = [\pi_{\mathrm{fb}}^{\mathrm{sr}}\left(\boldsymbol{o}_{\mathrm{fb}}\right), \pi_{\mathrm{fb}}^{\mathrm{pr}}\left(\boldsymbol{o}_{\mathrm{fb}}\right)]^{\top}$.

### 2.3.6 Learning algorithm

We use Twin Delayed DDPG (TD3) [7] as the learning algorithm. We considered using another promising method, Soft-Actor Critic [6], which utilizes the stochastic action and maximizes the trade-off objective between the expected sum of rewards and the entropy of the stochastic action. However, in residual reinforcement learning, we think that exploitation is more important than exploration, so we adopt TD3, which is exploration by fixed distributed noise.

The reinforcement learning objective is to maximize the expected reward, defined as

$$J(\phi) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}\left[\sum_{i=0}^{T} \gamma^i r_i\right], \tag{2.16}$$

where $\pi$ is the policy, $\phi$ is the parameters in the policy, and $\gamma$ is the time discount factor. Also, the state-action value function Q is defined as

$$Q^{\pi}_{\theta}(s_t, a_t) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi} \left[ \sum_{i=t}^{T} \gamma^i r_i | s_t, a_t \right], \tag{2.17}$$

which means that the action $a_t$ causes the time-discounted and cumulated reward when the state is $s_t$. Q-learning minimizes the temporal differential error (TD error). Q-function is written as follows based on a Bellman equation:

$$Q^{\pi}_{\theta}(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1}, a_{t+1}} \left[ Q^{\pi}_{\theta}(s_{t+1}, a_{t+1}) \right]. \tag{2.18}$$

We utilize multi-step learning, which is a method for stabilizing the learning process by using the forward-view $n$-step rewards. In Q-learning, the parameters of the target network are defined as $\bar{\theta}$ and the target value is defined as

$$y = r_t + \gamma r_{t+1} + \cdots + \gamma^n Q^{\pi}_{\bar{\theta}}(s_{t+n}, a_{t+n}). \tag{2.19}$$

Therefore, the objective for minimizing the TD error is written as

$$L_Q(\theta) = \frac{1}{N} \sum_i \text{HuberLoss} \left( y - Q^{\pi}_{\theta}(s_i, a_i) \right), \quad \text{HuberLoss}(\delta) = \begin{cases} \delta^2/2 & (|\delta| < 1) \\ |\delta| - 0.5 & (otherwise) \end{cases}, \tag{2.20}$$

where Huber loss is used for robustness to outliers. Also, $\bar{\theta}$, which is the parameter of the target network, is updated for tracking $\theta$. Concretely, $\bar{\theta}$ is updated as $\bar{\theta} \leftarrow (1 - \tau)\,\bar{\theta} + \tau\theta$, where $\tau$ is the hyper-parameter and the smaller $\tau$ is, the more slowly the target network is updated. Additionally, TD3 utilizes two methods for Q-learning: (i) clipped double Q-learning, where two Q-functions are used and a small value of the two Q-functions is adopted as the target value for avoiding overestimation, and (ii) target policy smoothing, where noise is added to the action when the Q value of the target network in the target value calculation is calculated for stabilizing learning.

Then, for optimizing the policy, we utilize the deterministic policy gradient method proposed by Silver et al. [53]. The gradient is calculated for maximizing $J(\phi)$, so it is obtained by using the Q-function as follows:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s \sim p_\pi} \left[ \nabla_a Q(s, a)|_{a=\pi(\cdot|s)} \nabla_\phi \pi_\phi(\cdot|s) \right]. \tag{2.21}$$

The basic policy gradient method updates the Q-function and the policy in one learning step, but TD3 makes the policy update once every multiple steps of learning, which is called a delayed policy update. In addition, we utilize prioritized experience replay [54] and distributed learning for learning efficiently.

Prioritized experience replay is a method where the experience that has a large TD error is sampled with priority, which makes it possible to learn efficiently. The priority is defined as

$$p_i = \left| y - Q_\theta^\pi(s_i, a_i) \right| + \epsilon, \tag{2.22}$$

where $\epsilon$ is a small value for avoiding the priority becoming zero. Then, the probability of the $i$-th experience is

$$P(i) = \frac{p_i^\alpha}{\sum_i p_i^\alpha}, \tag{2.23}$$

where $\alpha$ is the hyper-parameter. Also, importance sampling is applied for compensating the biased sampling. The weight for the importance sampling is defined as

$$w_i = \left( \frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta, \tag{2.24}$$

where $\beta$ is the hyper-parameter for deciding the degree of importance sampling and its bias is corrected if $\beta = 1$. $N$ is the size of replay memory. Also, for stabilizing, the weight is normalized to $1/\max_i w_i$. By using equation (23), the weight is written simply as

$$w_i \leftarrow \frac{w_i}{w_{\max}} = \left( \frac{1}{N} \cdot \frac{\sum_i p_i^\alpha}{p_i^\alpha} \right)^\beta \left( \frac{1}{N} \cdot \frac{\sum_i p_i^\alpha}{p_{\min}^\alpha} \right)^{-\beta} = \left( \frac{p_{\min}}{p_i} \right)^{\alpha\beta}. \tag{2.25}$$

For distributed learning, eight actor modules and one learner module are executed in parallel. The algorithm using prioritized experience replay and distributed learning is similar to the one in [55]. In [55], a large number of actors ($\sim 256$) is executed in parallel, so in the actor module, the priority is calculated. However, our implementation does not calculate the priority in the actor module because paralleled actor modules are not large. The pseudo codes are shown in Algorithms 3 and 4.

**Algorithm 3** Actor

```
Input: index of Actor $i$, Learner, policy update interval $K$, multi-step learning size $n$,
    standard deviation of action noise $\sigma$, time discount factor $\gamma$, feedback policy $\pi_{\text{fb}}$:
 1: Initialize policy network $\pi_\phi$
 2: Initialize Environment env
 3: Initialize local replay memory $LR$
 4: for episode = 0, 1, . . . do
 5:     env.reset($i$)
 6:     $s_o \leftarrow$ env.observation()
 7:     for $t = 0, 1, \ldots$ do
 8:         if $t$ mod $K = 0$ then $\pi_\phi \leftarrow$ Learner.copy_policy() end if
 9:         $a_t \leftarrow \pi_\phi(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$
10:         $a'_t = \pi_{\text{fb}}(s_t) + a_t$
11:         $r_t, done$ = env.step($a'_t$)
12:         $s_{t+1} \leftarrow$ env.observation()
13:         $LR$.add($s_t, a_t, r_t, s_{t+1}, done$)
14:         if $LR$.size() $\geq n$ or done then
15:             $p \leftarrow$ Learner.$p_{\text{max}}$
16:             $(s_t, a_t, \sum_{\tau=0}^{n} \gamma^\tau r_{t+\tau-1}, s_{t+n}, done) \leftarrow LR$.make_n_step_return()
17:             Learner.add_to_replay_memory($p, (s_t, a_t, \sum_{\tau=0}^{n} \gamma^\tau r_{t+\tau-1}, s_{t+n}, done)$)
18:         end if
19:         if done then break end if
20:     end for
21: end for
```

**Algorithm 4** Learner

```
Input: standard deviation of target policy smoothing noise $\tilde{\sigma}$, noise clipping coefficient $c$,
    time discount factor $\gamma$, multi-step return size $n$,target network update coefficient $\tau$,
    policy update frequency $F$, mini-batch size $N$:
Output: optimal policy $\pi^*$
 1: Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and policy network $\pi_\phi$
 2: Initialize target networks $\bar{\theta}_1, \bar{\theta}_2, \bar{\phi} \leftarrow \theta_1, \theta_2, \phi$
 3: Initialize ReplayMemory $R$
 4: $p\mathrm{max} \leftarrow 1.0$
 5: for $i = 0, 1, \ldots$ do
 6:   mini-batch($p, w, (s, a, r, s', done)$) $\leftarrow$ $R$.prioritized_sample()
 7:   $a' \leftarrow \pi_\phi(\cdot|s) + \epsilon, \quad \epsilon \sim \mathrm{clip}(\mathcal{N}(0, \tilde{\sigma}), -\mathrm{c}, \mathrm{c})$
 8:   $y \leftarrow r + \gamma^n \min_{j=1,2} Q_{\bar{\theta}_j}(s', a')$
 9:   update $p$ by TD error
10:   if $p > p\mathrm{max}$ then $p\mathrm{max} \leftarrow p$ end if
11:   update critics $\theta_j$ by $L_Q(\theta_j) = N^{-1} \sum w\mathrm{HuberLoss}(y, Q_{\theta_j}(s, a))$
12:   if $i$ mod $F = 0$ then
13:     update policy $\phi$ by $J_\pi(\phi) = N^{-1} \sum Q_{\theta_1}(s, \pi_\phi(\cdot|s))$
14:     update target networks:
        $\bar{\theta}_j \leftarrow \tau\bar{\theta}_j + (1 - \tau)\theta_j$
        $\bar{\phi} \leftarrow \tau\bar{\phi} + (1 - \tau)\phi$
15:   end if
16: end for
```

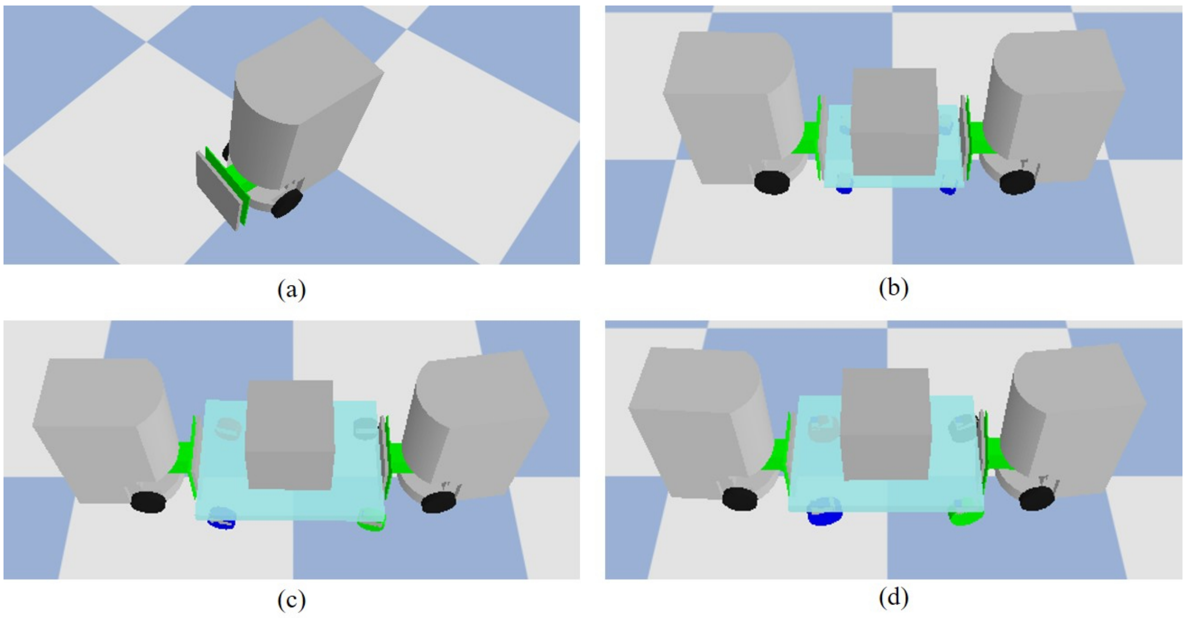


**Figure.2.4:** (a) Robot model. (b) Logistics cart model (size: 0.6 m). (c) Logistics cart model (size: 0.795 m). (d) Logistics cart model (size: 0.8 m).

## 2.4 Experiments

### 2.4.1 Simulation setting

We utilize Pybullet [18] which is python module for constructing the simulation environment. Three logistics cart sizes $\{0.6, 0.785, 0.8\}$ m are prepared. These models and the robot model, which are based on real-world logistics carts and robots, are shown in Fig. 2.4. The weight of the load is sampled from uniform distribution between 120 and 130 kg at the start of the episode if the size of the logistics cart is $\{0.785, 0.8\}$ m or between 100 and 110 kg if the size of the logistics cart is 0.6 m. The range of the curvature is $[-0.06, 0.66]$ $\mathrm{m}^{-1}$ and this is divided into eight areas uniformly. Each area is allocated to one of the actor modules. The value of the curvature is sampled from uniform distribution in the allocated area at the start of the episode. The frequency of the observation and control is 10 Hz. The termination conditions of the episode are that 30 s has passed or one or more of the following conditions have persisted for 1 s.

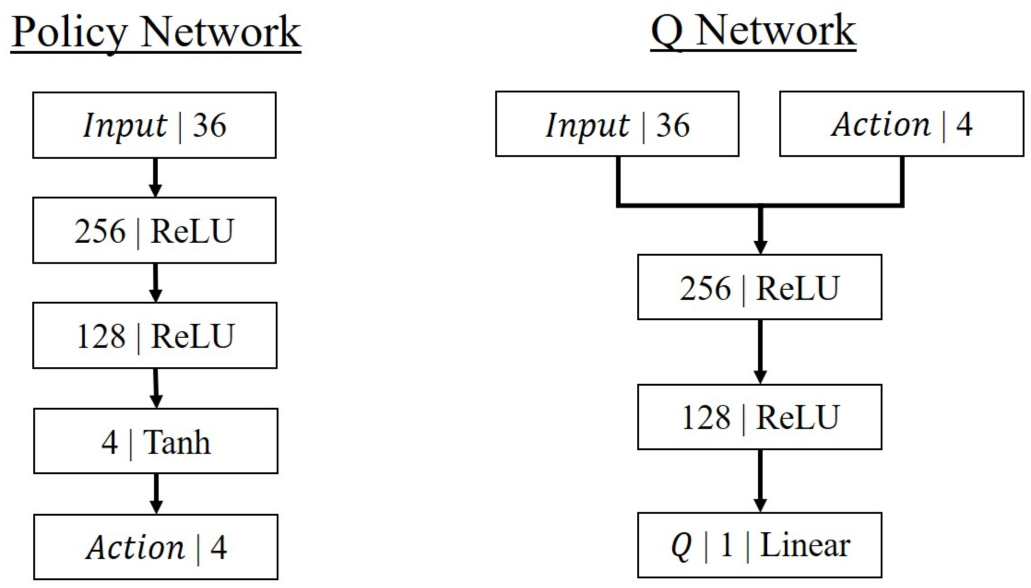


**Figure.2.5:** Network architecture.

- The deviation from the trajectory is larger than 0.4 m
- The orientation deviation from the reference is larger than $\pi/4.5$ rad
- Moving distance along the trajectory is smaller than 0.005 m
- At least one of the robots has been out of a certain area (refer to equation (2.4))

If the episode satisfies at least one of the termination conditions (except that time is up), the reinforcement learning controller receives a reward of $-10$.

## 2.4.2 Implementation details of proposed methodology

The output of the feedback controller is calculated by equation (13). The target velocity of the feedback controller $v_{\mathrm{tgt}}^{\mathrm{VL}}$ is set to 0.5 m/s. The feedback gains are set to the same values as the feedback controller in section 2.4.3. The policy network and the Q network architectures are shown in Fig. 2.5. In training, we use the Adam [56] optimizer, and hyper-parameters are set as follows: learning rate of $3 \times 10^{-4}$, replay memory size of $10^5$, target network update coefficient $\tau$ of 0.005, time discount factor $\gamma$ of 0.99, policy update interval $K$ of 100, multi-step learning size $n$ of 4, batch size of 256, standard deviation of action noise $\sigma$ of $0.1\times(a_{\max} - a_{\min})$, standard deviation of target policy smoothing noise $\bar{\sigma}$ of $0.2\times(a_{\max} - a_{\min})$, noise clipping coefficient $c$ of $0.5 \times (a_{\max} - a_{\min})$, policy update frequency $F$ of 2, and the hyper-parameters of prioritized experience replay $\alpha, \epsilon, \beta$ of $0.6, 0.01, 0.4$.

### 2.4.3 Baselines

#### RL-only

RL-only has each robot's linear velocity and angular velocity, both of which are four dimensions of continuous action. The network architecture and hyper-parameters are the same as the residual reinforcement learning implementation. RL-only does not use knowledge and instead learns from scratch.

#### Feedback controller

The output of the feedback controller is calculated by equation (2.13). The target velocity of the logistics cart is set to 0.5 m/s and the target angular velocity of the logistics cart is set to $\omega^{\mathrm{VL}}_{\mathrm{tgt}} = \rho v^{\mathrm{VL}}_{\mathrm{tgt}} = 0.5\rho$, where $\rho$ is the curvature. The feedback gains $K_x, K_y, K_\theta$ are explored in the range [0, 10] by TPE with 100 steps and are set to values that maximize the expected total reward of one episode. The expected total reward for one episode is calculated by running 15 episodes that are a combination of three logistics cart sizes $\{0.6, 0.785, 0.8\}$ m and five curvatures $\{0, 0.15, 0.3, 0.45, 0.6\}$ m$^{-1}$. The weight of the load is set to 120 kg at the start of the episode if the size of the logistics cart is $\{0.785, 0.8\}$ m or to 100 kg if the size of the logistics cart is 0.6 m. As a result, the feedback gains are set to $(K_x, K_y, K_\theta) = (3.82, 6.02, 1.43)$ and these are fixed during evaluation.

### 2.4.4 Effect of Residual RL's action scale

Three action scales of residual reinforcement learning are compared. When the action scale is 1.0, the linear velocity range of residual reinforcement learning is $[-1.0, 1.0]$ m/s and its rotation velocity is $[-\pi/3, \pi/3]$ rad/s. The results of five trials of each action scale are shown in Fig. 2.6. The left side of the figure plots the smoothed reward during training. The right side plots the results evaluated by executing 15 episodes that are a combination of three logistics cart sizes $\{0.6, 0.785, 0.8\}$ m and five curvatures $\{0, 0.15, 0.3, 0.45, 0.6\}$ m$^{-1}$ with models that are stored for each of $10^5$ experiences. The rule for sampling the weight of the load is the same as in section 2.4.3. The results on the left are evaluated with exploration

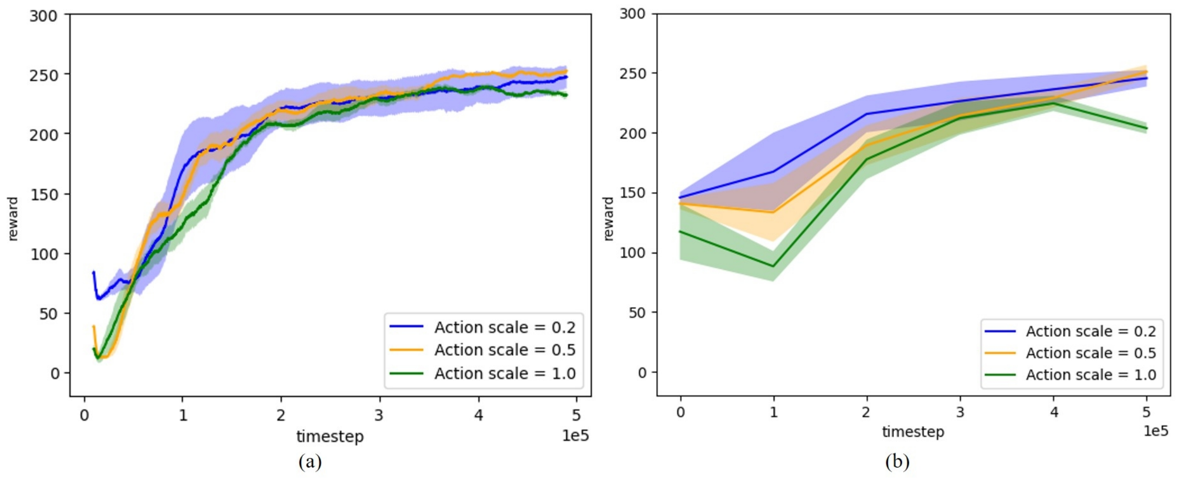


**Figure.2.6:** Learning curves of action scales' residual RL (a) with exploration noise and (b) without exploration noise.

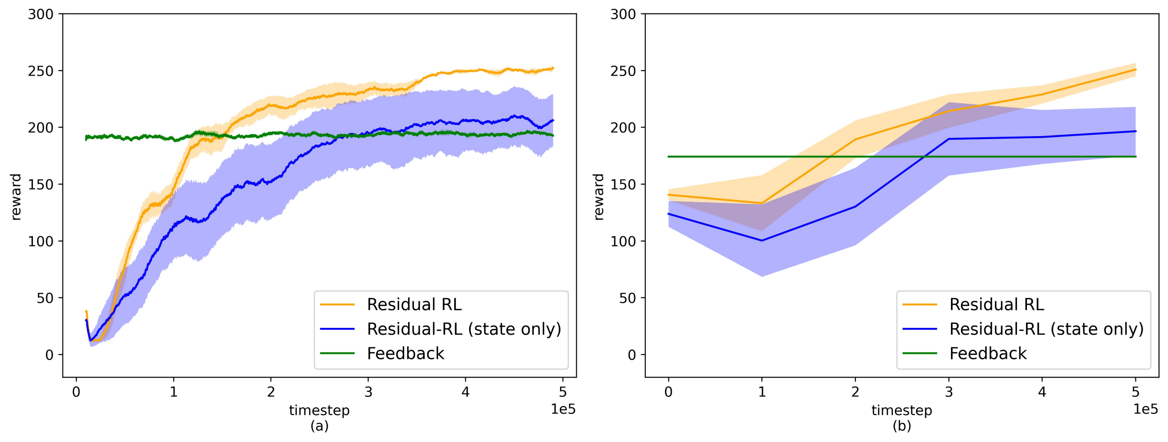


**Figure.2.7:** Learning curve of residual RL, residual RL (state only), and feedback controller (a) with exploration noise and (b) without exploration noise.

noise and those on the right without exploration noise. The lines mean the average and the shaded region represents half standard deviation.

From Fig. 2.6, we can see that the large action scale delays improvement of the policy. The maximum reward without exploration noise is obtained when the action scale is 0.5 and the timestep is $5 \times 10^5$; therefore, the action scale is set to 0.5.

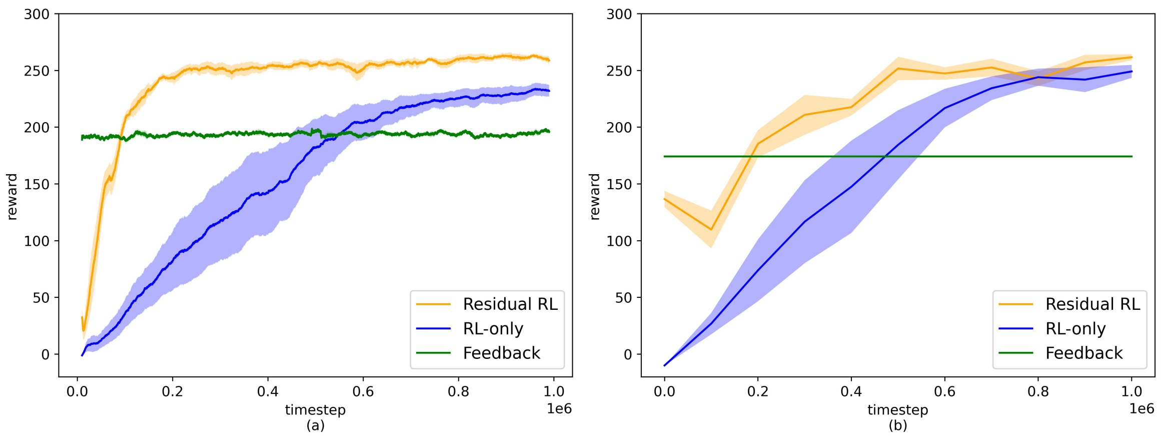


**Figure.2.8:** Learning curve of residual RL, RL-only, and feedback controller (a) with exploration noise and (b) without exploration noise.

