Master of Science Thesis

Deep recurrent neural network-based reinforcement learning technique for controlling quadrotors with unknown physical quantities

임의의 물리량을 가지는 쿼드로터 제어를 위한 심층 순환 신경망 기반 강화 학습 기법

February 2023

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major

Jae-Kyung Cho

Master of Science Thesis

Deep recurrent neural network-based reinforcement learning technique for controlling quadrotors with unknown physical quantities

임의의 물리량을 가지는 쿼드로터 제어를 위한 심층 순환 신경망 기반 강화 학습 기법

February 2023

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major

Jae-Kyung Cho

Deep recurrent neural network-based reinforcement learning technique for controlling quadrotors with unknown physical quantities

임의의 물리량을 가지는 쿼드로터 제어를 위한 심층 순환 신경망 기반 강화 학습 기법

Supervisor: Seung-Woo Seo, Seong-Woo Kim

This work is submitted as a Master of Science Thesis

December 2022

Graduate School of Engineering
Seoul National University
Electrical and Computer Engineering Major

Jae-Kyung Cho

Confirming the master's thesis written by
Jae-Kyung Cho

December 2022

Chair	Eun Suk Suh	(Seal)
Vice Chair	Seung-Woo Seo	(Seal)
Examiner	Seong-Woo Kim	(Seal)

Abstract

This thesis proposes a deep recurrent neural network-based controller for the quadrotor with reinforcement learning. The robot controller can be defined as an agent producing motor control action directly from the raw state of the robot. The controller needs to be fine-tuned according to the dynamics model of the robot being controlled because the dynamics model determines the state change when an action is executed. The dynamics model of all real quadrotors inevitably differs even if they are the same product because the physical quantities are uncertain. In particular, the dynamics can change significantly during the flight due to overheating motors or propeller damage. The objective of our low-level controller is to maintain its performance while changes in the dynamics model without prior knowledge or fine-tuning of the parameters.

To solve the problem, a reinforcement learning (RL) based controller including a recurrent neural network (RNN) structure is proposed. RL is used to train the controller by data-driven from the environment instead of mathematical modeling. Furthermore, RNNs help extracts information about the dynamics model from the state–action history sequence. However, learning is not performed by simply including the RNN in the RL loop since the quadrotor is not stable enough to get good data by random exploration. We proposed a method to increase learning stability by separating a dynamics extractor module that includes an RNN structure from the RL loop. The dynamic extractor is trained to predict dynamics information from the state-action sequence in a supervised-learning manner, and the actor-critic of RL is trained with the ground truth of the dynamics information provided by the simulator.

The proposed method is the first study to apply RNNs for the low-level controller of the quadrotor, and outperform the existing model-based controller and feed-forward network-based controller in the simulation environment. The training process is conducted in the simulation called Gym-pybullet-drone which can randomize all the pos-

sible quadrotor dynamics parameters that may affect the controller performance. Although training is conducted in the simulator, all hardware constraints are satisfied to verify the applicability to real drones. Further research is needed to verify and improve its performance using actual drones.

keywords: Quadrotor controller, recurrent neural network, reinforcement learning, robotics, deep learning

student number: 2020-27508

Contents

Abstract **i**

Contents **iii**

List of Figures **v**

List of Tables **vi**

1 Introduction **1**

1.1 Motivation 1

1.2 Contributions 5

1.3 Organizations 6

2 Related work **7**

2.1 Reinforcement learning 7

2.1.1 Policy gradient reinforcement learning 10

2.2 Memory-based robot controller 11

2.3 Quadrotor controller 12

2.3.1 Rule-based quadrotor controller 12

2.3.2 Learning-based quadrotor controller 13

3 Memory-based robot controller **16**

3.1 Introduction 16

3.2 Approach 18
3.3 Experimantal results 19

4 Memory-based quadrotor controller **25**
4.1 Introduction 25
4.2 Approach 25
4.2.1 Problem definition 25
4.2.2 Simulator setting and dynamics randomization 26
4.2.3 Actor-critic model 29
4.3 Experimental results 32
4.3.1 Stabilization experiment 35

5 Conclusion **42**
5.1 Conclusion 42

Bibliography **44**

Abstract in Korean **50**

List of Figures

1.1 Training step framework 3
1.2 Execution step framework 4
1.3 Trajectory comparison 5

3.1 Inverted-Pendulum environment 17
3.2 RNNfull actor-critic structure 22
3.3 RNNpolicy actor-critic structure 23
3.4 Learning curve of Inverted-Pendulum 24