### 2.4.5 Effect of time derivative component

We verify what happens when the velocity component is removed from the input to residual reinforcement learning. Fig. 2.7 shows the results of learning. The left side of the figure plots the smoothed reward during training. The evaluation method for the curve on the right is the same as in 2.4.4. The results of the feedback controller are also plotted, where those on the right side do not have a shaded area because Pybullet is the deterministic simulator.

Figure shows that the added information about the time derivative component contributes to the improvement of the performance. In addition, the standard deviation of residual RL is smaller than that of residual RL (state only).

### 2.4.6 Feedback Controller vs RL-only vs Residual RL

Fig. 2.8 shows the learning curves. The left side of the figure shows the training results and the right side shows the results of running episodes without exploration noise.

Residual reinforcement learning achieves more efficient learning and a smaller standard deviation than reinforcement learning. Final reward of residual reinforcement learning is slightly higher than that of reinforcement learning. Also, from the results on the right, it is clear that the reinforcement learning controllers acquire a higher reward than the feedback controller.

**Table.2.2:** Results of simulation experiments by the controllers at $5 \times 10^5$ steps. Feedback: Feedback controller, RL: RL-only controller, RRL: Residual RL controller.

| | | Reward | | | Average velocity [m/s] | | | Trajectory error [m] | | | Orientation error [rad] | | | Holding ratio | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Logistics cart size [m] | Curvature | Feedback | RL | RRL | Feedback | RL | RRL | Feedback | RL | RRL | Feedback | RL | RRL | Feedback | RL | RRL |
| | 0 | **284.70** | 203.19 | 264.12 | **0.491** | 0.404 | **0.491** | **0.0039** | 0.0324 | 0.0089 | **0.0032** | 0.0341 | 0.0299 | 0.977 | 0.979 | **0.980** |
| | 0.15 | **278.65** | 214.51 | 269.41 | 0.487 | 0.427 | **0.488** | **0.0046** | 0.0405 | 0.0085 | **0.0098** | 0.0323 | 0.0176 | 0.977 | **0.993** | 0.982 |
| 0.6 | 0.30 | 257.99 | 198.49 | **271.61** | 0.480 | 0.436 | **0.482** | 0.0108 | 0.0543 | **0.0067** | 0.0288 | 0.0630 | **0.0148** | 0.977 | **0.996** | 0.993 |
| | 0.45 | **252.2** | 181.61 | 211.27 | **0.470** | 0.423 | 0.449 | **0.0100** | 0.0461 | 0.0167 | 0.0337 | 0.1008 | **0.0202** | 0.977 | **0.993** | 0.888 |
| | 0.60 | -9.61 | 164.61 | **251.02** | - | 0.399 | **0.456** | - | 0.0477 | **0.0118** | - | 0.1203 | **0.0187** | - | 0.975 | **0.999** |
| | 0 | **282.05** | 211.41 | 264.88 | 0.492 | 0.402 | **0.496** | **0.0028** | 0.0275 | 0.0158 | **0.0096** | 0.0237 | 0.0188 | 0.973 | **0.992** | 0.976 |
| | 0.15 | **273.60** | 213.04 | 269.20 | 0.489 | 0.423 | **0.491** | **0.0070** | 0.0455 | 0.0094 | **0.0153** | 0.0330 | 0.0206 | 0.977 | **0.996** | 0.986 |
| 0.795 | 0.30 | 242.86 | 194.21 | **262.08** | 0.470 | 0.428 | **0.480** | 0.0194 | 0.0596 | **0.0107** | 0.0333 | 0.0478 | **0.0222** | 0.977 | **0.993** | 0.992 |
| | 0.45 | -9.60 | 169.65 | **251.80** | - | 0.398 | **0.465** | - | 0.0289 | **0.0145** | - | 0.0502 | **0.0229** | - | 0.916 | **0.997** |
| | 0.60 | -9.61 | 125.78 | **240.78** | - | 0.385 | **0.447** | - | 0.0820 | **0.0131** | - | 0.1010 | **0.0291** | - | 0.925 | **1.000** |
| | 0 | **276.67** | 212.79 | 211.16 | **0.488** | 0.398 | 0.449 | **0.0045** | 0.0237 | 0.0111 | **0.0127** | 0.0214 | 0.0141 | 0.973 | **0.991** | 0.777 |
| | 0.15 | 265.06 | 207.35 | **266.35** | **0.486** | 0.417 | **0.486** | 0.0138 | 0.0474 | **0.0091** | **0.0157** | 0.0377 | 0.0206 | 0.977 | **0.997** | 0.979 |
| 0.8 | 0.30 | 241.28 | 203.45 | **258.57** | 0.469 | 0.422 | **0.475** | 0.0186 | 0.0617 | **0.0119** | 0.0354 | 0.0421 | **0.0217** | 0.97 | **0.989** | 0.988 |
| | 0.45 | -7.34 | 177.34 | **248.44** | - | 0.406 | **0.460** | - | 0.0472 | **0.0151** | - | 0.0882 | **0.0237** | - | 0.932 | **0.993** |
| | 0.60 | -9.66 | 89.45 | **235.47** | - | 0.360 | **0.440** | - | 0.1055 | **0.0129** | - | 0.1508 | **0.0333** | - | 0.950 | **0.995** |

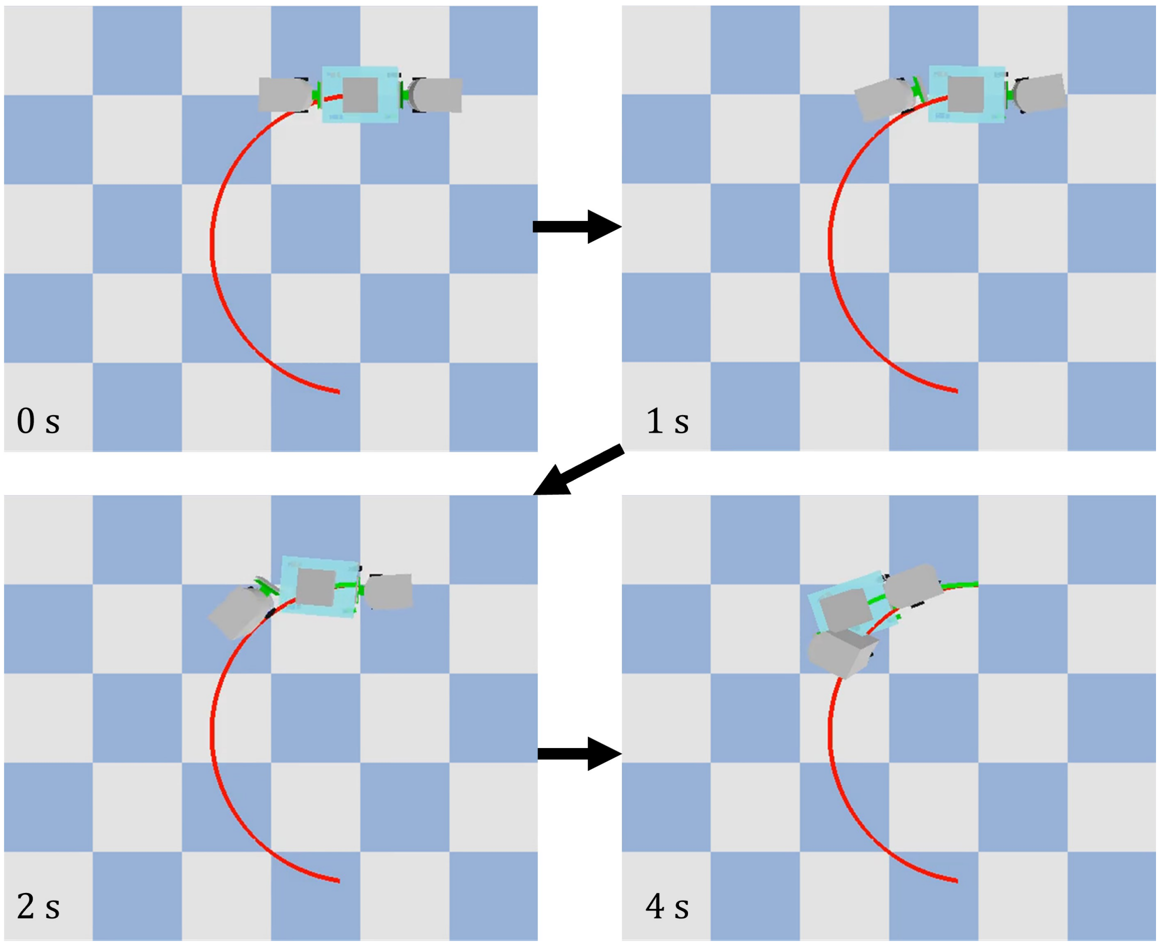


**Figure.2.9:** Example of feedback controller simulation experiment results. Logistics cart size is 0.8 m and trajectory curvature is 0.6 $m^{-1}$. Red line is the given trajectory and green line is actual trajectory of the logistics cart.

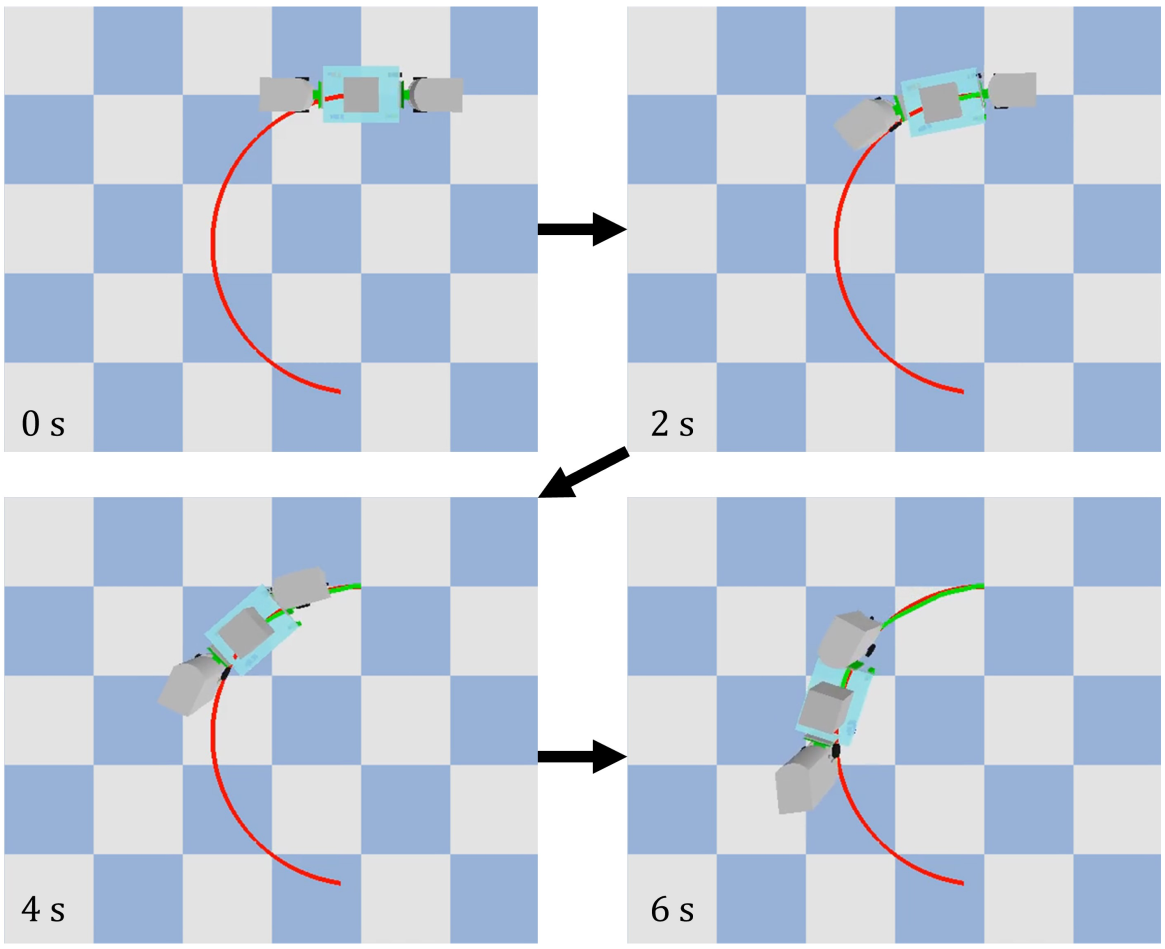


**Figure.2.10:** Example of residual RL controller simulation experiment results. Logistics cart size is 0.8 m and trajectory curvature is 0.6 $\mathrm{m}^{-1}$. Red line is the given trajectory and green line is actual trajectory of the logistics cart.

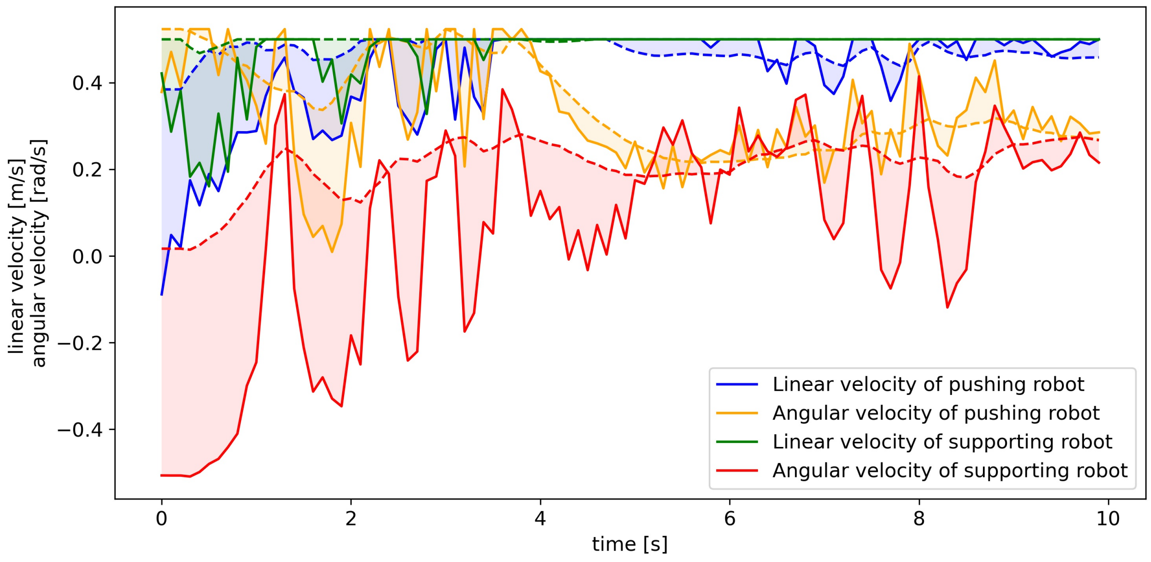
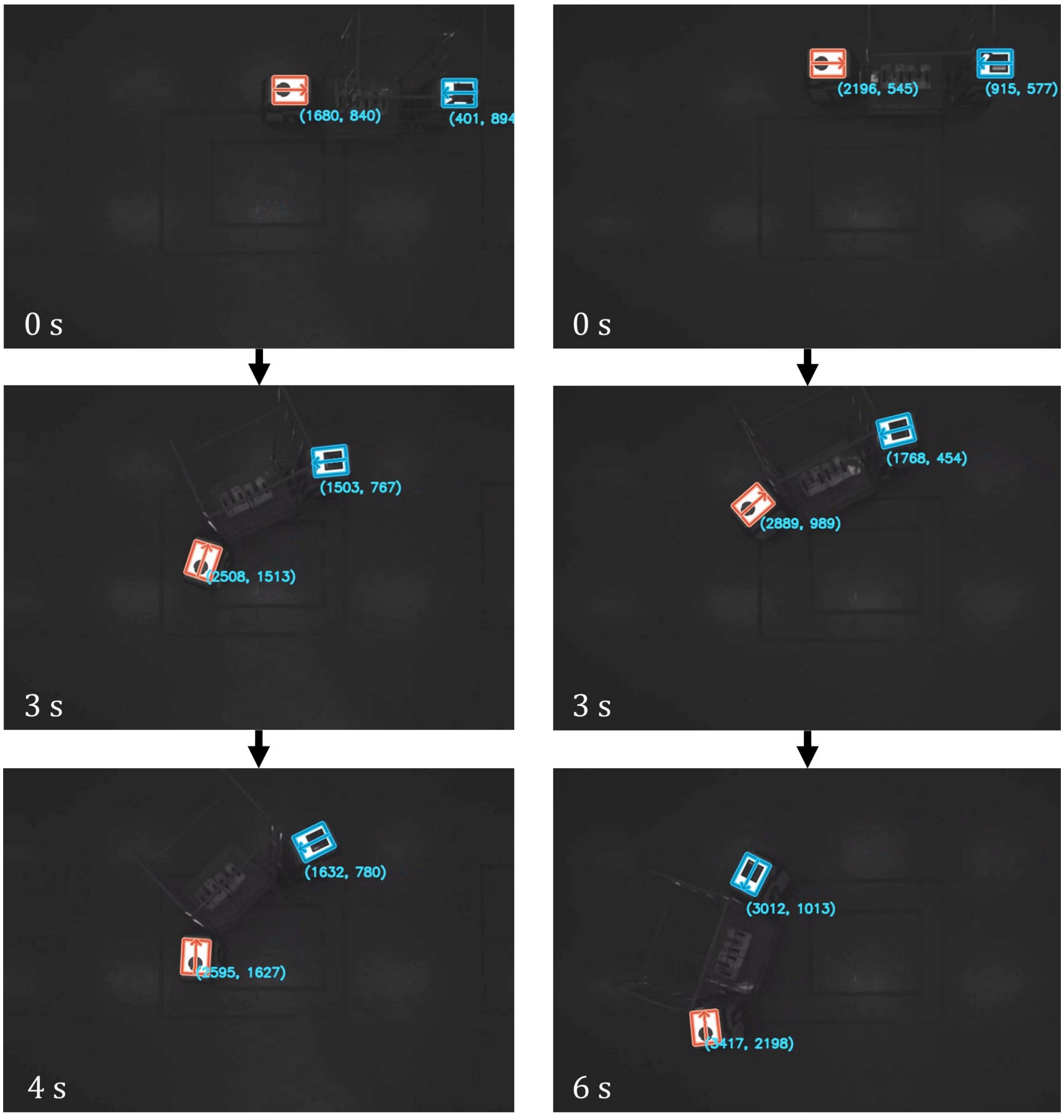


**Figure.2.11:** History of the residual RL controller's output for 10 s in the simulation experiment, where logistics cart size is 0.8 m and trajectory curvature is 0.6 m$^{-1}$. Dashed lines mean the output of the feedback controller and shaded regions mean the amount of the compensation by residual RL.

**Table.2.3:** Results of real-world experiments: (the number of successes) / (the number of experiments).

| Method | Logistics cart size [m] | Curvature [/m] | | | | |
|---|---|---|---|---|---|---|
| | | 0 | 0.15 | 0.30 | 0.45 | 0.60 |
| Feedback controller | 0.6 | 2/2 | 2/2 | 2/2 | 2/2 | 0/2 |
| | 0.795 | 2/2 | 2/2 | 2/2 | 1/2 | 0/2 |
| | 0.8 | 2/2 | 2/2 | 2/2 | 1/2 | 0/2 |
| Residual RL controller | 0.6 | 2/2 | 2/2 | 2/2 | 2/2 | **2/2** |
| | 0.795 | 2/2 | 2/2 | 2/2 | **2/2** | **2/2** |
| | 0.8 | 2/2 | 2/2 | 2/2 | **2/2** | **2/2** |



(a) (b)

**Figure.2.12:** Example of real-world experiment results. (a) Feedback controller, where logistics cart size is 0.8 m and trajectory curvature is 0.6 $m^{-1}$. (b) Residual RL controller, where logistics cart size is 0.8 m and trajectory curvature is 0.6 $m^{-1}$.

## 2.4.7 Simulation experiments

Next, we compare the reinforcement learning controllers and the feedback controller in a simulation environment. For this evaluation, each model of the reinforcement learning controllers at $5 \times 10^5$ steps is utilized. The expected total reward for one episode is calculated by running 15 episodes that are a combination of three logistics cart sizes $\{0.6, 0.785, 0.8\}$ m and five curvatures $\{0, 0.15, 0.3, 0.45, 0.6\}$ m$^{-1}$. The weight of the load is set to 120 kg at the start of the episode if the size of the logistics cart is $\{0.785, 0.8\}$ $m$ or to 100 kg if the size of the logistics cart is 0.6 m.

### Metrics

We evaluate the simulation results with the following five metrics. **Reward** is the total reward for one episode calculated by equation (6). **Average velocity** is the average velocity of the logistics cart for one episode. **Trajectory error** is the average deviation from the trajectory for one episode, which is defined as the shortest distance between the logistics cart and the trajectory. **Orientation error** is the average deviation from the reference orientation that is the tangential direction of the trajectory for one episode. **Holding ratio** is the average ratio of holding the logistics cart with the robots for one episode. If the robots stay in area defined by equation (4), holding is considered to be maintained.

### Simulation results

The results of the simulation experiments are shown in Table 2.2. Metrics without rewards are set to "–" if the reward is smaller than 0, as metrics without rewards cannot be evaluated precisely when the logistics cart is not transported far enough. The results of the learning-based controllers are represented by the average values of five learned models.

The feedback controller cannot make the logistics cart track the trajectory when the curvature is large, so its reward is lower than that of the learning-based controller. Also, the residual reinforcement learning controller improves the transportation performance compared to the feedback controller in many conditions. The reinforcement learning controller only performed better than the other methods in Holding ratio.

Figs. 2.9 and 2.10 show snapshots of the simulation experiments. The feedback controller cannot keep holding the logistics cart, while in contrast, the residual reinforcement learning controller keeps holding it and achieves stable transportation.

Fig. 2.11 shows the residual reinforcement learning controller's output in simulation experiments. The amount of compensation of linear velocity is large in the beginning of the movement. That of angular velocity is large throughout the episode.

### 2.4.8 Real-world experiments

Next, in real-world experiments, we compare the feedback controller and the residual reinforcement learning controller that obtained a higher reward than RL-only controller. We use the residual reinforcement learning controller learned in the simulation environment with no additional parameter tuning.

We run the experiment twice for each of 15 conditions that are a combination of three logistics cart sizes $\{0.6, 0.785, 0.8\}$ m and five curvatures $\{0, 0.15, 0.3, 0.45, 0.6\}$ $\text{m}^{-1}$. The weight of the load is set to 120 kg if the size of the logistics cart is $\{0.785, 0.8\}$ m or to 100 kg if the size of the logistics cart is 0.6 m. Also, the error rate of the robot localization by camera on ceiling is up to 1%, which is evaluated by a total station as a measurement instrument.

#### Real-world results

The results of the real-world experiments are shown in Table 2.3. The residual reinforcement learning controller has a better performance for the logistics cart transportation than the feedback controller, the same as in the simulation experiments. A snapshot of the real-world experiment is shown in Fig. 2.12. In this figure, when the curvature is 0.6 $\text{m}^{-1}$, the feedback controller cannot transport the logistics cart because the holding mechanism becomes unfastened. This behavior is similar to the results of the simulation experiments shown in Fig. 2.12. In contrast, the residual reinforcement learning controller can keep holding the logistics cart and achieves stable transportation.

## 2.5 Discussion

### 2.5.1 Effect of residual RL's action scale

The larger the action scale becomes, the worse the learning efficiency gets, as the high cost of exploration makes the action space large. In contrast, a small action scale reduces the better local minima in the action space, so the reward may be small. The fact that the reward becomes small if the action scale is 0 reinforces this consideration. Thus, the appropriate action scale has the potential for accelerating learning speed. In our work, we consider that robots basically do not need to move backward, so we can adopt a small action scale. As a result, the action scales 0.2 and 0.5 acquire the highest reward, with 0.5 obtaining a slightly higher one.

### 2.5.2 Effect of time derivative component

Fig. 2.7 indicates that the performance of the reinforcement learning controller is deteriorated by the lack of the time derivative information. This means the importance of that information in achieving proper logistics cart transport. Also, the standard deviation increases by removing the time derivative component. This results means that the learning becomes unstable. We consider that the problem is not regarded as an MDP because of the lack of time derivative information, therefore, it is difficult to solve as a reinforcement learning problem.

### 2.5.3 Learning curve of Residual RL and RL-only

Residual reinforcement learning has a better learning efficiency than reinforcement learning. One reason for this result is that utilizing the feedback controller as a base controller prompts the gathering of more varied experiences. Also, residual reinforcement learning has the smaller standard deviation than reinforcement learning. This means that residual reinforcement learning is stable learning method. The reason of this result is that it is hard to stagnate on an unfavorable local optimum because of base controller. These advantages of residual reinforcement learning are important because they accelerate learning speed and reduce the number of trials.

From Fig. 2.8, in the early period of learning, the residual reinforcement learning controller's reward is lower than that of the feedback controller. This is presumably because the Q-function does not acquire an accurate model, so it cannot evaluate the controller precisely and the controller cannot be improved. One idea for coping with this problem is that initial output of residual is set to 0. However, the results in previous research [28] indicates that this idea does not avoid the deterioration of the controller. We tried this idea in our environment, although it did not avoid this deterioration. The method to avoid this deterioration is future work.

### 2.5.4 Simulation experiments

Table 2.2 shows that the residual reinforcement learning controller has a better performance than both the reinforcement learning controller and the feedback controller under many conditions. In particular, when the curvature is large, the feedback controller cannot transport the logistics cart, while the residual and reinforcement learning controllers can.

The reason the learning-based controller is better than the feedback controller is that it can utilize information that is not used by the feedback controller and its output is nonlinearized. The feedback controller utilizes only each robot's position and orientation, whereas the learning-based controller utilizes the additional information shown in Table 2.1.

On the other hand, in some cases with a small curvature's trajectory, the feedback controller has a better performance than the residual reinforcement learning controller. This means that the residual reinforcement learning controller deteriorates the policy when it acquires the control law for transporting the logistics cart with a large curvature's trajectory. Avoiding this deterioration and creating a controller that has a better performance than the feedback controller in all available states remain issues for future work.

From Fig. 2.9, the logistics cart cannot track the trajectory and so orientation error is accumulated. Then, when the robots try to modify this error, they cannot keep holding the logistics cart. This occurs because the feedback controller controls the robots' state and does not consider the logistics cart's state. Another reason for this result is that the feedback controller makes only current errors become 0. In contrast, the residual reinforcement learning controller learns the policy for reducing the errors of the logistics cart and considers the long-period accumulated reward, so it can keep holding the logistics cart, as shown in Fig. 2.10.

Fig. 2.11 shows that residual reinforcement learning compensates the output of the feedback controller. The reason the amount of compensation of angular velocity is large is that the residual helps to change the orientation of the logistics cart for making it track the trajectory.

### 2.5.5 Real-world experiments

The results of the real-world experiments show that the residual reinforcement learning controller learned in simulation environments can be transferred to real-world control. However, the results of the real-world experiments differ slightly from the results of the simulation experiments. This difference presumably stems from some gap between the simulation and the real world, e.g., the friction of the wheels of the logistics cart, the gap of the holding position of the robots, and/or observation noise. The results of the real-world experiments suggest that the residual reinforcement learning controller absorbs the gap between the simulation and the real world. Randomization may be one cause of this, e.g., exploration noise and the noise added to the load weight. While reducing the gap between the simulation and the real world was not considered in our work, it is important to clarify how to transfer the reinforcement learning controller learned in a simulation environment to real-world tasks more robustly.

## 2.6 Summary

We proposed a system for cooperative logistics cart transportation with a residual reinforcement learning controller. The proposed controller is more sample efficient than a reinforcement learning controller trained from scratch and has a higher performance than the feedback controller. We showed that using simulation reduces the cost of gathering experiences, and the results of real-world experiments suggest that the residual reinforcement learning controller learned in a simulation environment can be transferred to real-world control.

As future work, we will investigate how to make the controller higher performance than the feedback controller in all available states and reduce the difference between simulation and real world.

# Chapter 3

# Efficient Reinforcement Learning for Delayed Feedback Environments

## 3.1 INTRODUCTION

In the context of robot control, a lot of research on reinforcement learning in simulation environments of real-world control have been proposed. In a simulation environment, an agent obtains an observation and applies a control output to environment instantaneously. However, in the real world, delayed feedback from an environment can occur, for example, due to preprocessing for an observation, communication delays, calculating a control output, control via wireless network communication, and centralized control in a large system. In [57], it was shown that observation delay and control delay are equivalent mathematically; therefore, we consider only observation delay to be the sum of control delay and observation delay in a real-world system. In [58], it was shown experimentally that it is one of the most important factors for transferring RL controller training in a simulation environment to real-world execution to incorporate the effect of delayed feedback into the simulation environment.

However, an RL controller's performance and learning efficiency degrade because of delayed feedback. The longer delayed feedback is, the worse an RL controller's performance and learning efficiency are. Therefore, the significant issue is how to acquire a better RL controller in a delayed feedback environment.

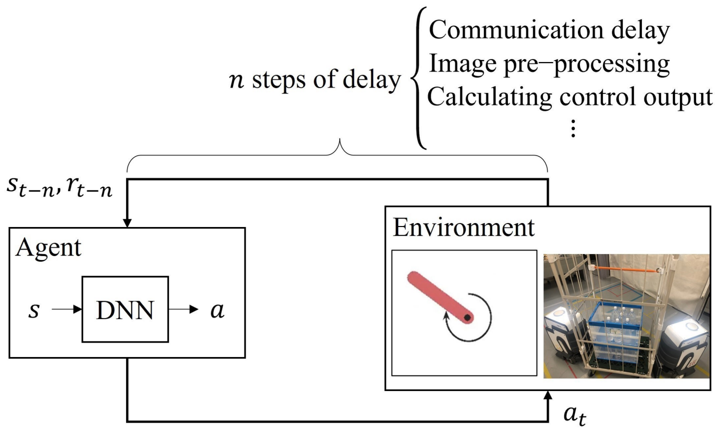


**Figure.3.1:** Problem setting of reinforcement learning in delayed feedback environment.

In this chapter, we propose an actor-critic reinforcement learning algorithm for achieving better performance and learning efficiency for RL controllers in delayed feedback environments. We assume that using undelayed feedback for training helps to acquire a better policy in delayed feedback environment. Also, in training phase, an agent can obtain undelayed feedback from simulator. Therefore, key idea of our algorithm is training a policy using undelayed feedback from simulator. First, we evaluate the proposed method in some continuous control task: pendulum swing up task [59], Hopper and HalfCheetah in Pybullet [18]. Then, we evaluate it with a logistics cart transportation system [26] as a real-world robot-control situation in a simulation environment. Finally, RL controllers learned in this simulation environment are transferred to real-world control without additional parameter tuning to evaluate the performance of the controllers.

## 3.2 RELATED WORK

### 3.2.1 Reinforcement learning in delayed feedback environment

Reinforcement learning methods for delayed feedback environments have been proposed. Many general reinforcement learning methods assume a Markov decision process (MDP); however, in a delayed feedback environment, if an agent uses only delayed observation, this setting assumes a partial observable Markov decision process (POMDP). Therefore, in [57], an extended observation was defined that includes delayed observation and an action history of the length of delay steps, and this setting was regarded as an MDP one.

Walsh et al. proposed a reinforcement learning method for constant delayed observation environments [60]. An agent has a state transition and reward predictor for model-based reinforcement learning.

Bouteiller et al. a proposed reinforcement learning method for random delayed observation and action environments [61]. An agent uses an $n$-step future sub-trajectory of observations and actions to reduce the estimation bias of a state-action value function and updates an action history in sub-trajectory sampling from a replay buffer by using the current policy. This processing helps learning by using new information whenever possible. However, updating an action history in a sub-trajectory is computationally expensive if the sub-trajectory length, which is defined in terms of delay steps, is long.

Our method utilizes undelayed feedback from simulator. This not only improves learning efficiency but also reduces the increase of learning cost.

### 3.2.2 Reinforcement learning using additional information for critic learning

Recently, efficient reinforcement learning methods using simulation information have been proposed. In the multi-agent actor-critic reinforcement learning domain, algorithms that use other agents' information as input to critics have been proposed [31, 62]. Also, Pinto et al. proposed a reinforcement learning algorithm for controlling robots from input images. This method uses the state from a simulator that is not obtained in the real world for training state-action value functions, and images for training policy. This improves the RL controller's performance [63].

As a similar idea, two-stage learning algorithms were proposed [64, 65]. First, a teacher policy is trained by using privileged information from a simulator. Then, a student policy is learned by imitating the teacher policy.

Our method is inspired by these research. We utilize undelayed feedback as input to critic in training phase for solving a reinforcement learning problem in delayed feedback environment.

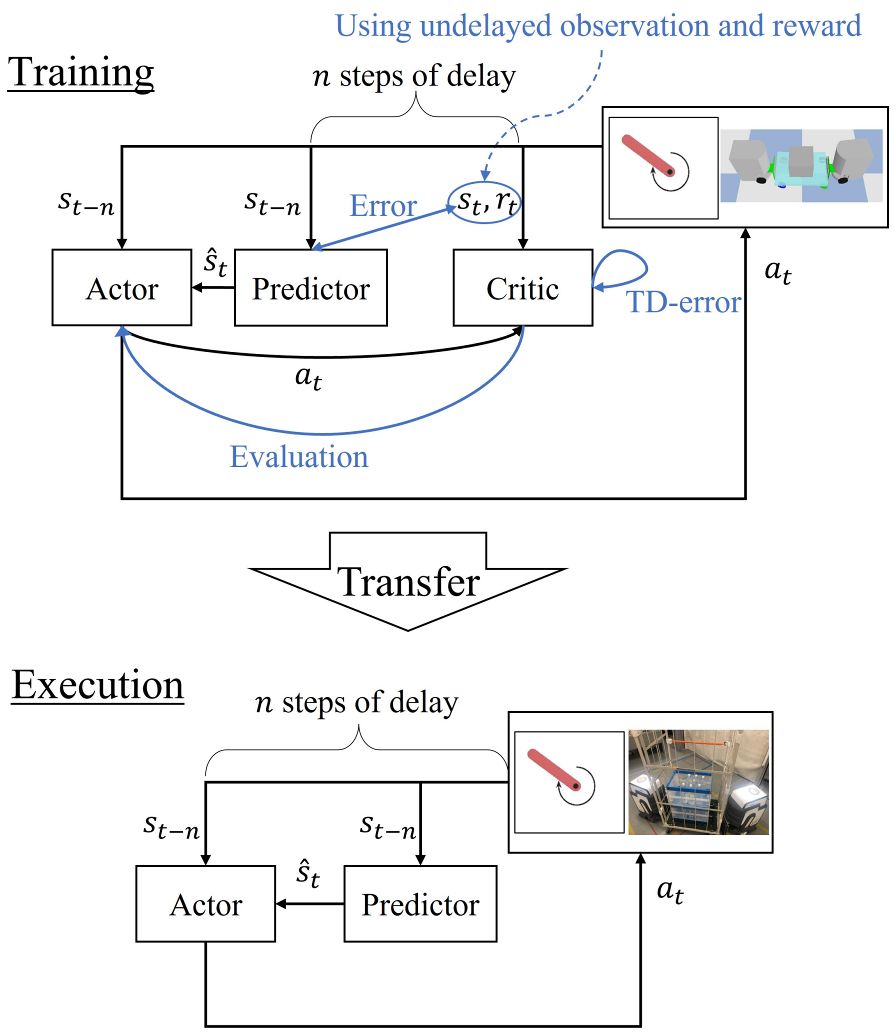


**Figure.3.2:** Overview of proposed method.

## 3.3 METHODOLOGY

In actor-critic reinforcement learning, it is probable that a Q-function trained by delayed feedback will lead to an unstable policy evaluation; therefore, the trained policy will probably be degraded. Thus, we propose a method for accurately estimating the Q-function using undelayed feedback as privileged information, which can lead to obtaining a better policy. We assume that the an agent know the amount of delay. Fig. 3.2 shows the overview of our proposed method.

### 3.3.1 Actor with delayed feedback and critic with undelayed feedback

The policy needs to calculate a control output from delayed observation $s_{t-n}$ in a simulation environment in order to transfer the policy to a real-world delayed feedback environment. Thus, we use extended observation $o_t = (s_{t-n}, a_{t-1}, \ldots, a_{t-n}) \in \mathbb{R}^{|S|+n|A|}$. $|S|$ is a observation space dimension, and $|A|$ is an action space dimension. This extended observation is based on [57]. For implementation of the extended observation, the action $a_{t-i}$ in the extended observation is a zero vector if $t - i < 0$. In this research, all models are approximated by a neural network and optimized by gradient descent. The policy maps extended observation $o_t$ to action $a_t$:

$$a_t = \pi_\phi (o_t), \tag{3.1}$$

where $\phi$ is a parameter of the policy.

The critic can utilize undelayed observations and rewards because it is only used in the training phase. First, the Q-function is defined as

$$Q_\theta(s_t, a_t) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi} \left[ \sum_{i=t}^{T} \gamma^i r_i | s_t, a_t \right]. \tag{3.2}$$

By approximating the expected value operation by means of an $N$ mini-batch, we define the loss function of the Q-function based on deep Q learning [1]

$$\mathcal{L}_\theta = (2N)^{-1} \sum (y - Q_\theta(s_t, a_t))^2, \tag{3.3}$$

where $\theta$ is a parameter of the Q-function, $r_t$ is a reward, and $\gamma$ is a discount factor. Also, $y = r_t + \gamma Q_{\bar{\theta}}(s_{t+1}, a_{t+1})$, and $\bar{\theta}$ is a parameter of the target network, which is updated as $\bar{\theta} \leftarrow (1 - \tau)\,\bar{\theta} + \tau\theta$ for tracking $\theta$ slowly, where $\tau$ is the hyper-parameter. The gradient of the loss function $\mathcal{L}_\theta$ is calculated:

$$\nabla_\theta \mathcal{L}_\theta = -N^{-1} \sum (y - Q_\theta(s_t, a_t))\, \nabla_\theta Q_\theta\,(s_t, a_t), \tag{3.4}$$

Also, the objective function of the actor is defined as:

$$J_\phi = \mathbb{E}\left[Q_\theta^\pi(s_t, \pi_\phi\,(o_t))\right] \tag{3.5}$$

When the Q-function evaluates a policy with (5), it can use the undelayed observation. Therefore, it can be expected to use an accurate gradient and obtain a better policy. Under implementation, the algorithm minimizes the loss function $\mathcal{L}_\phi = -J_\phi$. The gradient of the policy is calculated as:

$$\nabla_\phi \mathcal{L}_\phi = -N^{-1} \sum \nabla_a Q_\theta(s_t, a)|_{a=\pi_\phi(o_t)} \nabla_\phi \pi_\phi(o_t). \tag{3.6}$$

A gradient of a policy calculated via a Q-function is known as a deterministic policy gradient [53].

Our algorithm uses delayed observations and undelayed states and rewards. Thus, the tuple of the replay buffer becomes $(o_t, s_t, a_t, r_t, o_{t+1}, s_{t+1}, done_t)$. If the termination condition is satisfied, then $done_t = 1$, otherwise $done_t = 0$.

### 3.3.2 Predicting undelayed observation as input of actor

When the delay step is large, the extended observation space is large, and the policy is difficult to extract the information of the undelayed observation from the extended observation. Therefore, we consider an undelayed observation predictor for helping the policy learning. The predictor model's output is calculated in residual form:

$$\hat{s}_t = o_t[0] + f_\psi(o_t), \tag{3.7}$$

where $\psi$ is a parameter of the predictor, and $o_t[0] = s_{t-n}$.

Then, an action is defined as:

$$a_t = \pi_\phi(o_t, \hat{s}_t), \tag{3.8}$$

The policy's input includes the extended observation for mitigating the effect of the shift of the predicted undelayed observation because of learning the predictor simultaneously with reinforcement learning. The loss function of the critic is the same as (3). The gradient of the loss function of the actor is

$$\nabla_\phi \mathcal{L}_\phi = -N^{-1} \sum \nabla_a Q_\theta(s_t, a)|_{a=\pi_\phi(o_t,\hat{s}_t)} \nabla_\phi \pi_\phi(o_t, \hat{s}_t). \tag{3.9}$$

Next, we introduce the learning of the undelayed observation predictor. Using tuple $(o_t, s_t, a_t, r_t, o_{t+1}, s_{t+1}, done)$ sampling from the replay buffer, the estimated observation $\hat{s}_{t+1}$ is calculated from $o_{t+1}$ as follows.

$$\hat{s}_{t+1} = o_{t+1}[0] + f_\psi(o_{t+1}). \tag{3.10}$$

**Algorithm 5** Actor Undelayed Feedback Critic with Predictor (AUFCP)

```
Input: time discount factor $\gamma$, target network update rate $\tau$, standard deviation of action noise
    $\sigma$, standard deviation of target policy smoothing noise $\bar{\sigma}$, noise clipping coefficient $c$,
    policy update frequency $d$, batch size $N$:
Output: optimal policy $\pi^*$
 1: Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$ and policy network $\pi_\phi$
 2: Initialize target networks $\bar{\theta}_1, \bar{\theta}_2, \bar{\phi} \leftarrow \theta_1, \theta_2, \phi$
 3: Initialize predict network $f_\psi$
 4: Initialize replay buffer $\mathcal{B}$
 5: Initialize environment env
 6: $cum\_step = 0$
 7: for episode = 0, 1, . . . do
 8:     $s_0 \leftarrow$ env.reset()
 9:     $o_0 \leftarrow (s_0, 0, \ldots, 0)$
10:     for $t = 0, 1, \ldots$ do
11:         $\hat{s}_t \leftarrow o_t[0] + f_\psi(o_t)$
12:         $a_t \leftarrow \pi_\phi(o_t, \hat{s}_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$
13:         $s_{t-n+1}, s_{t+1}, r_t, done_t \leftarrow$ env.step($a_t$)
14:         $o_{t+1} \leftarrow (s_{t-n+1}, a_t, \ldots, a_{t-n+1})$
15:         $\mathcal{B}$.add($o_t, s_t, a_t, r_t, o_{t+1}, s_{t+1}, done_t$)
16:
17:         Sample mini-batch $(o, s, a, r, o', s', done)$ from $\mathcal{B}$
18:         $\hat{s'} \leftarrow o'[0] + f_\psi(o')$
19:         $\hat{s} \leftarrow o[0] + f_\psi(o)$
20:         Update predict network by (12)
21:         $a' \leftarrow \pi_{\bar{\phi}}(o', \hat{s'}) + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \bar{\sigma}), -c, c)$
22:         $y \leftarrow r + \gamma \min_{i=1,2} Q_{\bar{\theta}_i}(s', a')$
23:         Update critics by (4)
24:         if $cum\_step$ mod $d$ then
25:             Update policy by (6) or (9) using $Q_{\theta_1}$
26:             Update target networks:
                $\bar{\theta}_j \leftarrow \tau\bar{\theta}_j + (1 - \tau)\theta_j$
                $\bar{\phi} \leftarrow \tau\bar{\phi} + (1 - \tau)\phi$
27:         end if
28:
29:         $cum\_step \leftarrow cum\_step + 1$
30:         if done then break end if
31:     end for
32: end for
```

Then, the loss function of the undelayed observation predictor is calculated as follows.

$$\mathcal{L}_\psi = (2N)^{-1} \sum \{(s_t - \hat{s}_t)^2 + (s_{t+1} - \hat{s}_{t+1})^2\}/2. \tag{3.11}$$

Therefore, the gradient of the loss function $\mathcal{L}_\psi$ is

$$\nabla_\psi \mathcal{L}_\psi = (2N)^{-1} \sum (s_t - \hat{s}_t)\nabla_\psi f_\psi(o_t) + (s_{t+1} - \hat{s}_{t+1})\nabla_\psi f_\psi(o_{t+1}). \tag{3.12}$$

### 3.3.3 Learning algorithm

We use Twin Delayed DDPG (TD3) [7] as the learning algorithm. TD3 is one of the state of the art algorithm in the off-policy reinforcement learning. Our algorithm can be applied to other RL algorithms. The pseudo code of the proposed algorithm is shown in Algorithm 5.

**Table.3.1:** Implementation details

| Hyper-parameter | Pendulum swing up | Hopper HalfCheetah | Logistics cart transportation |
|---|---|---|---|
| Optimizer | | Adam [56] | |
| Learning rate | | $3 \times 10^{-4}$ | |
| Batch size | 64 | 256 | 256 |
| Replay buffer size | $10^5$ | $10^6$ | $2 \times 10^5$ |
| Target network update rate | | 0.005 | |
| Time discount factor | | 0.99 | |
| Action noise | | $\mathcal{N}(0, 0.1)$ | |
| Target policy smoothing noise | | $\mathcal{N}(0, 0.2)$ | |
| Noise clipping coefficient | 0.5 | 0.5 | 0.1 |
| Policy update frequency | | 2 | |
| Network architecture | | | |
| Actor | [input_dim, 256] [256,128] [128, action_dim] | [input_dim, 256] [256,256] [256, action_dim] | [input_dim, 256] [256,128] [128, action_dim] |
| Critic | [input_dim, 256] [256,128] [128, 1] | [input_dim, 256] [256,256] [256, 1] | [input_dim, 256] [256,128] [128, 1] |
| Predictor | [input_dim, 128] [128,64] [64, state_dim] | [input_dim, 128] [128,128] [128, state_dim] | [input_dim, 128] [128,64] [64, state_dim] |
| Activation function | | ReLU [66] | |

## 3.4 EXPERIMENTS

### 3.4.1 Methods for comparison

We compare three baseline methods with our method.

#### Simple reinforcement learning (Simple-RL)

The Q-function is learned with the delayed reward and extended observation $(s_{t-n}, a_{t-1}, \dots, a_{t-n})$, and the policy is evaluated by the Q-function and improved. Q-function and the policy is defined as $Q_\theta(o_t, a_t), \pi_\phi(o_t)$.

#### Reinforcement learning using prediction model (Predict-RL)

Predict-RL learns the predictors of the undelayed observation and reward simultaneously, and it uses the predicted undelayed observation and reward to learn the Q-function and the policy. This algorithm is similar to [60]. Q-function and the policy is defined as $Q_\theta(\hat{s}_t, a_t), \pi_\phi(\hat{s}_t)$. Also, this method has state predictor and reward predictor. Predicted state and reward are calculated as residual form:

$$\hat{s}_t = s_{t-d} + f_\psi(o_t, a_t), \tag{3.13}$$

$$\hat{r}_t = r_{t-d} + g_\xi(o_t, a_t). \tag{3.14}$$

These predictors $f_\psi, g_\xi$ are optimized by minimizing the mean squared error.

#### Reinforcement learning with undelayed feedback and prediction model (Predict/True-RL)

Predict/True-RL separates reinforcement learning and learning prediction model. Reinforcement learning module learns the policy using the undelayed feedback, and the predictor learns the prediction of the current observation from experiences gathering during learning the policy. The policy uses the predicted undelayed observation during the execution.