4.1 Quadrotor simulator and the real quadrotor model 27
4.2 RNNparam actor-critic structure 29
4.3 Learning curve of quadrotor controllers 34
4.4 Trajectory of stabilizing experiment 39

List of Tables

3.1 Range of dynamics parameter for Inverted-Pendulum 17

3.2 RNNpolicy hyperparameters 20

4.1 The randomization range of dynamics parameters and initial state in Gym-pybullet-drones environment 28

4.2 RNNparam hyperparameters 33

4.3 Stabilizing quadrotor experiment result 36

Chapter 1

Introduction

1.1 Motivation

A closed-loop low-level controller for a robot is built by considering its own dynamics model. Although a fine-tuned controller for one dynamics model usually operates well for similar dynamic models, the performance will not be maintained if the dynamics change significantly. Even if the robots appear homogeneous in the real world, they inevitably differ in terms of the mass of the links, joint friction, motor torque curves, and the gain of controllers. In other words, the dynamics model of a real robot will be different from the one that was originally modeled. In the case of quadrotors, the state transition model continuously changes during flight, motor performance deteriorates due to overheating, and thrust and torque can change due to propeller damage. Moreover, the movement of ancillary items mounted on the quadrotor, such as gimbal cameras and delivery items, can cause changes to the center of mass and moment of inertia. All these factors could disturb the pre-fine-tuned controller from performing optimally. Therefore, information about the dynamics must be obtained through continuous interaction with the environment, and a policy should be enacted that can derive optimal actions considering different dynamics models.

Deep Reinforcement Learning (DRL) has been used to solve control problems by

collecting data from interactions with the environment. In particular, DRL can be more effective than a model-based controller for tasks that requires complex and sensitive modeling, such as stabilizing the quadrotors. Model-free RL methods [1, 2, 3], and model-based RL methods [4] have been proposed for training a controller consisting of feed-forward neural networks to control quadrotors. Although these controllers, which were composed of feed-forward neural networks, could operate robustly without accurate dynamics information, it was challenging to achieve customized performance for different dynamics models. In other words, feed-forward networks are not aware of the change in dynamics and it leads to deterioration of the controller performance. The use of recurrent neural networks (RNNs) has been suggested as a method for extracting unknown dynamics model information through continuous interaction with the environment [5]. However, the performance of RNNs has only been verified in relatively stable robots, such as robotic arms [5, 6]. Accordingly, we were interested in whether a low-level controller trained by this method could respond to changes in the dynamics model of the quadrotor.

Pascanu *et al*. [7] showed the high learning instability of RNNs. We experimentally confirmed whether the RNN structure can respond to dynamics changes in the simplest robot environment called Inverted-Pendulum before applying to the quadrotor controller. Instability problems such as high hyper-parameter sensitivity are occurred when learning a structure including an RNN in the end-to-end manner using an RL loop. In order to improve stability, we propose a method that does not include RNN in the critic of the actor-critic structure, which is not actually used for the execution step. Furthermore, we verify that gated recurrent unit (GRU) improves the learning performance most effectively among various types of RNN.

In this thesis, we propose an RNN-based low-level controller for quadrotors that extracts dynamics information from the state–action history sequence to respond to dynamics model changes. To the best of our knowledge, this is the first attempt at applying RNN to a low-level quadrotor controller. In particular, the quadrotor is difficult

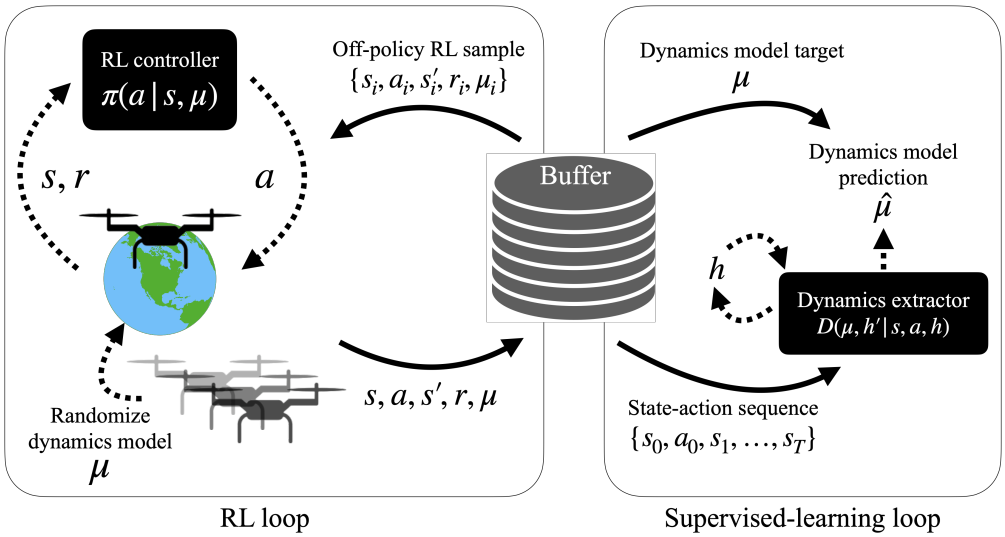


Figure 1.1: Training step framework. The dynamics model of the quadrotor which is expressed as parameter μ is randomized for every episode of the RL loop. The RL controller produces an action based on the current state s and ground truth of dynamics model parameters μ. Rolled out data is stored in the buffer, and the RL controller is learned using the off-policy RL algorithm. The dynamic extractor including the RNN structure predicts the dynamics model parameters from the state–action sequence. The supervised-learning method with the dynamics model ground truth stored in the buffer as a target is used for training of the dynamics extractor.

to obtain good data through random exploration, and the control frequency is high which makes the length of the state-action sequence long. Because of these characteristics, it does not work to embed RNNs into actor and critic in an RL loop and to train end-to-end manner. Therefore, we divided the entire policy module into the dynamic extractor module comprising the RNN and the RL controller composed of feed-forward networks. Fig. 1.1 displays the training step overview of the separate modules. The RL controller composed with feed-forward networks produces actions based on the current state and the information about dynamics model. The use of the ground truth for the dynamics model enables stable training of the RL loop. The dynamics

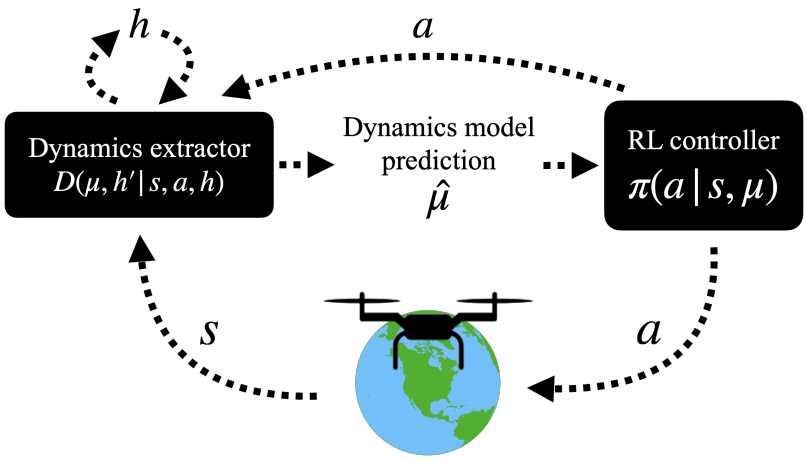


Figure 1.2: Execution step framework. Since dynamics model information cannot be known in the execution stage, the dynamics extractor predicts the dynamics model $\hat{\mu}$. Based on the predicted dynamics information, the RL Controller produces the final action.

extractor module directly predicts dynamics parameters from state–action sequences using RNN in supervised-learning manner. When learning is completed, the RL controller produces an action based on the dynamics model information predicted through the dynamics extractor, as shown in Fig. 1.2. With the proposed method, it is possible to maintain the control performance in response to changes in dynamics.

Fig. 1.3 displays the improved performance of the proposed method in an example scenario, which aims to recover from a flipped state and reach the red dot regardless of the orientation of the quadrotor, where the quadrotor is flipped and the performance of one motor is reduced by 30% due to overheating. The proposed RNN-based controller exhibits faster recovery from the flipped status and a shorter trajectory compared to the feed-forward networks-based controller.