#### Proposed method (AUFCP-RL)

AUFCP-RL is the full model of our proposed algorithm.

We use four continuous control tasks for comparison: pendulum swing up, HopperBulletEnv-v0, HalfCheetahBulletEnv-v0 and logistics cart transportation. Details on the implementation of each control task are given in Table 3.1.

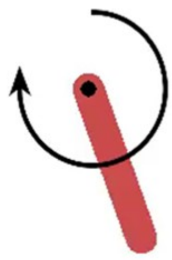


**Figure.3.3:** Pendulum swing up task [59]. Purpose is to keep pendulum standing upright.

### 3.4.2 Pendulum swing up

Fig. 3.3 shows the environment of the pendulum swing up. This task is implemented in OpenAIGym which is an open source Python library. The purpose is to keep the pendulum standing upright. Each controller issues a control output at 20 Hz. The time step of one episode is 200 , and the time of one episode is 10 s.

#### Observation

The state is three dimensions and defined as

$$s = \left(\cos\theta, \sin\theta, \dot{\theta}\right), \tag{3.15}$$

where $\theta$ is the angle of the pendulum. When the pendulum is kept standing upright, $\theta = 0$. $\dot{\theta}$ is limited to $[-8, 8]$rad/s.

#### Action

The action is torque force placed on the rotation axis of the pendulum, and its value is limited to $[-2, 2]$Nm.

#### Reward shaping

The reward function is defined as:

$$r = \cos\theta - 0.01\dot{\theta}^2 - 0.001a^2. \tag{3.16}$$

If the pendulum is standing upright, and the values of the angular velocity and control input are small, the reward approaches the highest reward of 1.

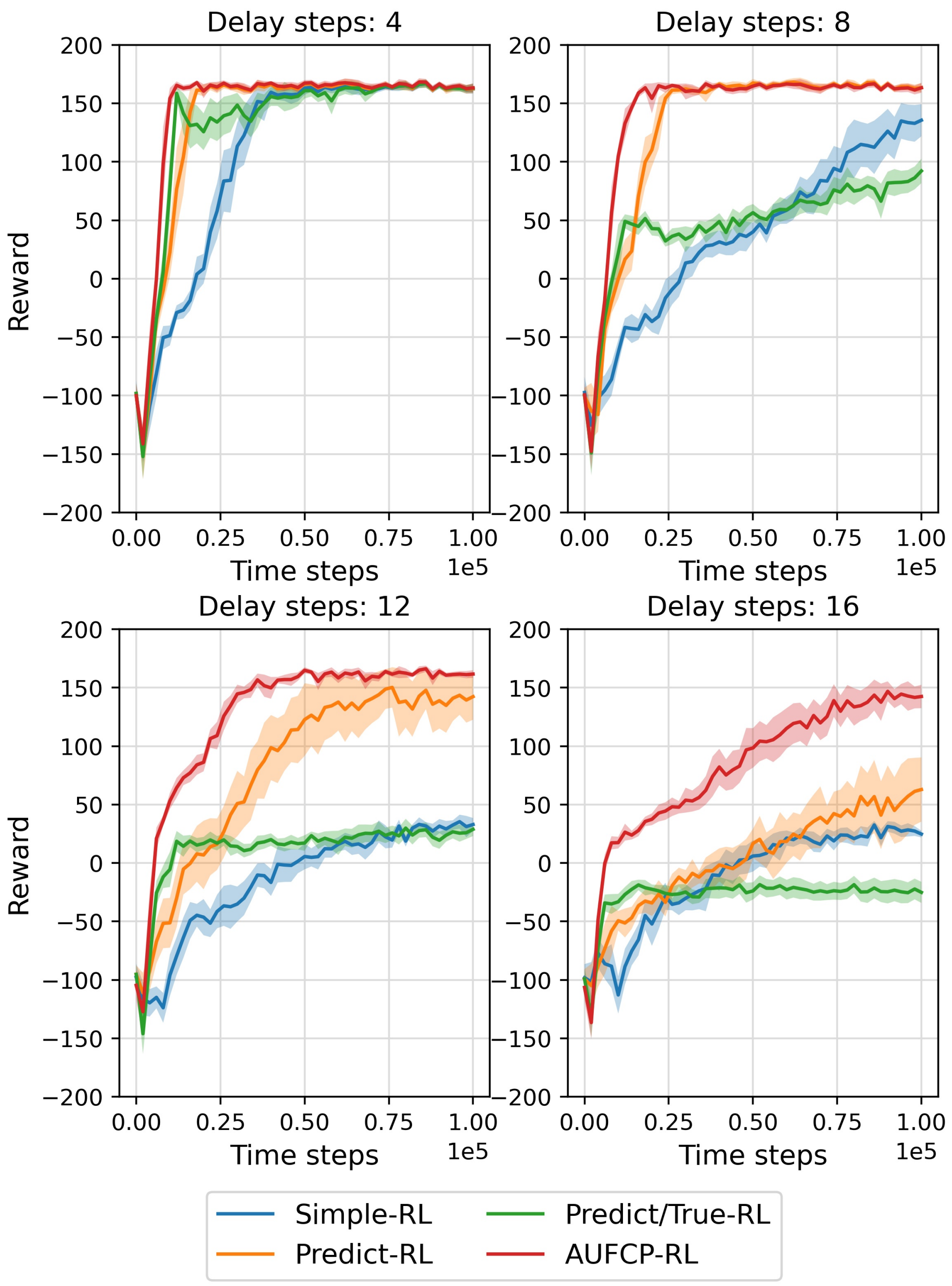


**Figure.3.4:** Learning curves of each method in pendulum swing up task under four delay step conditions [4, 8, 12, 16] steps. Lines mean average rewards, and shaded areas correspond to half standard deviation over 10 trials.

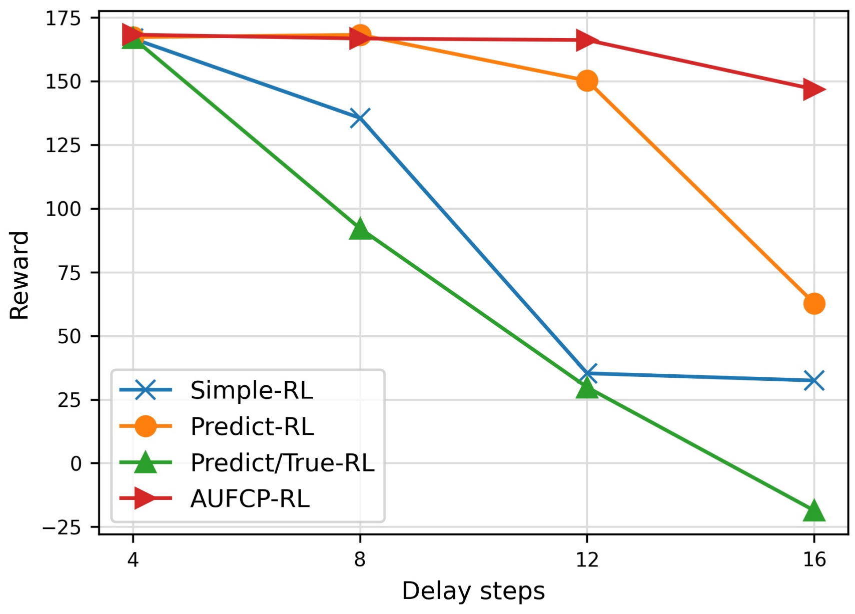


**Figure.3.5:** Max average rewards of each method under four delay step conditions [4, 8, 12, 16] steps in pendulum swing up task.

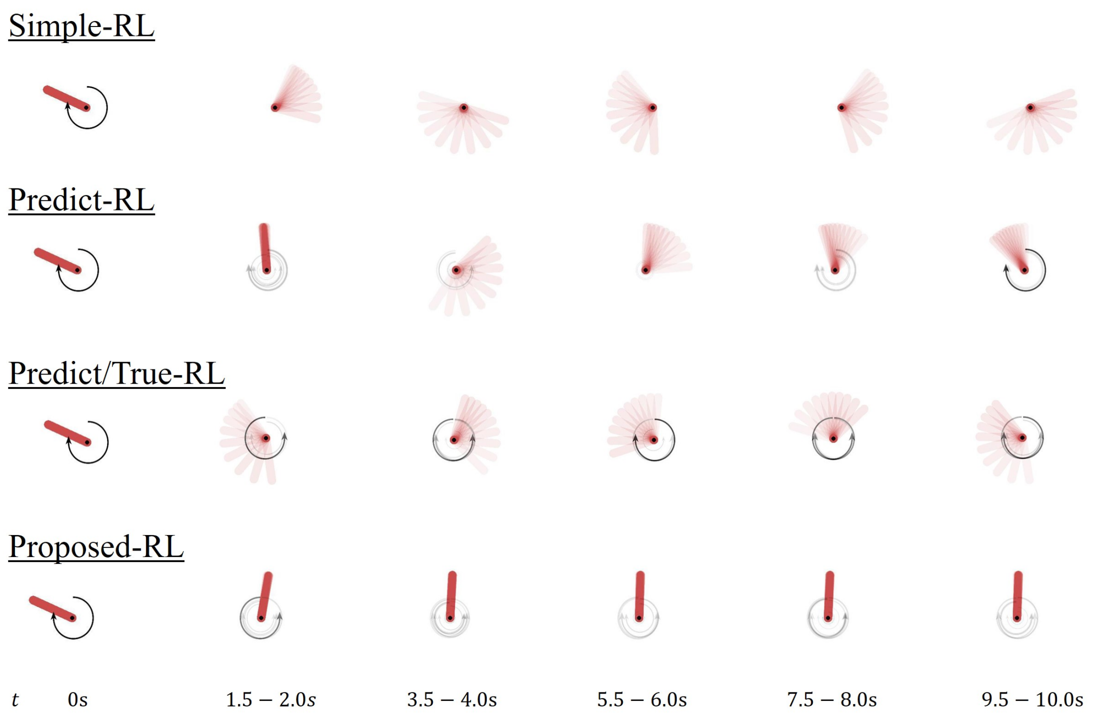


**Figure.3.6:** The Examples of the pendulum swing up task in 16 steps delayed feedback environment. The results show that our proposed method acquires a better policy than that of a conventional method in delayed feedback environment.

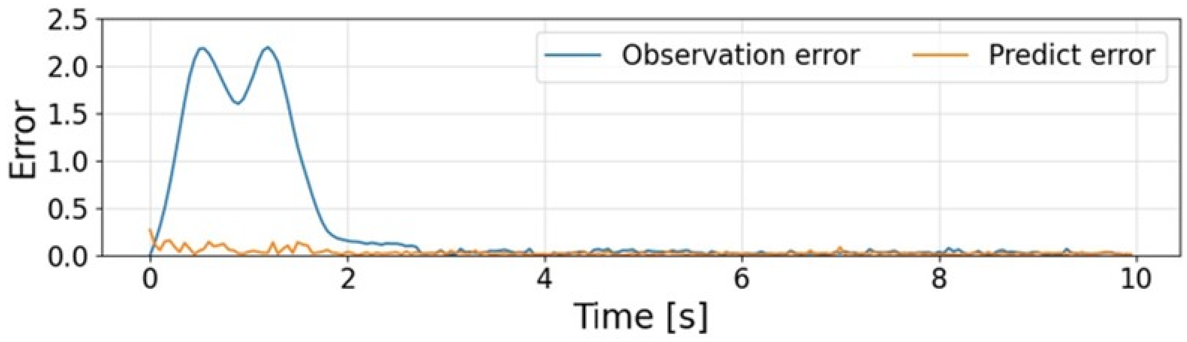


**Figure.3.7:** The example of the errors between true observation and delayed observation or predicted one of each experiments in pendulum swing up task with AUFCP-RL.

### Results

Each controller was trained for 10k time steps over 10 trials under 4 delay step conditions [4, 8, 12, 16] steps. The performance of each controller was evaluated on the basis of the average reward over 10 episodes. Fig. 3.4 shows the learning curves under four delay step conditions [4, 8, 12, 16] steps, and the max average rewards of each delay step and each method are plotted in Fig. 3.5. In all delay step conditions, AUFCP-RL had better learning efficiency and reached a better policy. Simple-RL degrades the learning efficiency as the delay step increases. Also, Predict-RL obtains the better policy than Simple-RL, however, the learning efficiency and the final performance of Predict-RL are worse than AUFCP-RL.

Fig. 3.6 shows the example of the pendulum swing up task in 16 steps delayed feedback environment with each method. Only AUFCP-RL succeeded to keep the pendulum swing up.

Fig. 3.7 shows the example of the error between true observation and delayed or predicted one in pendulum swing up task with AUFCP-RL. In early steps, the error between true observation and delayed one (Observation error) was large.

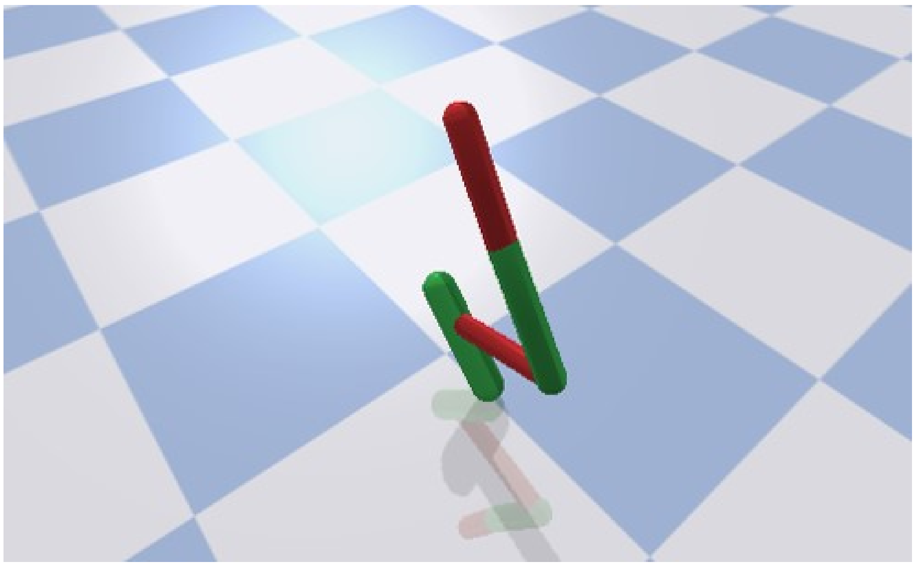


**Figure.3.8:** HopperBulletEnv-v0 [18]. This task environment is implemented in Pybullet.

## 3.4.3 Hopper

Fig. 3.8 shows the environment of Hopper. It is implemented in Pybullet and named "HopperBulletEnv-v0". The purpose of this agent is to walk to forward as fast as possible. Each controller issues a control output at 60 Hz. The max time step of one episode is 1000.

### Observation

The observation is 15 dimensions. The observation includes height, target direction (sin, cos), linear velocity, attitude (roll, pitch), joints angle (three dims.), joints angular velocity and foot contact (whether the foot contact with the floor or not).

### Action

The action is torques to be applied to each joint of the foot and three dimensions.

### Reward

The reward function is defined as

$$r = r_{alive} + v + \sum_{i}^{N} \frac{c_e|a_i\omega_i| + c_t a_i^2}{N} + c_{jl}N_{jl}. \tag{3.17}$$

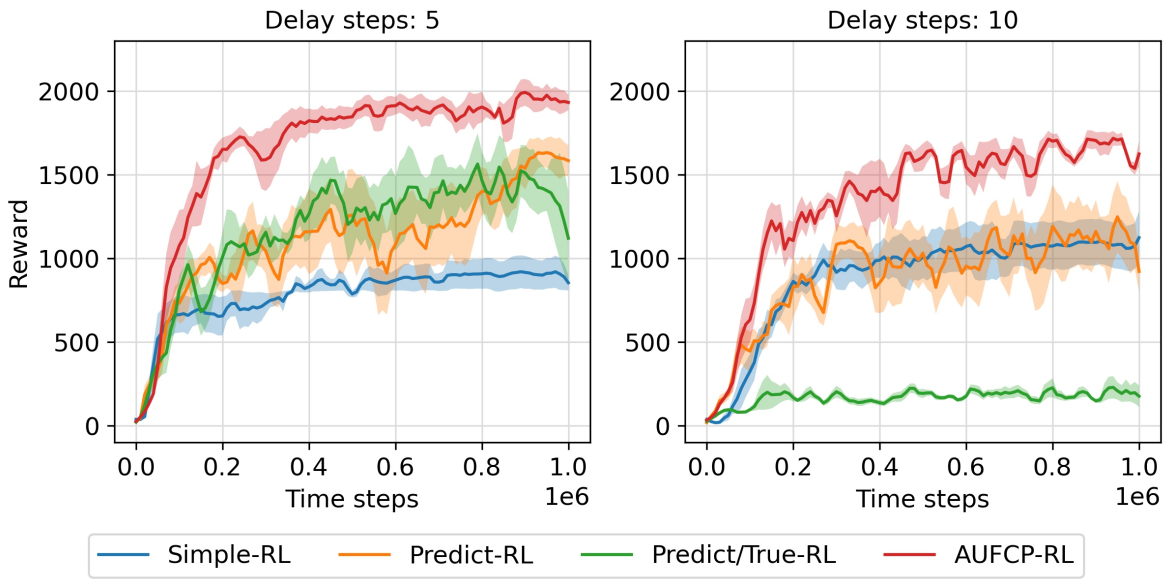


**Figure.3.9:** Learning curves of HopperBulletEnv-v0. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

$r_{alive}$ is the reward for maintaining the robot in a state where it can move forward, 1 if it can, -1 otherwise. If $r_{alive} < 0$, the episode is terminated. $v$ is the approach speed to the target. Also, $a$ is the action, $\omega$ is the angular velocity of the driving joint, $i$ is the joint's number, and the third term is defined as the penalty for power consumption. $N_{jl}$ is the number of actuating joints that are bent to the limit. For the constant terms, $c_e$ is $-2$ and $c_t$ and $c_{jl}$ are $-0.1$.

### Results

Each controller was trained for 1 million steps over five trials under two delay step conditions [5, 10]. Fig. 3.9 shows the learning curves. AUFCP-RL has better learning efficiency and higher reward than other methods. State prediction in Predict-RL is working effectively, therefore, the reward of Predict-RL is higher that of Simple-RL. When the delay step is 5, Simple-RL obtains the lowest reward. However, when the delay step is 10, Predict/True-RL is the worst performance.

## 3.4.4 HalfCheetah

Fig. 3.10 shows the environment of HalfCheetah. It is implemented in Pybullet and named "HalfCheetahBulletEnv-v0". The purpose of this agent is to walk to forward as possible. Each controller issues a control output at 60 Hz. The time step of one episode is 1000.

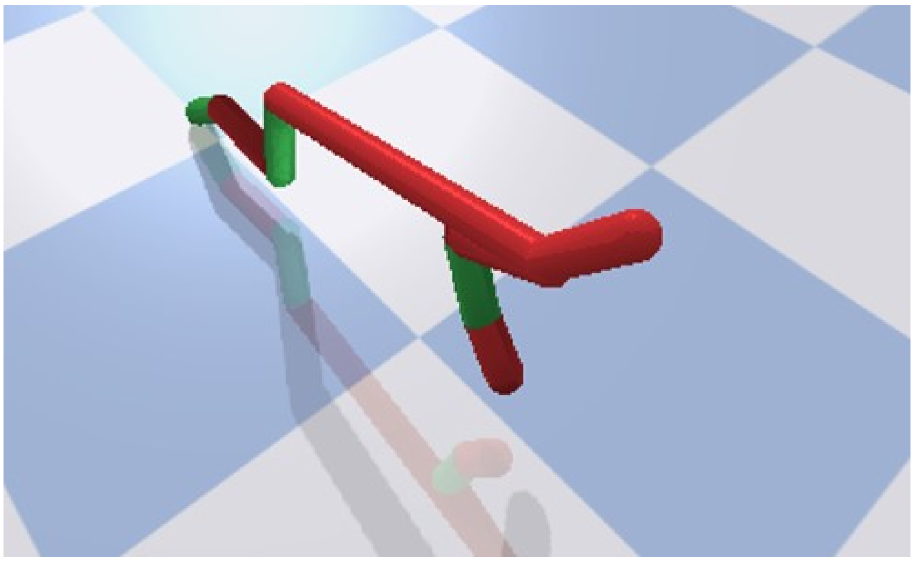


**Figure.3.10:** HalfCheetahBulletEnv-v0 [18]. This task environment is implemented in Pybullet.

### Observation

The observation is 26 dimensions. The observation includes height, target direction (sin, cos), linear velocity, attitude (roll, pitch), joints angle (six dims.), joints angular velocity and feet contact ("feet" mean front and back feet, shins and thighs).

### Action

The action is torques to be applied to each joint of the front and back feet and six dimensions.

### Reward

The reward is defined by the same formula as Hopper's. As opposed to Hopper environment, the episode does not terminate even if $r_{alive} < 0$. The number of HalfCheetah's feet and joints are more than those of Hopper's, therefore, HalfCheetah is more likely to be penalized.

### Results

Each controller was trained for 1 million steps over five trials under two delay step conditions [5, 10]. Fig. 3.11 shows the learning curves. AUFCP-RL has the best learning efficient and obtain the best performance. When the delay step is 5, the performance of Predict-RL is worse than that of Predict/True-RL. On the other hand, when the delay step is 10, the performance of Predict-RL is better than that of Predict/True-RL. Simple-RL has the lowest reward in all methods.

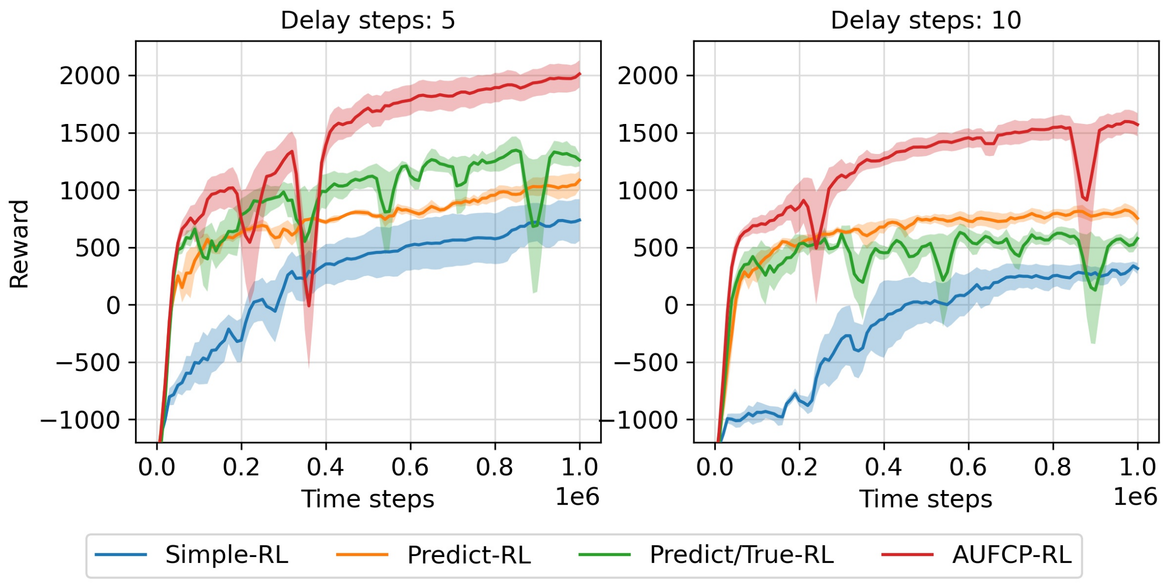


**Figure.3.11:** Learning curves of HalfCheetahBulletEnv-v0. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

### 3.4.5 Logistics cart transportation

This task is the same as logistics cart transportation in Chapter 2. A system overview is shown in Fig. 2.1.