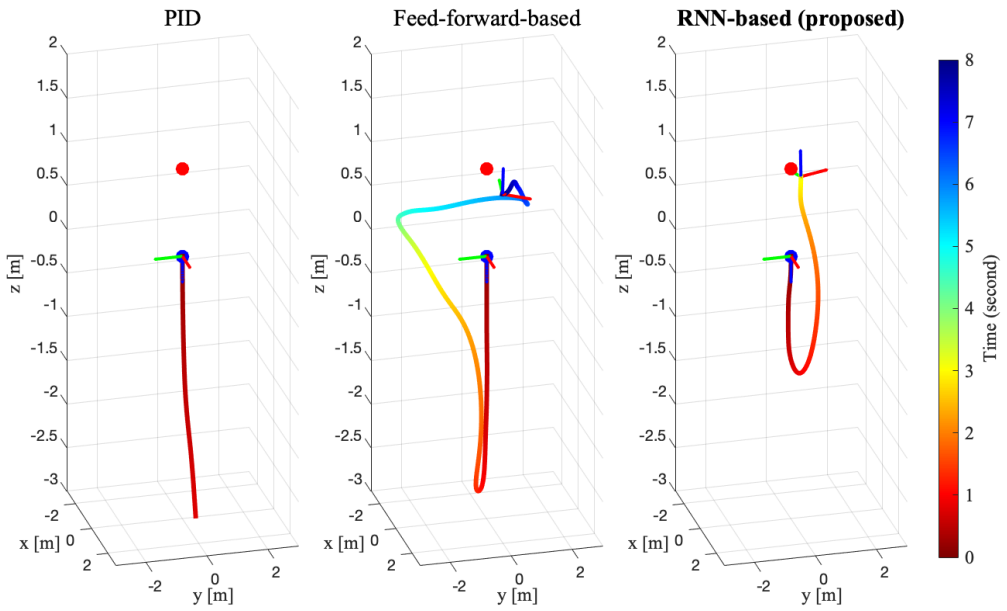


Figure 1.3: Comparison of controlled trajectories of the proposed method and existing methods in a dynamics change scenario. The scenario aims to control quadrotors to their target positions when the performance of one motor is reduced by 30%. Quadrotors are initialized as flipped status (blue dots) and aim to move to the goal (red dots).

1.2 Contributions

The main contributions are summarized below.

Firstly, the types of RNNs and the structure that is advantageous for dynamics information extraction are presented in Chapter 3. In the simple robot simulation environment, inverted pendulum, learning was carried out with the Off-policy RL algorithm and the actor-critic structure model grafted with RNN. This revealed the structure that uses Gated Recurrent Unit (GRU), a type of RNNs, only for the actor is the most efficient for dynamics information extraction.

Secondly, a new RNN-based controller for the quadrotor is presented in Chapter 4. The learning instability problem of RNNs is experimentally shown in the quadrotor environment. Therefore, the training process is separated to learn the models com-

posed of feed-forward networks with RL, and additionally learn the dynamics extractor module including the RNN structure. With the dynamics extractor, the controller successfully responded to changes in the dynamics model.

Lastly, a controller model that can be used in the real-world is proposed. The modified quadrotor simulation is suggested to allow randomization of the dynamics model and addition of the sensor noise, which enables to make it close to the real world. Furthermore, the hardware constraints are satisfied including memory size and computational speed while maintaining the performance.

1.3 Organizations

The rest of the thesis is organized as follows. Chapter 1 reviews the basic concept of reinforcement learning, and the memory-based robot controller under the unknown dynamics model. In addition, state-of-the-art studies of two types of quadrotor controller, rule-based and learning-based, is introduced. Chapter 3 addresses the efficient structure of robot controllers using RNNs which can operate under the unknown dynamics model. Chapter 4 presents the quadrotor-specified controller using RNNs response to changes in the dynamics model and proposed a new learning method to improve the performance. Chapter 4 utilize the result of the Chapter 3. Lastly, Chapter 5 ends up the thesis with the conclusion and future works.

Chapter 2

Related work

This chapter introduces the background of reinforcement learning and robot control studies. In addition, existing works about using recurrent neural networks to deal with unknown information for robot control are described. Finally, the existing rule-based and learning-based quadrotor controllers are introduced.

2.1 Reinforcement learning

Reinforcement learning (RL) is one of the machine learning techniques. Unlike general machine learning techniques represented by supervised learning that require data to be obtained in advance, RL accumulates data through interaction with the environment. During the interaction, the agent receives feedback from the environment. Various types of feedback are possible, from binary feedback of success and failure to carefully designed feedback. The agent learns to create an action to acquire the goal based on the feedback. Reinforcement learning is very similar to human learning and has generality to obtain various behaviors that appear in AI and nature [8]. Deep RL (DRL) uses deep neural networks in the RL structure and achieved superhuman performances in various fields, such as video games [9, 10], strategy board games [11, 12], and robotics [13].

The standard RL problem can be described as finding a policy to maximize a

return under the Markov decision process (MDP) [14]. MDP is defined as 5-tuple $< \mathbb{S}, \mathbb{A}, p_\mu, r, \gamma >$ where the terms refer to state, action, transition probability, reward function, and discount factor, respectively. The state of the agent $s_t \in \mathbb{S}$ is the robot state at timestep t, which is commonly a vector of sensor measurement in robotic fields. The action of the agent $a_t \in \mathbb{A}$ is the control signal or high-level decision at timestep t. The action is derived by the function called policy $\pi(a|s) : \mathbb{S} \rightarrow \mathbb{A}$ and transit the state from s_t to s_{t+1}. Due to the stochasticity of the environment, the probability of transit state by action is determined by transition probability model $p_\mu(s_{t+1}|s_t, a_t)$. It should be noted that the transition probability depends on parameter μ, which is the set of physical quantities that affect the dynamics model. For instance in robotic arm, robot arm length, mass, and joint friction can be included the physical quantities. In the case of the quadrotor, mass, body size, and thrust-to-weight ratio of the propeller can be included. Since the dynamic model of each environment is usually fixed, generality is not lost even if the transition model is expressed as $p(s_{t+1}|s_t, a_t)$. The reward function $r : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$ returns a scalar value that indicates how valuable the action was taken in the state. For simplicity, the reward at timestep t is denoted as $r_t = r(s_t, a_t)$.

The objective of RL is to train the policy to get maximum cumulated future reward as below.

$$R(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \dots. \tag{2.1}$$

The discount factor γ indicates the degree of influence of the future reward on the current state, which is the value in range [0,1]. To summarize, the objective $J^\mu(\pi)$ for the learning process on the given MDP was to find an optimal policy π^* that maximizes the expected return

$$\begin{aligned} \pi^* &= argmax_\pi J^\mu(\pi), \\ J^\mu(\pi) &= \mathbb{E}_{s_0, a_0, s_1, a_1, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right], \end{aligned} \tag{2.2}$$

where the action $a_t \sim \pi(\cdot|s_t)$ and the next state $s_{t+1} \sim p_\mu(\cdot|s_t, a_t)$.

To solve this objective problem, RL proposes the concept of value function and

action–value function. The value function V_π and the action-value function Q_π expresses how valuable a particular state is, and how valuable a state–action pair is, respectively. These two value functions are formed as follows:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) | s_t = s \right], \tag{2.3}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) | s_t = s, a_t = a \right]. \tag{2.4}$$

These value functions are important because an optimal policy can be achieved based on the optimal action–value function as equation 2.5.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*_\pi(s,a). \tag{2.5}$$

Following equation are called the Bellman equation, which are the recursive form of two value functions.

$$V_\pi(s) = \mathbb{E}_\pi \left[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s \right], \tag{2.6}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a \right]. \tag{2.7}$$

The recursive form can be solved through iterative update of the dynamic programming manner. The solution to obtain a policy is as follows:

$$V_{\pi'}(s) = \max_a \sum_{s' \in \mathbb{S}} p(s'|s,a) \left[r + \gamma V_{\pi'}(s') \right]. \tag{2.8}$$

$$\pi'(s) \doteq \underset{a}{\operatorname{argmax}} \sum_{s' \in \mathbb{S}} p(s'|s,a) \left[r + \gamma V_\pi(s') \right], \tag{2.9}$$

The equation 2.8 and the equation 2.9 are denoted a policy evaluation and a policy improvement, respectively. Iterative update of both equation train the policy closer to the optimal policy in the equation 2.5.

2.1.1 Policy gradient reinforcement learning

Policy gradient (PG) using function approximation is a method for directly finding an optimal parameter ϕ^* of the parameterized policy π_θ that maximizes the objective function $\theta^* = argmax_\theta J^\mu(\pi_\theta)$ [15]. This method enables using RL in the environment with continuous action space. The gradient of the objective function $\nabla_\theta J^\mu(\pi_\theta)$ is estimated as follows:

$$\nabla_\theta J^\mu(\pi_\theta) \propto \mathbb{E}_\pi \left[\sum_a Q_\pi(s,a) \nabla \pi_\theta(a|s) \right]. \tag{2.10}$$