#### Observation

Observation is the robots' states and estimated logistics cart state in the local coordinates. The detail is shown in Table. 2.1.

#### Action

The continuous action space is defined as each robot's linear velocity and rotational velocity ($|A| = 4$). The range of the linear velocity of each robot was $[-0.5, 0.5]$ m/s, and the range of the rotational velocity of each robot was $[-\pi/6, \pi/6]$ rad/s.

#### Reward shaping

The reward is calculated as:

$$r = \begin{cases} c_d d_{\mathrm{dv}} r_{\mathrm{rp}} \left(r_{\mathrm{pd}} + r_{\mathrm{od}}\right) - c_a \dot{a}^2 & (0 \leq d_{\mathrm{dv}}) \\ c_d d_{\mathrm{dv}} \left(4 - r_{\mathrm{pd}} - r_{\mathrm{od}}\right) - c_a \dot{a}^2 & (otherwise) \end{cases} . \quad (3.18)$$

As a difference the reward in Chapter 2, we add a penalty regarding the differentiation of RL output $\dot{a}$. The coefficient $c_d$ is set to 10 and the coefficient $c_a$ is set to $10^{-4}$.

If an episode satisfies the termination condition except when the time is up, the reward becomes $-10$.

### Residual reinforcement learning

To improve the learning efficiency, we used residual reinforcement learning [10, 27, 28]. In residual reinforcement learning, the control output is calculated as the sum of the base controller output and the reinforcement learning controller output:

$$a = \pi_h(o_\mathrm{h}) + \pi_\phi(o_\mathrm{rl}), \tag{3.19}$$

where $\pi_h$ is a given base controller (e.g., feedback controller), and $\pi_\phi$ is the residual element and optimized by reinforcement learning. $o_\mathrm{h}$ and $o_\mathrm{rl}$ mean the observation for the base controller and that of residual reinforcement learning controller, and do not have to be identical.

### Base controller for residual reinforcement learning

We used a feedback controller proposed in [38] for cooperative object transportation done using formation-based control as the base controller. In this method, an object is regarded as a virtual leader, and the robots are regarded as followers. Then, the robots are controlled by feedback control to maintain the relative positions and orientations on the basis of the object. See section 2.3.4 for more information on the base controller.

### Evaluation setting

In the simulation environment, each controller was trained for 50k time steps over five trials, and the number of delay steps was set to 4. The performance of each controller was measured on the basis of the average reward over 15 episodes.

Then, we transferred the top two controllers of each method trained in the simulation to a real-world experiment without additional parameter tuning. The real-world experiment is executed 15 trials and each trial is for 7 s. Therefore, each method was evaluated on the basis of the average normalized reward over 30 trials. The normalized reward was calculated by dividing the cumulated reward by time steps.

### Simulation Results

As shown in Fig. 3.12, the proposed methods obtained higher rewards and more learning efficiency than the conventional methods. Simple-RL, which uses an extended observation

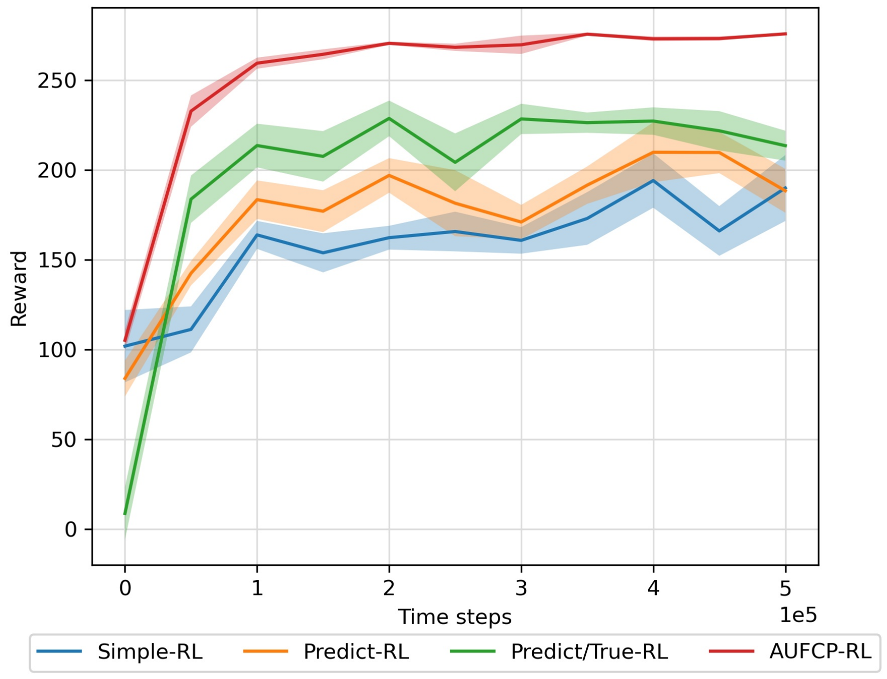


**Figure.3.12:** Learning curve of logistics cart transportation. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

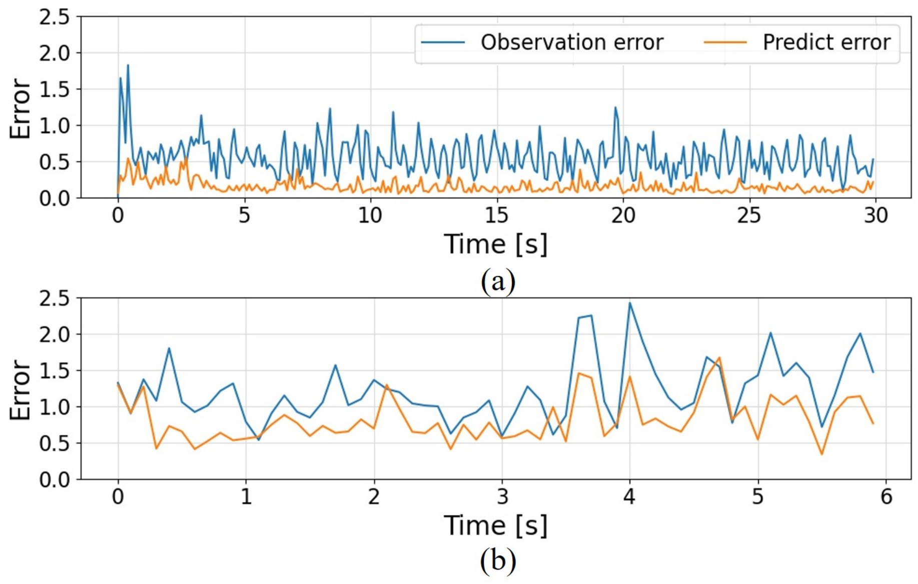


**Figure.3.13:** The examples of the errors between true observation and delayed observation or predicted observation (a) in simulation, and (b) in real world. The error means the norm between true observation and delayed observation or predicted observation.

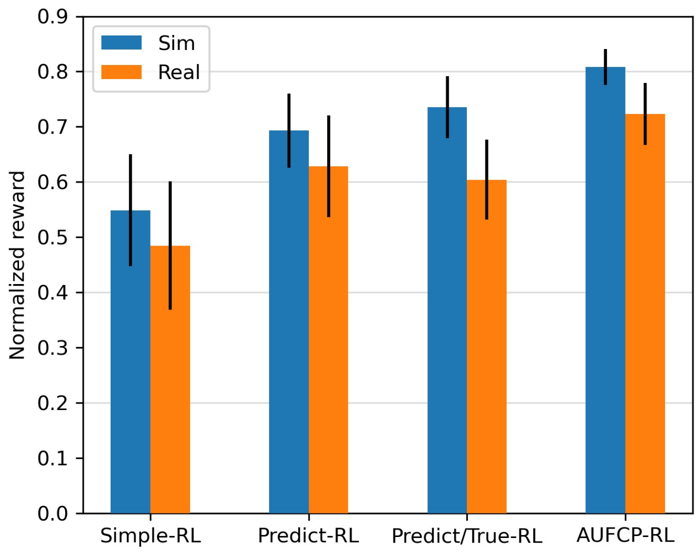


**Figure.3.14:** Results of real-world experiment. Normalized rewards of real-world experiment are average over 30 trials and visualized as orange bars in figure. Blue bars show average normalized reward in simulation under same conditions as real-world experiment. Black error bars show half standard deviation.

space, obtained a lower reward, the same as with the result for the pendulum swing up task. Also, the standard deviations of Predict-RL and Simple-RL were larger than that of AUFCP-RL.

Fig. 3.13 (a) shows the example of the error of delayed or predicted observation, which is the result of the execution by AUFCP-RL in simulation experiment. The error of delayed observation had some magnitude continuously. The error of predicted observation was smaller than that of delayed observation.

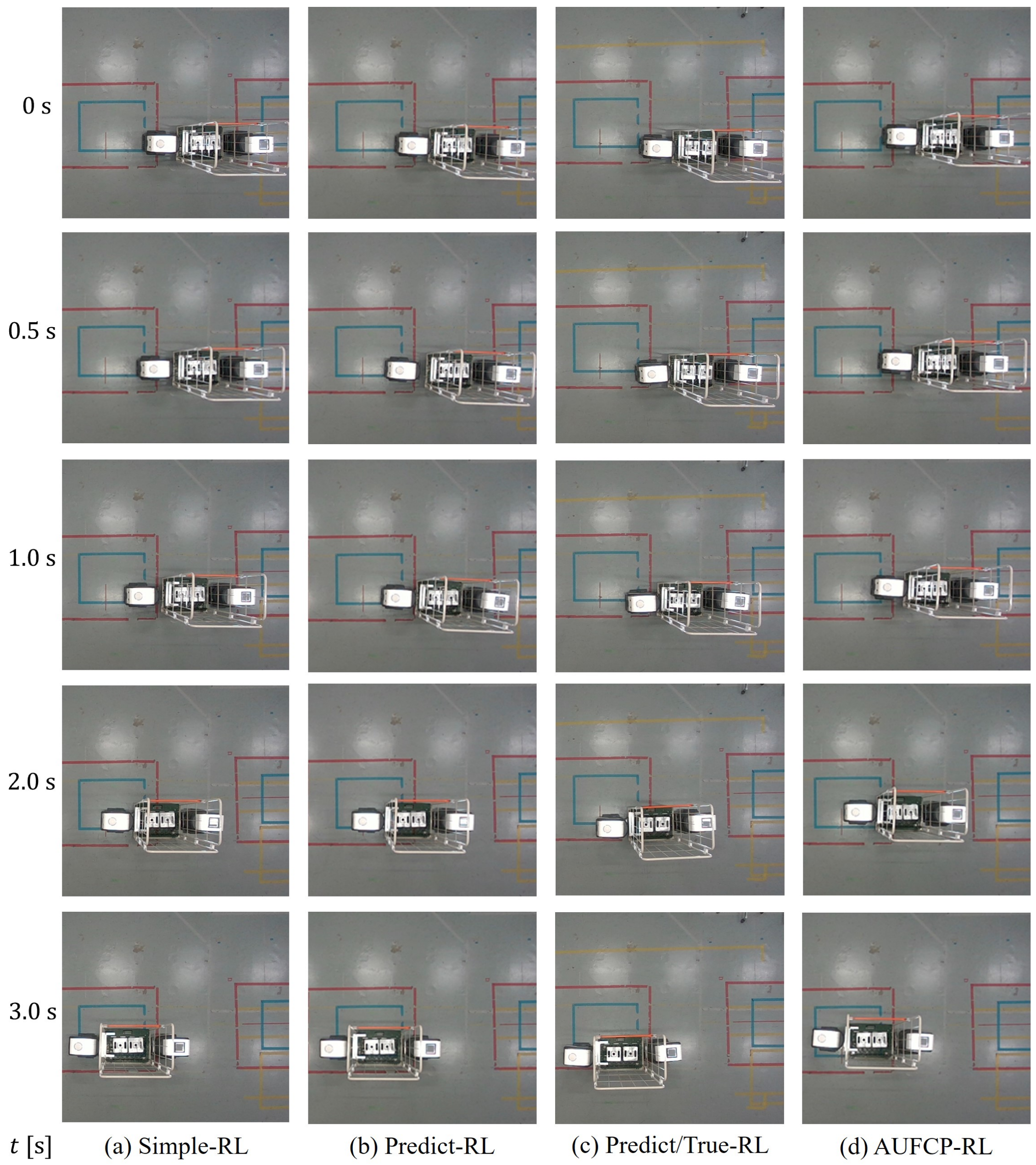


**Figure.3.15:** The example of the results of real-world experiments for 3 s. (a) Simple-RL, (b) Predict-RL, (c) Predict/True-RL, and (d) AUFCP-RL.

#### Real-world experiment

The controllers trained in the simulation were transferred to a real-world experiment and evaluated on the basis of the average normalized rewards. Fig. 3.14 shows the average normalized reward of each RL controller in the simulation and real world. The improvements of the proposed RL controllers in the simulation environment were reflected in the real-world experiment, and the proposed methods performed better than the conventional methods.

Fig. 3.13 (b) shows the example of the error of delayed or predicted observation, which is the result of the execution by AUFCP-RL in real-world experiment. The errors in real-world experiment were larger than the errors in simulation environment. The error of predicted observation was smaller than that of delayed observation.

Fig. 3.15 shows that the RL controllers of conventional methods did not keep holding the logistics cart. Simple-RL did not hold the logistics cart at 1.0 s, and Predict-RL did not hold the logistics cart at 0.5 s. Predict/True-RL made the robots move vibrationally. On the other hand, AUFCP-RL kept to hold the logistics cart.

### 3.4.6 Ablation study

Ablation study is conducted to confirm the proper functioning of the elements included in the proposed method. We evaluates each element in pendulum swing up with [8, 16] steps of delays, Hopper and HalfCheetah with [5, 10] steps of delays, and logistics cart transportation with 4 steps of delay in simulation environments. Specifically, we compare the full model with one without the predictor module (no-pred), one using only the predict state (no-concat) and one without the undelayed feedback critic (no-ufc).

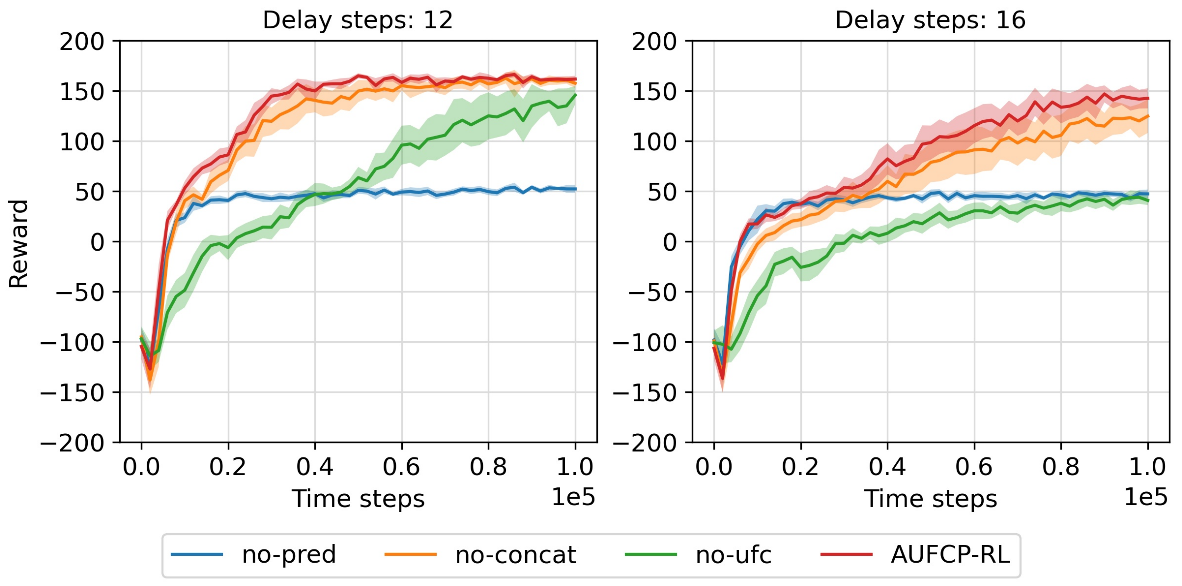


**Figure.3.16:** Ablation study in pendulum swing up at delay steps of 12 and 16. Lines mean average rewards, and shaded areas correspond to half standard deviation over 10 trials.

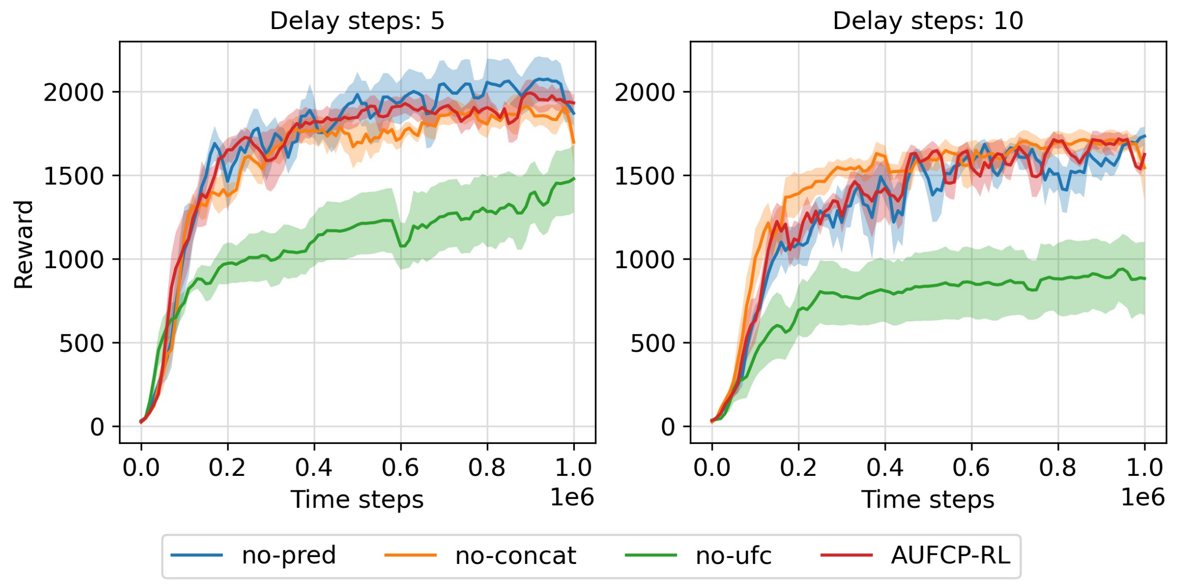


**Figure.3.17:** Ablation study in HopperBulletEnv-v0 at delay steps of 5 and 10. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

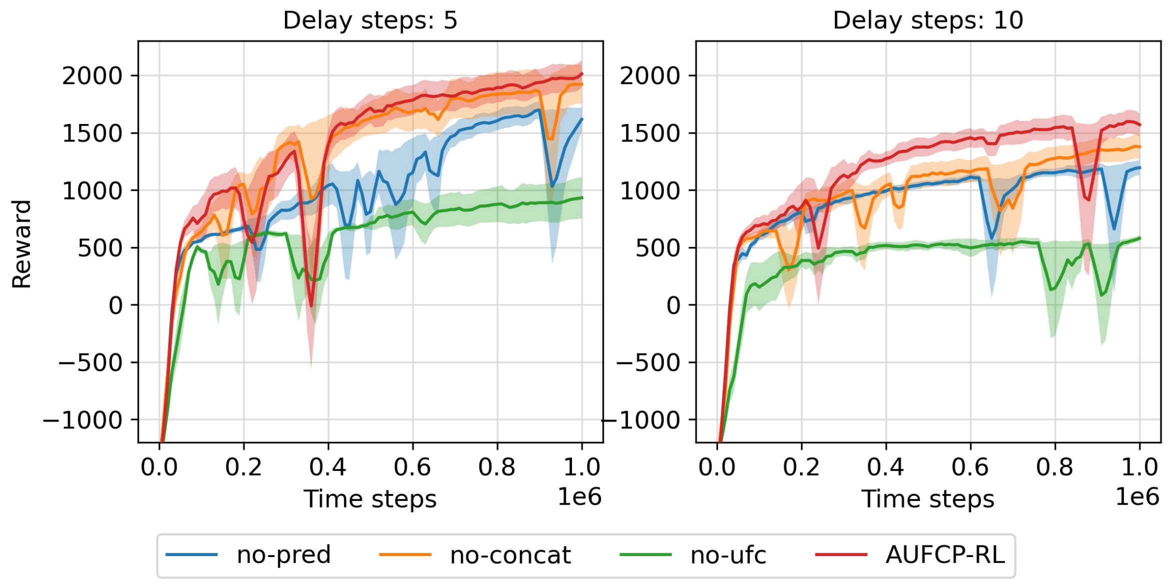


**Figure.3.18:** Ablation study in HalfCheetahBulletEnv-v0 at delay steps of 5 and 10. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

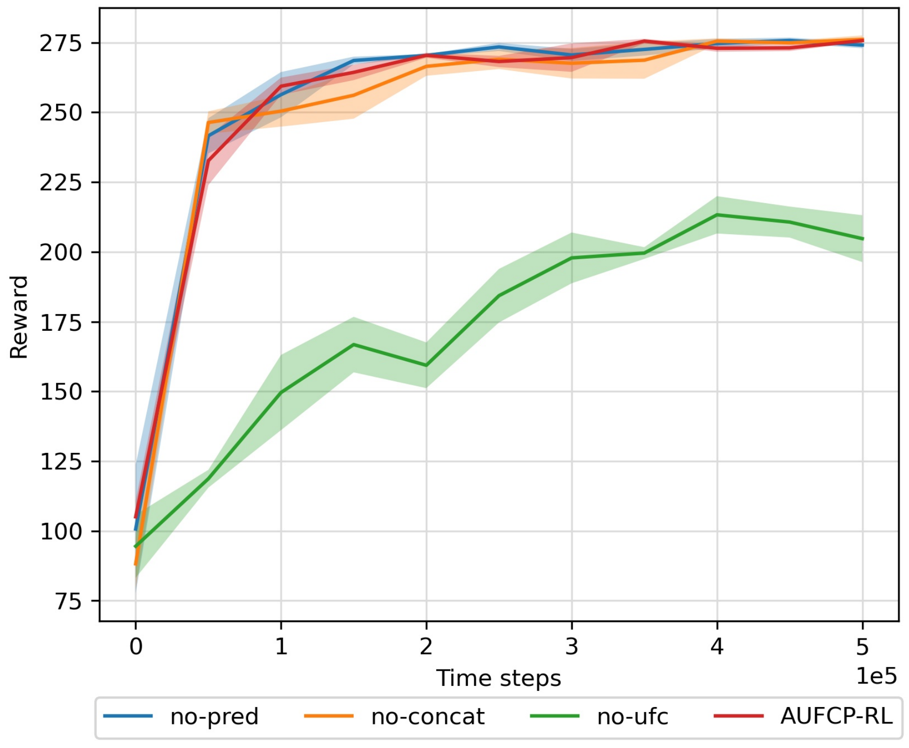


**Figure.3.19:** Ablation study in logistics cart transportation at delay step of 4. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

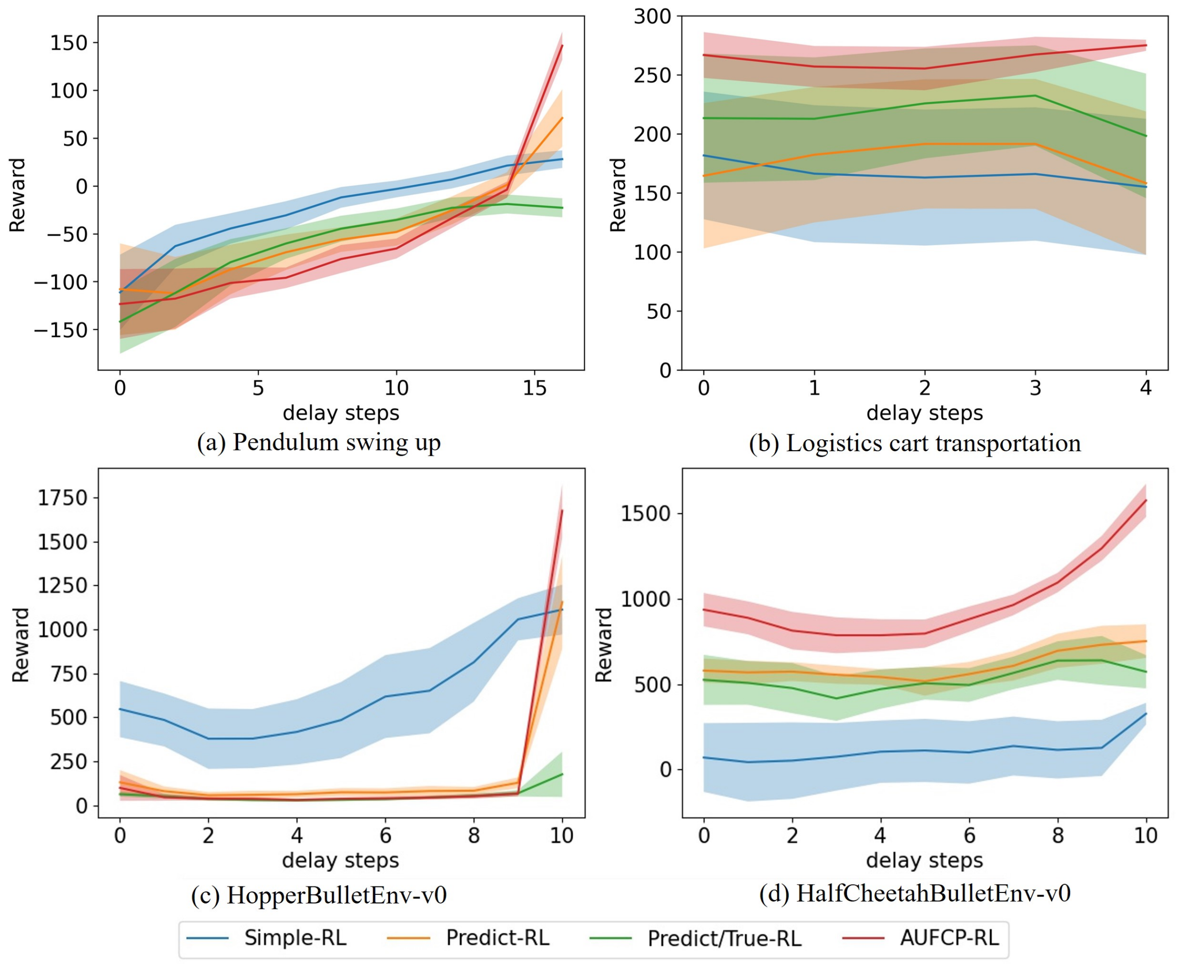


**Figure.3.20:** Robustness to the shift of the delay step from the learned delay step of each controller. Lines mean average rewards, and shaded areas correspond to half standard deviation.

## Results

Figs. 3.16 - 3.19 show that full model is superior in all tasks. The results of pendulum swing up and HalfCheetah show that the predictor and concatenating the predict state and the extended observation are effective. In the other tasks, Full model obtains the same rewards compared to the one without these elements. Especially, predictor model contributes to improve performance in pendulum swing up and HalfCheetah.

### 3.4.7 Robustness to the shift of the delay step

In this study, the delay is assumed to be constant and the amount of delay is known. On the other hand, in the real world, the amount of delay is not always known, and random delays often occur. Therefore, in this section, we test how the controllers learned by each method behave under delays that deviate from the learned delay step. Specifically, we use the controller trained with the delay of 16 steps for pendulum swing up, the controller trained with the delay of 4 steps for logistics cart transportation and the controller trained with the delay of 10 steps for Hopper and HalfCheetah. We execute each controller under some delay conditions that deviate from the learned delay step, and plot the means and standard deviations of the rewards.

#### Results

Fig. 3.20 shows the results of the experiments with some delay steps including the delay steps deviated from the learned delay step. In logistics cart transportation and HalfCheetah, each controller are relatively robust to the shift of the delay step. On the other hand, in other tasks, all controllers are not robust to the shift of the delay step. While Simple-RL does not perform well with learned delay step, it is relatively robust to the shift of delay step. In logistics cart transportation and HalfCheetah, AUFCP-RL outperforms the other methods in all the delay steps evaluated.

## 3.5 DISCUSSION

The four simulation experiments showed that evaluating the policy with the undelayed feedback critic improves the performance of the policy and learning efficiency.

### 3.5.1 Results of simulated continuous control tasks

In some simulated continuous tasks, using the predicted undelayed observations as input of the policy improved the performance of the policy.

Fig. 3.7 shows that the error between the true undelayed observation and the delayed one was large when the pendulum was swung to stand upright. This causes the difficulty to select a suitable action for controlling an agent and to evaluate the action for learning the policy based on the delayed observation because of the difference from the undelayed observation. Also, the dimension of the extended observation space increases in proportion to the number of delay steps and the action history becomes dominant in the extended observation space when the the number of delay steps is large. These makes extracting an essential undelayed observation information from the extended observation and mapping from the extended observation to the action difficult.

The reason why the performance of AUFCP-RL outperform is that using the undelayed feedback helps the accurate evaluation by the Q-function in these situations. Also, Predict-RL and AUFCP-RL can use the predicted observations which is near to the undelayed observation. Therefore, we consider that Predict-RL and AUFCP-RL improve the performances of the policies effectively when the error between the true observation and delayed one.

We consider that the deterioration of Predict/True-RL was caused by the difference between the true undelayed observation and the predicted undelayed observation. Thus, we consider that the performance deterioration of the Predict/True-RL controller was larger than that of other methods when the delay step is large.

Fig. 3.12 shows that the standard deviations of baseline methods were larger than that of AUFCP-RL. We believe that this result indicates that the Q-functions of these methods did not use the undelayed observation; thus, the Q-functions had large uncertainty, and this caused the policies update to be unstable.

## 3.5.2 Real-world experiments

The real-world experiment used the top two controllers of each method; therefore, Fig. 3.14 shows better situations. The large standard deviations of the baseline methods means that the probability of obtaining undesirable RL controllers with these methods was larger than with AUFCP-RL. The reason of the smaller reward of Predict/True-RL in the real-world experiment is that its policy was sensitive to the noise because it only used the true undelayed observation during training.

Fig. 3.13 (a) and (b) show that the results of the simulation experiment and the results of real-world experiment had similar features: (i) the error of predicted observation was smaller than that of delayed observation, and (ii) the error of delayed observation had some magnitude continuously. These suggest that the simulator approximated the real world to some extent. However, the errors in real world was larger than the errors in simulation, and the normalized reward in real world was degraded from that in simulation shown in Fig. 3.14. These also indicate that the simulator was still not sufficient to approximate the real world. The difference between the simulator and the real-world is important research topic called Sim2Real. In order to improve the performance of our method in real world, we need to work on Sim2Real.

### 3.5.3 Ablation study

We consider that the results of ablation study shows that elements included in the proposed model were effective because these rarely reduced the performance of the policy and often contributed to improve the performance of the policy.

The fact that the predictor module worked effectively suggests that it was often difficult to extract essential information from the extended observation and learned the appropriate policy. The reason for the effectiveness of concatenation is that it mitigated the change of input distribution to reinforcement learning because of learning the predictor module in parallel with reinforcement learning.

Investigating the results of ablation study in detail, the tasks in which the predictor module was effective are pendulum swing up and HalfCheetah. The difference between these tasks and the other two tasks is whether their tasks include transition from a low reward state to a high reward state or not. For example, in pendulum swing up task, the controller is required to swing up the pendulum from its lowered position, and in HalfCheetah, the controller is required to get up from a kneeling position. On the other hand, in Hopper, the episode is terminated if the agent cannot maintain the standing state, and in logistics cart transportation, the episode is terminated if the robots cannot keep holding the logistics cart. Recovery from a low reward state to a high reward state is likely to include dramatic change of a state, therefore, it is difficult to calculate an appropriate action from a delayed observation.

### 3.5.4 Robustness to the shift of the delay step

In pendulum swing up and Hopper, the reinforcement learning controller cannot deal with the shift of delay step, therefore, it is important to know the number of delay step in advance.

The robustness to the amount of the delay may be affected by the instability of high reward states. In other words, it might be more difficult for pendulum swing up and Hopper to maintain a rewarding state compared to the other two tasks. Due to the delay gap, we consider that the agents more prone to unstable states, and the inability to recover from such a state prevent them from obtaining the reward.

## 3.6 Summary

In this chapter, we proposed a method for efficient learning in a delayed feedback environment. The performance of the proposed method was evaluated in four simulation experiments. The proposed method realized efficient learning and obtaining the superior policy. Real-world experiments showed that the improvement of the policy by the proposed method in the simulation environment can be reflected in the real-world execution.

# Chapter 4

# Feature extraction for Domain Randomized Environment

## 4.1 Introduction

A lot of research on reinforcement learning for robot control using some physical simulation were proposed, including those in the previous chapters. In Chapters 2 and 3, we assumed that we could use the accurate physical simulation, but we do not always have access to such simulation. When we do not have the accurate simulator, we need a method to reduce the performance degradation of the policy transferred from a simulation in a real-world environment. Domain adaptation and domain randomization have been proposed as the two main methods to realize the sim-to-real transfer. In domain adaptation, the agent extracts a feature from a simulation and a real world and trains the extractor so that the features of a simulation and a real world are close. For learning the extractor, standard domain adaptation methods in a reinforcement learning domain require the data collected real-world execution. Therefore, the learning cost increases. In domain randomization, image textures and/or environment parameters are randomized. By randomizing these, the learned policy is made robust and is expected to work well in the real-world environment. One of the challenges of this method is that randomization makes the problems more difficult.

In this chapter, we propose a method for efficiently extracting important feature as state representation from observation and action histories in a domain randomized environment. This method is interpreted as performing domain adaptation in a domain randomized environment.

Also, privileged information obtained from a simulation uses for evaluating the policy by the Q-function accurately.

## 4.2 Related Works

### 4.2.1 Domain Adaptation

Domain Adaptation is an attempt to acquire a reinforcement learning controller learned in a simulation environment that work in a real-world environment. For realizing it, this idea learns to bring the representation of the real-world environment and that of the simulation environment closer together.

The attempt to obtain the controller that work in the real-world environment under the simulation environment by tuning environment parameters of the simulation based on data gathered in the real world can be considered as a kind of domain adaptation and is often called system identification.

Kaspar et al. [67] proposed a method to adjust the parameters of the simulation to match the real-world environment in a peg insertion task with an articulated arm robot. The parameters are identified by CMA-ES [68] so that the trajectories of the robots in the simulation and the real-world environment are close. Golemo et al. [69] uses a behavior policy to learn the difference between the state transitions of the real-world environment and the simulation environment. The simulation is corrected with the learned differences, and the policy is learned in the corrected simulation environment.

### 4.2.2 Domain Randomization

Domain randomization is the idea that by randomizing the parameters of the training environment, the generalization performance of the controller is improved, and thus the controller is expected to function when it is transferred to the real-world environment.

One of the domains to be randomized is an image. Tobin et al. [70] studied image-based object grasping and trained on simulated images that were transitioned to real images by randomizing the rendering in the simulation. The task is to estimate the position of an object from an image, and after estimating its position, a controller is used to grasp it.

Higgins et al. [71] proposed a method to extract essential information and obtain the robust policy in a domain randomized environment with $\beta$-VAE [72]. In this work, training $\beta$-VAE for estimating the latent variables and reinforcement learning are decoupled. Xing et al. [73] used Cycle-Consist VAE [74] to separate the latent variables from the image into domain-specific and domain-general ones, and only use the domain-general information. Similar to [71], learning Cycle-Consist VAE and reinforcement learning are decoupled. James et al. [75] realizes sim-to-real in an object grasping task by learning a generator that transforms an image into a canonical image. In this work, the learning process is decomposed into a generator that generates a canonical image from a simulated or real image and reinforcement learning. The canonical image generator uses the image-conditioned generative adversarial network (cGAN) [76] by U-Net [77]. Input to the reinforcement learning includes the canonical image, the randomized image in the simulation or the image in the real world, whether the gripper is open or not, and the height of the gripper.

These studiesexpect to acquire essential information as latent representations in the domain randomized environment. This can be interpreted as performing domain adaptation in the domain randomized environment.

There is also environment parameters as the domain to be randomized. In the work of Tan et al. [58], a reinforcement learning model transfer of a quadruped robot was attempted. This paper shown that accurate robot models, motor behaviors, and delay of the observation were reflected in the simulation is important for transferring the policy, and using a compact observation space and randomizing environment parameters are also effective. Peng et al. [78] and Andrychowicz et al. [79] proposed to use Long Short Term Memory (LSTM) [80] to implicitly respond to changes of environment parameters. However, these studies used an on-policy reinforcement learning algorithm to use the LSTM layer and runs a large parallel computation to ensure the diversity of experiences, which is computationally expensive. Yu et al. [81] learns the estimator for environment parameters from historical data and the estimated environment parameters are input to reinforcement learning to acquire the better policy.

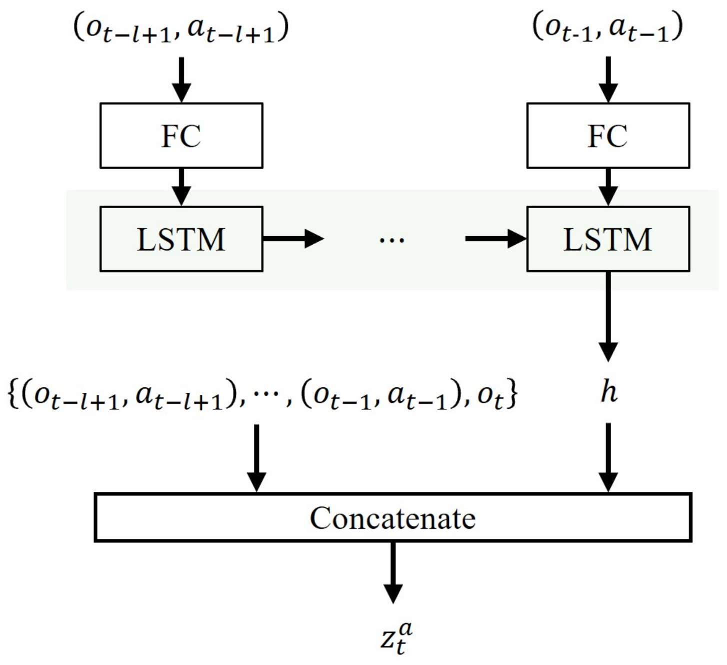
The less information we have about the real-world environment, the larger the domain range to be randomized in the simulation environment needs to be, which may make learning more difficult. Therefore, Chebotar et al. [82] proposed a method to adaptively change the randomization range in the simulation environment through a loop between the simulation and the real-world environment. In this study, the distributions of the environment parameters in the simulation are modified to minimize the misalignment of trajectories between the simulation and the real world, however, since the simulation is not differentiable, the sampling-based optimization method relative entropy policy search [83] is used. This is the method that combines domain randomization and system identification.

Another direction of research on domain randomization is to consider how to randomly sample from a domain [84]. The basic idea in [84] is to focus on sampling regions of the environment parameter space where it is difficult to obtain the reward. Specifically, two environments are run, a reference environment and a sampled environment, and a discriminator that identifies whether it is a reference or sampled environment is trained. The environment parameter sampler is designed so that the more the discriminator determines that the environment is sampled, the higher the reward. When the policy is well trained, it is expected that the reference environment and the sampled environment are indistinguishable. Therefore, environment parameter sampler is likely to sample environment parameters that are distinguishable, and the agent is expected to have not acquired the appropriate policy in that environment.

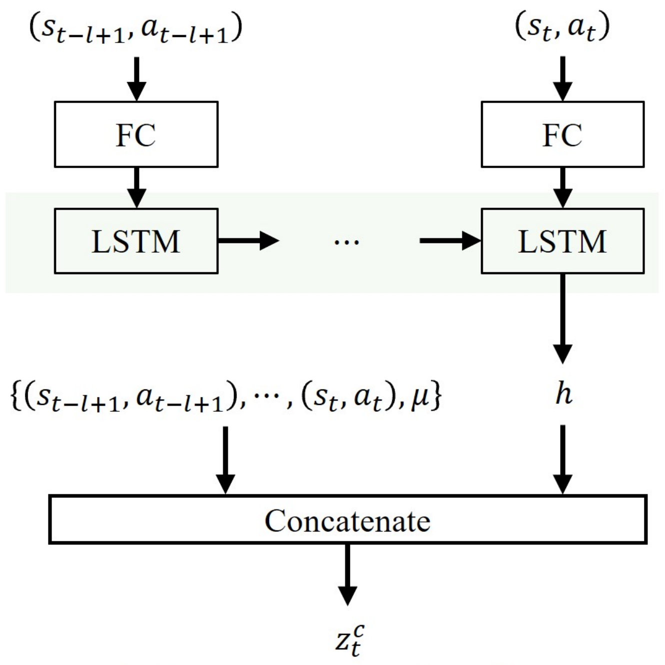
### 4.2.3 State representation learning in reinforcement learning

In [85], a feature extractor learns a low-dimensional representation that can predict the next state from a high-dimensional observation in advance, and connects it before the reinforcement learning network. This paper shown that this improves the performance of the reinforcement learning controller by efficiently extracting important feature from observations.

Also, Ota et al. [86] built on the work of Munk et al. to test whether learning high-dimensional representation from states improves the performance of the reinforcement learning controller. The experimental results showed that the high-dimensional feature is effective for reinforcement learning.



(a) Feature extractor in Actor



(b) Feature extractor in Critic

**Figure.4.1:** Feature extractors using LSTM.

## 4.3 Methodology

In this section, we describe the method for efficiently extracting features and obtaining the better controller in a domain randomized environment. We expect that this method leads to obtain the robust controller against the change of environment parameters and observation noise and stabilizes transferring the controller from the simulation to the real world.

### 4.3.1 Feature extraction to approach the MDP problem

Next, we introduce state representation learning to estimate the next state $s_{t+1}$ from the feature $z_t$ extracted by the network in Fig. 4.1. The feature for the actor is defined by concatenating the observation and action histories $h_t^o = (o_{t-l+1}, a_{t-l+1}, \dots, o_{t-1}, a_{t-1}, o_t)$ with the output of LSTM. Feature extractor for the actor is defined as

$$z_t^a = f_{\psi_a}(h_t^o). \tag{4.1}$$

where $\psi_a$ is the parameter of the feature extractor for the actor. The network architecture is shown in Fig. 4.1 (a)

As introduced in Chapter 3, we use true observations and environment parameters as privileged information. By using these variables as input to the feature extractor for the critic, we can expect that the evaluation of the policy can be more accurate, and thus the policy can be better. Specifically, the input to the feature extractor for the critic is $h_t^s = (s_{t-l+1}, a_{t-l+1}, \dots, s_{t-1}, a_{t-1}, s_t)$, $a_t$ and $\mu$ which is the values of environment parameters. Therefore, state representation for the critic is calculated as

$$z_t^c = f_{\psi_c}(h_t^s, a_t, \mu). \tag{4.2}$$

where $\psi_c$ is the parameter of the feature extractor for the critic. The network architecture is shown in Fig. 4.1 (b)

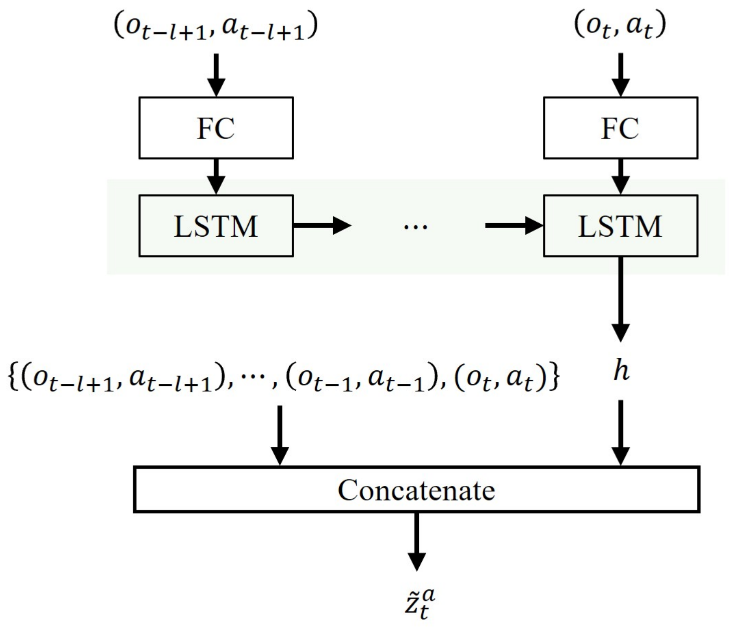
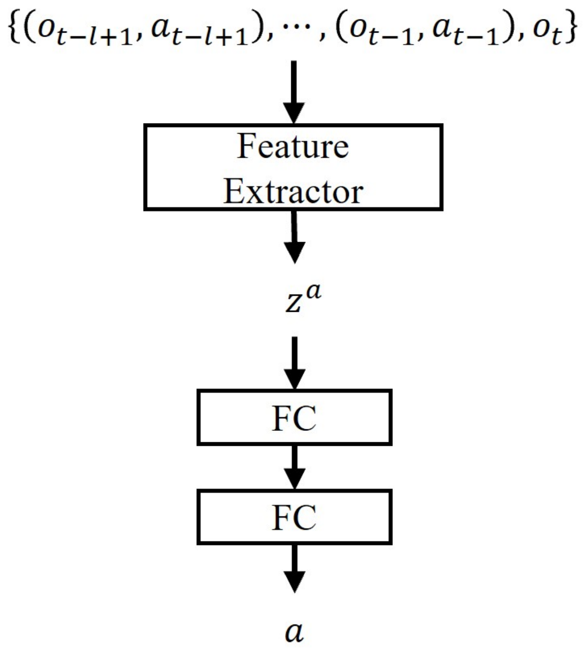


**Figure.4.2:** Feature extractor using LSTM for predicting the next state in the actor.



$\{(o_{t-l+1}, a_{t-l+1}), \cdots, (o_{t-1}, a_{t-1}), o_t\}$
Feature Extractor
$z^a$
FC
FC
$a$

**Figure.4.3:** Actor architecture with feature extraction. FC means fully connected layer.

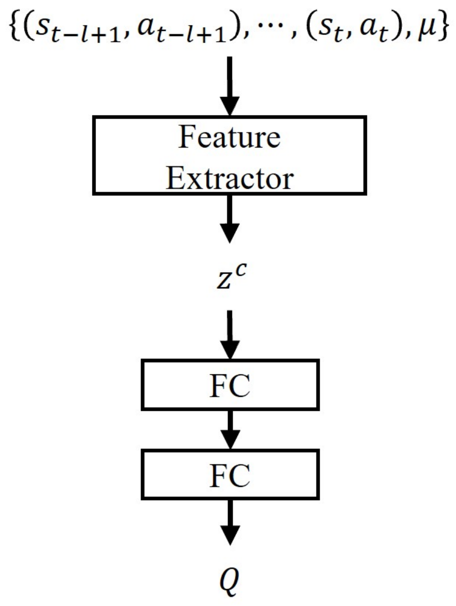


**Figure.4.4:** Critic architecture with feature extraction.

Predicting the next state from state representation for the actor is defined as

$$\hat{s}_{t+1} = g_{\xi_a}(\tilde{z}_t^a). \tag{4.3}$$

where $g_{\xi_a}$ is the linear transformation function, and $\tilde{z}_t^a$ is calculated by the architecture in Fig. 4.2. The network parameters for calculating $\tilde{z}_t^a$ is $\psi_a$. The difference from Fig. 4.1 (a) is that $a_t$ is added and one additional calculation is performed by LSTM using $(o_t, a_t)$ as input. Also, predicting the next state from state representation for the critic is defined as

$$\hat{s}_{t+1} = g_{\xi_c}(z_t^c). \tag{4.4}$$

where $g_{\xi_c}$ is also the linear transformation function.

The loss function for feature extractors are defined as

$$L_{fe} = \mathbb{E}\left[||s_{t+1} - \hat{s}_{t+1}||^2\right]. \tag{4.5}$$

This forces the learned state representation to be such that the transition to the next state can be estimated. In other words, the fact that the next state can be obtained only from the current state representation and behavior means that it is Markovian, which is consistent with the assumptions of reinforcement learning and is expected to facilitate the policy training.

**Algorithm 6** Reinforcement Learning with Learning Feature Extractor

**Input:** time discount factor $\gamma$, target network update rate $\tau$, standard deviation of action noise $\sigma$, standard deviation of target policy smoothing noise $\bar{\sigma}$, noise clipping coefficient $c$, policy update frequency $d$, batch size $N$, length of history $l$:

**Output:** optimal policy $\pi^*$

```
1:  Initialize critics $Q_{\theta_1}, Q_{\theta_2}$, policy $\pi_\phi$ and feature extractors $f_{\psi_a}, f_{\psi_c}$
2:  Initialize target networks $\bar{\theta}_1, \bar{\theta}_2, \bar{\phi} \leftarrow \theta_1, \theta_2, \phi$
3:  Initialize replay buffer $\mathcal{D}$
4:  Initialize environment env
5:  Initialize history queue (FIFO) $h_0^o$ and $h_0^s$ of length $l$ by zero padding
6:  $o_0, s_0, \mu \leftarrow$ env.reset()
7:  $h_0^o$.append(0, $o_0$), $h_0^s$.append(0, $s_0$)
8:  for $t = 0, 1, \ldots$ do
9:      $z_t \leftarrow f_\psi\left(h_t^o\right)$
10:     $a_t \leftarrow \pi_\phi(z_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma)$
11:     $o_{t+1}, s_{t+1}, r_t, done_t \leftarrow$ env.step($a_t$)
12:     $h_{t+1}^o \leftarrow h_t^o$.append(($a_t, o_{t+1}$)), $h_{t+1}^s \leftarrow h_t^s$.append(($a_t, s_{t+1}$))
13:     $\mathcal{D}$.add$\left(h_t^o, h_t^s, a_t, r_t, h_{t+1}^o, h_{t+1}^s, done_t, \mu\right)$
14:
15:     Sample mini-batch $(h^o, h^s, a, r, h'^o, h'^s, done, \mu)$ from $\mathcal{D}$
16:     $z^a \leftarrow f_{\psi_a}(h^o)$, $z^c \leftarrow f_{\psi_c}(h^s, a)$
17:     Update state representation networks by (4.5)
18:
19:     Resample mini-batch $(h^o, h^s, a, r, h'^o, h'^s, done, \mu)$ from $\mathcal{D}$
20:     $z^a \leftarrow f_{\psi_o}(h^o)$, $z^c \leftarrow f_{\psi_s}(h^s, a, \mu)$, $z'^a \leftarrow f_{\psi_o}(h'^o)$
21:     $a' \leftarrow \pi_{\bar{\phi}}(z'^a) + \epsilon, \quad \epsilon \sim$ clip$(\mathcal{N}(0, \bar{\sigma}), -c, c)$
22:     $z'^c \leftarrow f_{\psi_s}(h'^s, a', \mu)$
23:     $y \leftarrow r + \gamma \min_{i=1,2} Q_{\bar{\theta}_i}(z'^c)$
24:     Update critics by (4.6)
25:     if $t$ mod $d$ then
26:         Update policy by (4.7) using $Q_{\theta_1}$
27:         Update target networks:
            $\bar{\theta}_j \leftarrow \tau\bar{\theta}_j + (1-\tau)\theta_j$
            $\bar{\phi} \leftarrow \tau\bar{\phi} + (1-\tau)\phi$
28:     end if
29:
30:     if $done_t$ then
31:         Initialize history queue (FIFO) $h_{t+1}^o$ and $h_{t+1}^s$ by zero padding
32:         $o_{t+1}, s_{t+1}, \mu \leftarrow$ env.reset()
33:         $h_{t+1}^o$.append((0, $o_{t+1}$)), $h_{t+1}^s$.append((0, $s_{t+1}$))
34:     end if
35: end for
```

**Table.4.1:** Network architecture

| | |
|---|---|
| Actor | [input_dim, 256]<br>[256,256]<br>[256, action_dim] |
| Critic | [input_dim, 256]<br>[256, 256]<br>[256, 1] |
| Feature extractor | FC: [input_dim, 128]<br>LSTM: [128, 128] |
| Activation function | ReLU [66] |

The structure of the actor and the critic are Fig. 4.3 and Fig. 4.4. The loss functions for training the reinforcement learning controller are defined as

$$L_{actor} = -\mathbb{E}\left[Q_\theta\left(f_{\psi_a}\left(h_t^o, \pi_\phi\left(z_t^a\right)\right)\right)\right], \tag{4.6}$$

$$L_{critic} = \mathbb{E}\left[\|y - Q_\theta(z_t^c)\|^2\right]. \tag{4.7}$$

where $y = r_t + \gamma Q_{\bar{\theta}}(z_t^c)$. State representation learning and reinforcement learning are performed independently. Therefore, the gradient information of the actor and critic losses do not affect the feature extractor module. The pseudo code of the proposed method is shown in Algorithm. 6.

## 4.4 Experiment

### 4.4.1 Methods for comparison

#### Simple-RL

It is the simple reinforcement learning controller that learns the policy from a one step of the observation.

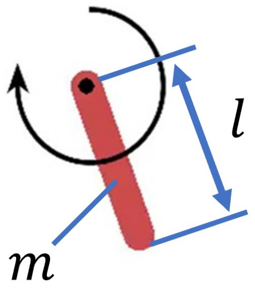


**Figure.4.5:** The randomized parameters in pendulum swing up task. $l$ is the length of the pendulum, and $m$ is the weight of the pendulum.

### RL with privileged critic and history of 4 steps(PCHIST-RL)

In this method, the true observation and environment parameters are used to learn the critic. Also, the controller is learned from the historical observation and action of 4 steps for acquiring environment information implicitly.

### RL with privileged critic and LSTM (PCLSTM-RL)

The true observation and environment parameters are used to learn the critic. We also expect to acquire environment information implicitly through using LSTM. Similar to the idea of Peng et al. [78], but they use the on-policy reinforcement learning method and learn from the history of the entire episode, while we learn from the historical observation and action of 4 steps to realize off-policy reinforcement learning. We adopt the network architecture of [87], which is a study of applying LSTM to the historical data as a way to deal with the noise or missing observations.

### Proposed model (Proposed-RL)

This is the proposed model described in section 4.3. The implementation details is shown in Table. 4.1.

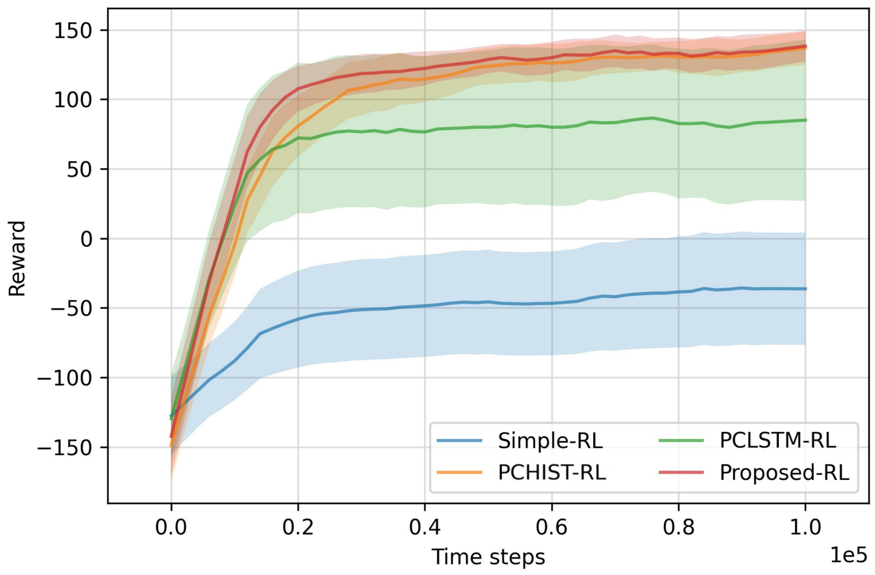


**Figure.4.6:** Learning curves of pendulum swing up in domain randomized environment. Lines mean average rewards, and shaded areas correspond to half standard deviation over 10 trials.

### 4.4.2 Pendulum swing up

#### Problem setting

We randomize some environment parameters of the pendulum swing up environment in Section 3.4.2. Specifically, we randomize the weight and length of the pendulum, and add Gaussian noise to the observations for constructing a domain randomized environment. The length of the pendulum is in the range [0.5, 1.5] (default: 1.0), and the weight of the pendulum is in the range [0.8, 1.8] (default: 1.0). The randomized parameters are sampled based on the uniform distribution. The reward, action, and observation are the same as in section 3.4.2.

#### Results

Fig. 4.6 shows that the rewards obtained by PCHIST-RL and Proposed-RL were almost the same and higher than the others. Also, sample efficiency of Proposed-RL was better than that of PCHIST-RL. PCLSTM-RL had large standard deviation. This result indicates that some PCLSTM-RL controllers fell into local optimum. Simple-RL had the lowest learning efficiency and reward.

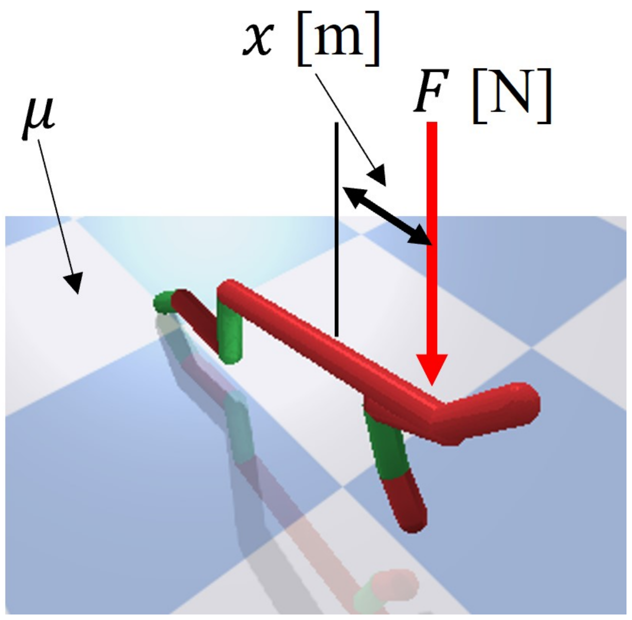


**Figure.4.7:** The randomized parameters in pendulum swing up task. $\mu$ is the floor friction coefficient, $F$ is the amount of the force applied to the body, and $x$ is the position to apply the force.

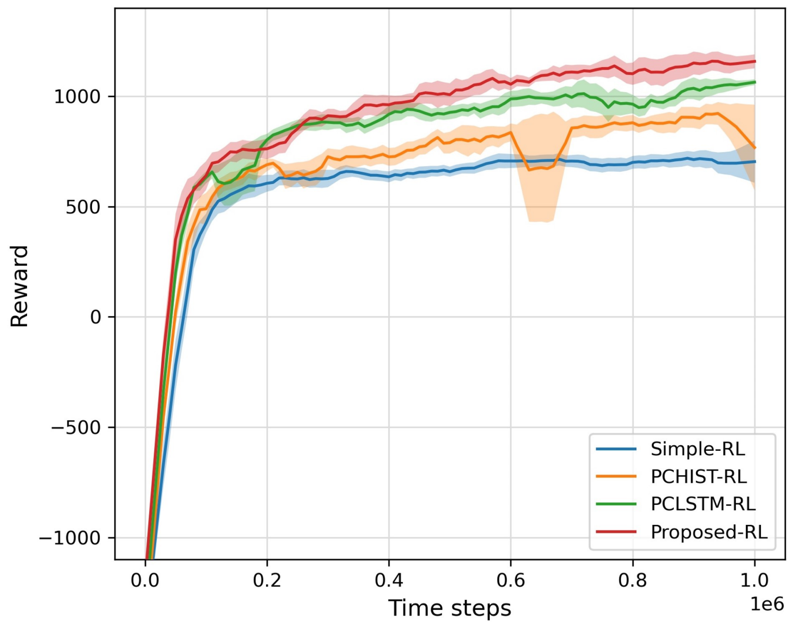


**Figure.4.8:** Learning curves of HalfCheetah in domain randomized environment. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

**Table.4.2:** Environment parameters of logistics cart transportation.

| Randomized parameters | Range |
|---|---|
| maximum torque of the motor | $[0.5, 1.5]$ |
| friction coefficient of the holding surface | $[0.17, 1.0]$ |
| damping coefficient of the rotating part of the holding mechanism | $[0.6, 1.0]$ |
| resistance force of the rotation of the wheel of the logistics cart | $[0.8, 1.2]$ |
| resistance force of the wheel axle of the logistics cart | $[0.8, 1.2]$ |
| weight of the load | $[0.7, 1.3]$ |
| floor friction coefficient | $[0.8, 1.2]$ |

### 4.4.3 HalfCheetah

#### Problem setting

We randomize some environment parameters of the HalfCheetahBulletEnv-v0. Specifically, we randomize the floor friction coefficient and the external force to the randomized position of the body and Gaussian noise to the observations for constructing a domain randomized environment. The floor friction coefficient is in the range $[0, 3.0]$, the force is in the range $[0, 50.0]$ N and the position to apply the force is in the range $[-0.5, 0.5]$ m. The randomized parameters are sampled based on the uniform distribution. The reward, action, and observation are the same as in section 3.4.4.

#### Results

Fig. 4.8 shows that Proposed-RL obtained the highest reward in all methods. Also, the reward of PCLSTM-RL was better than that of PCHIST-RL, unlike the experimental results of pendulum swing up. On the other hand, Simple-RL acquired the lowest reward as well as the experimental results of pendulum swing up.

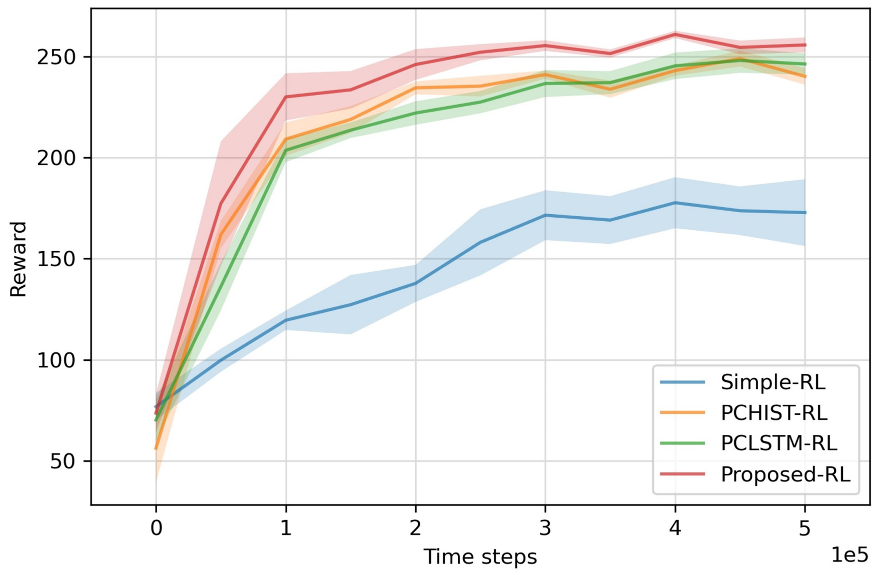


**Figure.4.9:** Learning curves of logistics cart transportation in domain randomized environment. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

### 4.4.4 Cooperative Logistics cart transportation

#### Problem setting

Based on the logistics cart transportation system in section 3.4.5, the maximum torque of the motor, the friction coefficient of the holding surface, the damping coefficient of the rotating part of the holding mechanism, the resistance force of the rotation of the wheels of the logistics cart and the resistance force of the wheel axle, the weight of the load, and the floor friction coefficient are randomized, and observation is added Gaussian noise. The randomized parameters are shown in Table 4.2. This table shows the randomization range of the environment parameters. A value of 1.0 means that the value of its environment parameter is equal to that in Chapter 3. In other words, the values of randomized ranges in Table 4.2 mean the ratios from the default values.

In logistics cart transportation system, delayed feedback is assumed to occur as in Chapter 3. Therefore, we use the extended observation $(o_t, a_{t-d}, \ldots, a_{t-1})$ as the observation at time $t$. Here, $o_t$ is the state with $d$ steps delayed and added gaussian noise. The state used as input to

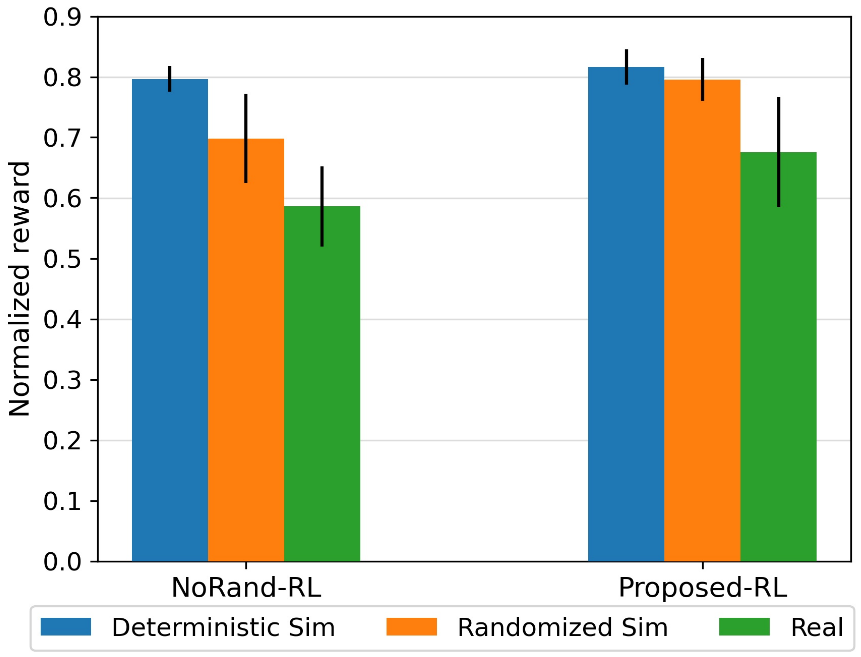


**Figure.4.10:** Results of real-world experiment. Black error bars show half standard deviation.

the critic is $(s_t, a_{t-d}, \ldots, a_{t-1})$, where $s_t$ is the state at time $t$ and not added any observation noise.

We also evaluate sim-to-real transfer performance of the proposed controller trained in domain randomized environment. Specifically, we compare the proposed method with the controller that is trained in the deterministic environment and calculates the action from the one step observation because of Markov property (NoRand-RL). We transferred the top one controller of the proposed method trained in domain randomized environment to a real-world experiment without additional parameter tuning. The real-world experiment is executed 10 trials and each trial is for 7 s. The normalized reward is calculated by dividing the cumulated reward by time steps.

### Results of simulation experiments

Fig. 4.9 shows that Simple-RL did not obtain the better policy as appropriate as that in other tasks. PCLSTM-RL obtained higher rewards than PCHIST-RL slightly. Proposed-RL had the highest reward and the most sample efficiency in all methods.

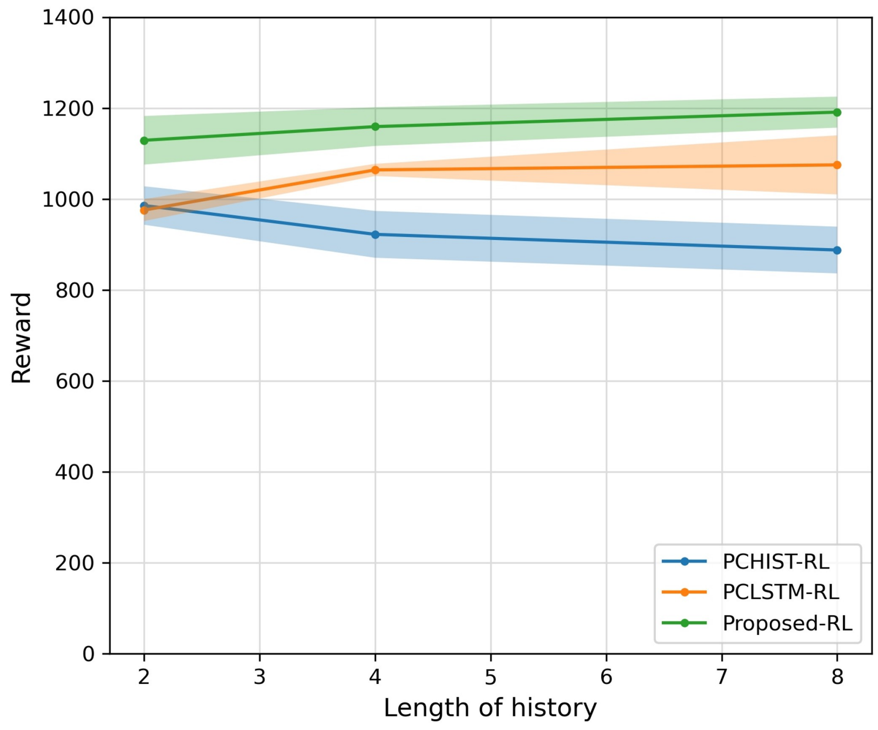


**Figure.4.11:** Robustness to the history length. Lines mean average rewards, and shaded areas correspond to half standard deviation over five trials.

#### Results of real-world experiments

Fig. 4.10 shows the results of the real-world experiments. Each controller has three experimental results. Blue bars are normalized rewards of deterministic simulation experiments which are average over 50 trials. Orange bars are these of domain randomized simulation experiments which are average over 50 trials. Also, green bars shows normalized rewards of real-world experiments which are average over 10 trials. The normalized rewards in the deterministic simulation were almost the same. However, in the domain randomized and real-world environments, Proposed-RL outperformed NoRand-RL.

### 4.4.5 Robustness to the length of the history

#### Problem setting

We evaluate PCHIST-RL, PCLSTM-RL, and Proposed-RL under three history's lengths [2, 4, 8] in the HalfCheetah task. The detail of the experimental environment is the same as in section 4.4.3.

#### Results

Fig. 4.11 shows the result of the experiments under some history's length in the HalfCheetah task. Proposed-RL had the robustness to the change of the history's length and the highest performance in all conditions. The performance of PCHIST-RL degrades as the history length increases.

## 4.5 Discussion

The fact that the proposed method could efficiently construct the controller with high performance indicates that it could extract useful features for control. The reason why Simple-RL did not acquire the better policy was that it could not obtain essential information about randomized domain from only one step observation, therefore, it learned the overly generic controller for domain randomized environments. While PCHIST-RL achieved the better performance than PCLSTM-RL in pendulum swing up, it was inferior to PCLSTM-RL in HalfCheetah and logistics cart transportation. This might be influenced by the complexity of the environment. In other words, it is possible that in pendulum swing up, sufficient features could be extracted without using LSTM, while in HalfCheetah or logistics cart transportation, LSTM could extract favorable features to learn the better controller. PCLSTM-RL failed in some seeds and worked well in others, indicating that it was stuck in a local optimum. Some researchers suggested that deepening the neural network layer naively often deteriorates the performance of the policy [88, 89, 90]. It is possible that the performance deterioration of PCLSTM-RL controller was caused by increasing the number of layers in the network because of LSTM architecture. On the other hand, our architecture decoupled the reinforcement learning module and the feature extractor module, therefore, we consider that it avoided the performance deterioration and extracting the features by LSTM improved the performance of the controller.

The results of experiments for logistics cart transportation in Fig. 4.10 indicates that Proposed-RL controller was more robust than NoRand-RL controller. Also, blue bars in Fig. 4.10 suggests that degradation due to domain randomization of the controller was small.

The performance degradation of PCHIST-RL with longer history in Fig. 4.11 suggests that long history makes it difficult to extract essential information from it. We consider that the results of other methods show that using LSTM helped to extract essential information from long history.

## 4.6 Summary

In this chapter, we proposed a efficient reinforcement learning method in a domain randomized environment. Through three simulation experiments, we confirmed that the feature extractor could extract favorable features by LSTM from historical data and learned the policy efficiently.

In addition, we shown that using historical data and privileged information could obtain the better policy, while previous methods of on-policy reinforcement learning in a domain randomized environment have compared the policy using only one step observation with the LSTM-based policy and shown the effectiveness of LSTM architecture.

One of the future challenges is how to determine the range of the randomized domain, and how to acquire the controller appropriately without collapsing its performance because of the large domain randomization.

# Chapter 5

# Conclusion

## 5.1 Summary of contributions

In this dissertation, we have introduced the methods for efficient reinforcement learning in a simulation environment for robot control. The goal of this research is to make it easier to apply reinforcement learning to real robotic systems by reducing the learning cost of reinforcement learning. For efficient learning, a lot of research used a physical simulation. However, in previous works, useful information from simulation and knowledge of control is not utilized enough. Although using a simulation reduces learning cost, the difference between simulation and real-world often make it difficult to transfer a reinforcement learning controller from a simulation to real-world. Therefore, we developed efficient reinforcement learning methods utilizing various information. Also, we evaluated these through logistics cart transportation task and some simulation task. The following is a summary of contributions.

In Chapter 2, we have proposed a method to improve the learning efficiency by residual reinforcement learning using control knowledge in a physical simulation. Using the control knowledge improved the learning efficiency and made learning process stable. Furthermore, the use of the simulation reduced the cost of data collection. The results of experiments shown that the reinforcement learning controller learned in the physical simulation environment can be transferred to control robots in a real-world environment.

In Chapter 3, we have introduced a method to improve the learning efficiency and the performance of a reinforcement learning controller in a delayed feedback environment. The delayed feedback occurs in robot control in a real-world environment and deteriorates the reinforcement learning efficiency. Many general reinforcement learning methods assume a Markov decision process (MDP). However, in a real-world environment, an agent can only obtain a delayed observation. In the proposed method, the agent uses the undelayed observations and rewards as privileged information and incorporate them into the learning process to make learning the controller more efficient. In the experiments, we applied the proposed method to the three simulation task and logistics cart transportation as in Chapter 2. The results of experiments shown that the proposed method can efficiently learn and obtain the controller stably.

In Chapter 4, we have introduced two-stage learning algorithm to learn the policy efficiently in a domain randomized environment for transferring it to a real-world environment. Domain randomization is a method that randomly samples textual noise of an image and/or environment parameters (friction, weight, etc.). Learning in randomly sampled environment prompts to obtain a robust controller. However, randomization makes the problem more complicated, and it may not be possible to obtain the controller that can use in a practical situation. Our two-stage learning algorithms includes learning the feature extractor and the reinforcement learning controller. Our proposed method adopted LSTM for extracting the environment feature from observation and action histories. We evaluated the proposed method in the pendulum swing up, HalfCheetah and logistics cart transportation tasks. The results of simulation experiments shown that agents learned by the proposed method obtained the favorable policy more efficient than other methods. Also, the results of real-world experiments suggests that our proposed architecture with domain randomization is useful for sim-to-real transfer.

We proposed efficient reinforcement learning methods in the simulation environment for realizing robot control in the real-world environment. This dissertation makes it easy to introduce reinforcement learning into real robot control.

## 5.2 Future works

Finally, we discuss remaining challenges related to this dissertation and to reinforcement learning research using a physical simulation for robot control.

### Different formulations for residual reinforcement learning

We have formulated the output of the residual reinforcement learning controller as the sum of the output of the base controller and the output of the reinforcement learning controller.

However, other architecture, for example, the output of the residual reinforcement learning controller as the product of the output of the base controller and the output of the reinforcement learning controller or adding a gate mechanism is possibly effective to improve the performance.

### Accurate state predictor for a delayed feedback environment

We have shown that undelayed feedback and using predictor can efficiently learn a reinforcement learning controller with higher performance than conventional methods, however, performance degradation is inevitable. To minimize the performance degradation, an accurate predictor model is necessary. In this dissertation, we used the simple fully connected neural network, but we believe that there is room for improvement this degradation.

Model-based reinforcement learning is based on learning models that generate state transitions, and various models for generating state transitions can be candidates. There are two main directions: models that consider all previous observations and models that consider only one previous step. In SimPLe [91], the next observation is estimated from the observations of the past four steps. It also incorporates a stochastic latent variable and discretizes it. The generative model Z-forcing [92] for series data used in [93] is a model that considers all past observations and incorporates stochastic latent variables and deterministic latent variables by Recurrent Neural Networks (RNNs). It also improves performance by forcing the system to consider future series data when training the probabilistic latent variables so that they contain meaningful information and useful information for estimating the long-term future. In PlaNet [94], the authors propose the Recurrent State-Space Model (RSSM), which learns probabilistic latent representations by Variational Auto Encoder (VAE) from deterministic latent representations by RNNs. They also propose a method for learning to estimate the state of multiple steps in the future. On the other hand, [95] is a model-based reinforcement learning that uses a model that considers only one past step. In this study, the authors showed that a 1-step state-transition generation model with stochastic latent variables can achieve the same or better performance faster than RNNs. The algorithm used to obtain the controller is Imagination-Augmented Agents (I2A) [96].

### Comparison with on-policy method with recurrent neural network

In this dissertation, we have focused on off-policy reinforcement learning, however, an on-policy reinforcement learning method can be more suitable for learning a controller with time series data. On-policy reinforcement learning is a method in which a behavioral policy and a learning policy are matched, and it does not use experience replay. Therefore, on-policy reinforcement learning is considered to be less sample efficiency than off-policy reinforcement learning. However, it works well with RNNs because it does not use experience replay. In Chapter 4, we used short observation and action histories for training the off-policy reinforcement learning controller, however, we consider that it is necessary to compare on-policy and off-policy methods from various perspectives such as computational resources and the performance of the policy.

### Robustness to the amount of the delay step and random delay

In Chapter 3, we have proposed an efficient reinforcement learning algorithm in an environment with a constant step of delay. On the other hand, a random step of delay is often encountered in a real-world environment. Our proposed method had some robustness to the shift of delay step in some tasks. However, as shown in Chapter 3, the proposed method was insufficient to deal with the shift of delay step in the others. Therefore, it is necessary to develop an algorithm that deal with a random delayed feedback environment, and an algorithm that is robust to the shift of the delay step.

### Selection of the model architecture for extracting feature

In Chapter 4, we have adopted the feature extractor based on LSTM, however, there are various choices for extracting feature representation, for example, a model that focuses on a local part of observation and action histories, such as Convolutional Neural Networks (CNNs) [97]. Also, we believe that methods extracting stochastic feature representation is promising. It is possible that stochastic feature representation can represent incomplete information. It is necessary to examine these models and find a model that is more suitable for extracting state representation from the observation and action histories.

### Randomized range and selection of environment parameters

In Chapter 4, we have focused on domain randomization to improve the transfer to the real-world environment. We believe that the remaining major work in domain randomization are the selection of appropriate environment parameters for randomization and the setting of the randomization ranges. We consider that the performance in the domain randomized simulation is likely to be poor and the transfer to the real environment will not be successful if the randomization is inappropriate. Alternatively, it is possible to develop a method to select an appropriate range that can be solved from a large randomization range and learn within this range.

# Bibliography

[1] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[2] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[3] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[4] Shixiang Gu et al. “Continuous deep q-learning with model-based acceleration”. In: *International conference on machine learning*. PMLR. 2016, pp. 2829–2838.

[5] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[6] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1861–1870.

[7] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1587–1596.

[8] Alex Kendall et al. “Learning to drive in a day”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8248–8254.

[9] Dmitry Kalashnikov et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *arXiv preprint arXiv:1806.10293* (2018).

[10] Andy Zeng et al. “Tossingbot: Learning to throw arbitrary objects with residual physics”. In: *IEEE Transactions on Robotics* 36.4 (2020), pp. 1307–1319.

[11] Jemin Hwangbo et al. “Control of a quadrotor with reinforcement learning”. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[12] Jemin Hwangbo et al. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* 4.26 (2019).

[13] Lei Tai, Giuseppe Paolo, and Ming Liu. “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 31–36.

[14] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[15] Shital Shah et al. “Airsim: High-fidelity visual and physical simulation for autonomous vehicles”. In: *Field and service robotics*. Springer. 2018, pp. 621–635.

[16] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[17] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[18] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. `http://pybullet.org`. 2016–2021.

[19] Eric Rohmer, Surya PN Singh, and Marc Freese. “V-REP: A versatile and scalable robot simulation framework”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1321–1326.

[20] Nathan Koenig and Andrew Howard. “Design and use paradigms for gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[21] Tom Erez, Yuval Tassa, and Emanuel Todorov. “Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx”. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 4397–4404.

[22] Jack Collins et al. “A Review of Physics Simulators for Robotic Applications”. In: *IEEE Access* (2021).

[23] Markus Giftthaler et al. “Automatic differentiation of rigid body dynamics for optimal control and estimation”. In: *Advanced Robotics* 31.22 (2017), pp. 1225–1237.

[24] Eric Heiden et al. “NeuralSim: Augmenting differentiable simulators with neural networks”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 9474–9481.

[25] Jean Feng and Noah Simon. “Sparse-input neural networks for high-dimensional nonparametric regression and classification”. In: *arXiv preprint arXiv:1711.07592* (2017).

[26] Taichi Kumagai, Shinya Yasuda, and Hiroshi Yoshida. “A prototype of a cooperative conveyance system by wireless-network control of multiple robots”. In: *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*. Vol. 1. IEEE. 2019, pp. 231–236.

[27] Tobias Johannink et al. “Residual reinforcement learning for robot control”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 6023–6029.

[28] Tom Silver et al. “Residual policy learning”. In: *arXiv preprint arXiv:1812.06298* (2018).

[29] Tianyu Li et al. “Using deep reinforcement learning to learn high-level policies on the atrias biped”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 263–269.

[30] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. “Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications”. In: *IEEE transactions on cybernetics* 50.9 (2020), pp. 3826–3839.

[31] Ryan Lowe et al. “Multi-agent actor-critic for mixed cooperative-competitive environments”. In: *arXiv preprint arXiv:1706.02275* (2017).

[32] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. “Cooperative multi-agent control using deep reinforcement learning”. In: *International Conference on Autonomous Agents and Multiagent Systems*. Springer. 2017, pp. 66–83.

[33] Elio Tuci, Muhanad HM Alkilabi, and Otar Akanyeti. “Cooperative object transport in multi-robot systems: A review of the state-of-the-art”. In: *Frontiers in Robotics and AI* 5 (2018), p. 59.

[34] Fusao Ohashi et al. “Transportation of a large object by small mobile robots with handcarts and outrigger”. In: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. IEEE. 2014, pp. 70–75.

[35] Kazuhiro Kosuge and Tomohiro Oosumi. “Decentralized control of multiple robots handling an object”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS’96*. Vol. 1. IEEE. 1996, pp. 318–323.

[36] Ashley Stroupe et al. “Behavior-based multi-robot collaboration for autonomous construction tasks”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 1495–1500.

[37] Toni Machado et al. “Multi-constrained joint transportation tasks by teams of autonomous mobile robots using a dynamical systems approach”. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 3111–3117.

[38] Alpaslan Yufka and Metin Ozkan. “Formation-based control scheme for cooperative transportation by multiple mobile robots”. In: *International Journal of Advanced Robotic Systems* 12.9 (2015), p. 120.

[39] Russell G Brown and James S Jennings. “A pusher/steerer model for strongly cooperative mobile robot manipulation”. In: *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*. Vol. 3. IEEE. 1995, pp. 562–568.

[40] ZhiDong Wang, Yasuhisa Hirata, and Kazuhiro Kosuge. “Control multiple mobile robots for object caging and manipulation”. In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 2. IEEE. 2003, pp. 1751–1756.

[41] Weiwei Wan et al. “Multirobot object transport via robust caging”. In: *IEEE transactions on systems, man, and cybernetics: systems* 50.1 (2017), pp. 270–280.

[42] Wenya Wan, Chong Sun, and Jianping Yuan. “The caging configuration design and optimization for planar moving objects using multi-fingered mechanism”. In: *Advanced Robotics* 33.18 (2019), pp. 925–943.

[43] Elon Rimon and Andrew Blake. “Caging planar bodies by one-parameter two-fingered gripping systems”. In: *The International Journal of Robotics Research* 18.3 (1999), pp. 299–318.

[44] Peam Pipattanasomporn and Attawith Sudsang. “Two-finger caging of concave polygon”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 2137–2142.

[45] Thomas F Allen, Joel W Burdick, and Elon Rimon. "Two-finger caging of polygonal objects using contact space search". In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1164–1179.

[46] Satoshi Makita and Weiwei Wan. "A survey of robotic caging and its applications". In: *Advanced Robotics* 31.19-20 (2017), pp. 1071–1085.

[47] Lin Zhang et al. "Decentralized Control of Multi-Robot System in Cooperative Object Transportation Using Deep Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 184109–184119.

[48] Joohyun Woo, Chanwoo Yu, and Nakwan Kim. "Deep reinforcement learning-based controller for path following of an unmanned surface vehicle". In: *Ocean Engineering* 183 (2019), pp. 155–166.

[49] Peide Cai et al. "High-speed autonomous drifting with deep reinforcement learning". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1247–1254.

[50] Yutaka Kanayama et al. "A stable tracking control method for an autonomous mobile robot". In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE. 1990, pp. 384–389.

[51] James Bergstra et al. "Algorithms for hyper-parameter optimization". In: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation. 2011.

[52] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[53] David Silver et al. "Deterministic policy gradient algorithms". In: *International conference on machine learning*. PMLR. 2014, pp. 387–395.

[54] Tom Schaul et al. "Prioritized experience replay". In: *International Conference on Learning Representations (ICLR)*. 2016.

[55] Dan Horgan et al. "Distributed prioritized experience replay". In: *International Conference on Learning Representations (ICLR)*. 2018.

[56] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *ICLR (Poster)*. 2015.

[57] K. V. Katsikopoulos and S. E. Engelbrecht. "Markov decision processes with delays and asynchronous cost collection". In: *IEEE transactions on automatic control* 48.4 (2003), pp. 568–574.

[58] J. Tan et al. “Sim-to-real: Learning agile locomotion for quadruped robots”. In: *arXiv preprint arXiv:1804.10332* (2018).

[59] G. Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[60] T. J Walsh et al. “Learning and planning in environments with delayed feedback”. In: *Autonomous Agents and Multi-Agent Systems* 18.1 (2009), pp. 83–105.

[61] Y. Bouteiller et al. “Reinforcement Learning with Random Delays”. In: *International Conference on Learning Representations*. 2020.

[62] J. Foerster et al. “Counterfactual multi-agent policy gradients”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[63] L. Pinto et al. “Asymmetric Actor Critic for Image-Based Robot Learning”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018. DOI: 10.15607/RSS.2018.XIV.008.

[64] D. Chen et al. “Learning by cheating”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 66–75.

[65] J. Lee et al. “Learning quadrupedal locomotion over challenging terrain”. In: *Science robotics* 5.47 (2020).

[66] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Icml*. 2010.

[67] Manuel Kaspar, Juan D Muñoz Osorio, and Jürgen Bock. “Sim2real transfer for reinforcement learning without dynamics randomization”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 4383–4388.

[68] Nikolaus Hansen and Andreas Ostermeier. “Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation”. In: *Proceedings of IEEE international conference on evolutionary computation*. IEEE. 1996, pp. 312–317.

[69] Florian Golemo et al. “Sim-to-real transfer with neural-augmented robot simulation”. In: *Conference on Robot Learning*. PMLR. 2018, pp. 817–828.

[70] Josh Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[71] Irina Higgins et al. “Darla: Improving zero-shot transfer in reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1480–1490.

[72] Irina Higgins et al. “Beta-vae: Learning basic visual concepts with a constrained variational framework”. In: *ICLR (Poster)*. PMLR. 2017.

[73] Jinwei Xing et al. “Domain Adaptation In Reinforcement Learning Via Latent Unified State Representation”. In: *arXiv preprint arXiv:2102.05714* (2021).

[74] Ananya Harsh Jha et al. “Disentangling factors of variation with cycle-consistent variational auto-encoders”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 805–820.

[75] Stephen James et al. “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12627–12637.

[76] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1125–1134.

[77] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[78] Xue Bin Peng et al. “Sim-to-real transfer of robotic control with dynamics randomization”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 3803–3810.

[79] Marcin Andrychowicz et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[80] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[81] Wenhao Yu et al. “Preparing for the unknown: Learning a universal policy with online system identification”. In: *arXiv preprint arXiv:1702.02453* (2017).

[82] Yevgen Chebotar et al. “Closing the sim-to-real loop: Adapting simulation randomization with real world experience”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8973–8979.

[83] Jan Peters, Katharina Mulling, and Yasemin Altun. “Relative entropy policy search”. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[84] Bhairav Mehta et al. “Active domain randomization”. In: *Conference on Robot Learning*. PMLR. 2020, pp. 1162–1176.

[85] Jelle Munk, Jens Kober, and Robert Babuška. “Learning state representation for deep actor-critic control”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 4667–4673.

[86] Kei Ota et al. “Can Increasing Input Dimensionality Improve Deep Reinforcement Learning?” In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7424–7433.

[87] Lingheng Meng, Rob Gorbet, and Dana Kulić. “Memory-based Deep Reinforcement Learning for POMDP”. In: *arXiv preprint arXiv:2102.12344* (2021).

[88] Peter Henderson et al. “Deep reinforcement learning that matters”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[89] Samarth Sinha et al. “D2rl: Deep dense architectures in reinforcement learning”. In: *arXiv preprint arXiv:2010.09163* (2020).

[90] Kei Ota, Devesh K Jha, and Asako Kanezaki. “Training Larger Networks for Deep Reinforcement Learning”. In: *arXiv preprint arXiv:2102.07920* (2021).

[91] Lukasz Kaiser et al. “Model-based reinforcement learning for atari”. In: *arXiv preprint arXiv:1903.00374* (2019).

[92] Anirudh Goyal et al. “Z-forcing: Training stochastic recurrent networks”. In: *arXiv preprint arXiv:1711.05411* (2017).

[93] Nan Rosemary Ke et al. “Modeling the long term future in model-based reinforcement learning”. In: *International Conference on Learning Representations*. 2018.

[94] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2555–2565.

[95] Lars Buesing et al. “Learning and querying fast generative models for reinforcement learning”. In: *arXiv preprint arXiv:1802.03006* (2018).

[96] Sébastien Racanière et al. “Imagination-augmented agents for deep reinforcement learning”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 5694–5705.

[97] Kunihiko Fukushima and Sei Miyake. “Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition”. In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

# AppendixA

# Details of formation-based feedback controller formulation

The calculation process of the formation-based feedback controller is written below. The positions defined in the below equation are in local coordinates at time $k$.

$$\begin{aligned}
\theta_{\mathrm{tgt}}^{\mathrm{sr}}(k) &= \pi + \tan^{-1}\frac{y_{\mathrm{tgt}}^{\mathrm{sr}}(k) - y_{\mathrm{tgt}}^{\mathrm{sr}}(k-1)}{x_{\mathrm{tgt}}^{\mathrm{sr}}(k) - x_{\mathrm{tgt}}^{\mathrm{sr}}(k-1)} \\
&= \pi + \tan^{-1}\frac{y_{\mathrm{tgt}}^{\mathrm{sr}}(k) - y_{\mathrm{tgt}}^{\mathrm{sr}}(k-1)}{x_{\mathrm{tgt}}^{\mathrm{sr}}(k) - x_{\mathrm{tgt}}^{\mathrm{sr}}(k-1)} - \tan^{-1}\frac{y_{\mathrm{tgt}}^{\mathrm{VL}}(k-1)}{x_{\mathrm{tgt}}^{\mathrm{VL}}(k-1)} + \tan^{-1}\frac{y_{\mathrm{tgt}}^{\mathrm{VL}}(k-1)}{x_{\mathrm{tgt}}^{\mathrm{VL}}(k-1)} \\
&= \pi + \tan^{-1}\frac{-R(1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}) + L\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}}{R\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}} + L(1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}})} + \tan^{-1}\frac{R\left(1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}\right)}{R\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}} - \frac{\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}}{2} \\
&= \pi + \tan^{-1}\left(-\frac{(1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}) + \rho L\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}}{\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}} + \rho L(1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}})}\right) + \tan^{-1}\frac{1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}}{\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}} - \frac{\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}}{2} \\
&= \pi + \rho L - \frac{\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}}{2} \qquad \text{(A.1)}
\end{aligned}$$

The calculation from lines 4 to 5 utilizes the sum of angle identities. $L$ is defined as in Fig. 2.3 and the positions are defined as

$$\boldsymbol{x}_{\mathrm{tgt}}^{\mathrm{sr}}(k) = \left[x_{\mathrm{tgt}}^{\mathrm{sr}}(k), y_{\mathrm{tgt}}^{\mathrm{sr}}(k)\right] = [L, 0]^{\top},$$

$$\begin{aligned}
\boldsymbol{x}_{\mathrm{tgt}}^{\mathrm{sr}}(k-1) &= \boldsymbol{x}_{\mathrm{tgt}}^{\mathrm{VL}}(k-1) + \left[L\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}, -L\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}\right]^{\top} \\
&= \left[-R\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}} + L\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}, R(1-\cos\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}) - L\sin\omega_{\mathrm{tgt}}^{\mathrm{VL}}t_{\mathrm{s}}\right]^{\top}.
\end{aligned}$$

# Acknowledgements

本論文の執筆にあたり、お力添えいただいた方々に心より感謝申し上げます。

指導教員である矢入健久先生には、修士課程からの 5 年間大変お世話になりました。研究に関するアドバイスや共同研究の紹介など、本研究を遂行するにあたって重要な機会を多くいただきました。

堀浩一先生、中須賀真一先生、土屋武司先生、太田順先生には、本論文の副査として、中間審査などで多くの助言いただき、博士論文を執筆するうえで大変助けになりました。堀先生には、研究室での研究会等においても修士課程からの 5 年間で多くの助言をいただき、大変お世話になりました。

共同研究として、NEC システムプラットフォーム研究所の金友大氏、吉田裕志氏、熊谷太一氏、安田真也氏、佐藤夏彦氏には様々な助言や実験の機会を頂きました。特に、安田氏には実験に際して多くのサポートを頂き、大変お世話になりました。

研究室のメンバーとは、研究会等で様々な議論ができ、本研究を進めるうえで助けになりました。

最後に、あらゆる面で支えてくれた家族に心より感謝いたします。