The optimal parameter is achieved by using the gradient ascent method. As shown in the equation 2.10, the action–value function Q_π is needed to calculate the gradient of the policy. Although classical PG method [16] used the return value for each episode instead of unknown action–value function, this method has the disadvantage of large variance. The actor-critic method [17] improves the high variance problem and increase learning speed by using a parameterized value function with an objective function as follows:

$$\underset{\psi}{\text{minimize}} \left[r + V_\psi(s') - V_\psi(s) \right]^2 \tag{2.11}$$

All parameterized policy gradient methods can be combined with deep neural networks, which is the powerful approximator [18]. The on-policy PG method, represented by A2C [19], TRPO [20] and PPO [21], only obtains data used for learning with the current policy. Although these methods guaranteed policy improvement in every training step, they suffered from low sample efficiency, which causes slow learning speed, and were even not trained in an environment where sparse rewards are given. The off-policy method was proposed to solve this problem, represented by SAC [22], DDPG [23], and TD3 [24], which use the replay buffer that accumulates the data obtained from the previous policy. The SAC algorithm used in this study has an actor-critic structure in addition to a stochastic policy with maximizing entropy for exploration.

2.2 Memory-based robot controller

MDP assumes that the full state can be achieved. However, it is common to obtain the state through sensor measurement in the real world, which is not the full state. This violates the basic assumption of RL that the system should maintain the Markov property, so the performance of the policy obtained by RL cannot achieve optimal performance. In other words, RL aims to maximize return under the Partially Observable MDP (POMDP) in real world. In POMDP, two types of information, observation and belief state, are added to the existing MDP [25]. Observation is incomplete state information that measures full state. Belief state b aims to estimate the true state by continuously updating as follows:

$$b'(s') = \eta O(o|s', a) \sum_{s \in \mathbb{S}} p(s'|s, a) b(s), \tag{2.12}$$

where the agent observes o with probability model $O(\cdot|s', a)$, and normalize value η. The recursive form of the equation 2.12 can be untied as follows:

$$\begin{aligned} b_t(s_t) &= \eta O(o_t|s_t, a_{t-1}) \sum_{s_{t-1} \in \mathbb{S}} p(s_t|s_{t-1}, a_{t-1}) \left[\eta O(o_{t-1}|s_{t-1}, a_{t-2}) \dots\right] \\ &= \mathbf{b}(o_t, a_{t-1}, o_{t-1}, \dots, a0, o_0), \end{aligned} \tag{2.13}$$

where $\mathbf{b}$ is an arbitrary belief function. In other words, the belief state estimates the true state by extracting information from past observations and actions, and RL learns based on the belief state, not observation.

RNN is a representative neural networks structure that extracts information from history and uses it as an input for the current inference. The RDPG method [5] combining DDPG and RNN showed that RNN can implicitly update belief state in the simulator environment. In particular, RNN-based RL shows the performance in the real world which is the POMDP. Peng *et al.* [6] used the dynamics randomization technique to find the optimal policy in each dynamics model, rather than finding a policy that operates robustly in various dynamics models. In the simulation, dynamics models were randomly sampled for each episode in a predefined range, and the

agent was trained with RDPG algorithm. The policy was a combination of a feed-forward branch and an LSTM branch, and the LSTM branch played a role in extracting dynamics model information from the state–action history. In other words, a policy $\pi(a_t|s_0, a_0, s_1, a_1, \ldots, s_t)$ that returns an action based on the previous state and action history was used instead of a policy $\pi(a_t|s_t)$ that returns an action for the given state immediately before. Accordingly, it was possible to obtain a policy that returns an optimal action, even if the dynamics model of the robot in the real world was different from that of the simulator and there is no relevant information.

2.3 Quadrotor controller

2.3.1 Rule-based quadrotor controller

The aim of a low-level controller is to determine the four motor signals so that the quadrotor can be controlled from the current position to the desired position. Li *et al*. [26] proposed a PID control method through quadrotor modeling and Ren *et al*. [27] presented a quadrotor-specified PID control method that performs position and altitude control in two steps. Mellinger *et al*. [28] suggested a quadrotor controller that generates a trajectory first by minimizing the fourth derivative of position and then controls through PID control. These PID-based closed-loop control methods do not require separate quadrotor dynamics modeling, but fine-tuning of the gain value is required.

Model predictive controller(MPC) has been proposed for optimum control for specific constraints when the dynamics model of the robot is known [29]. MPC generates an optimal action by using various conditions as a cost function, such as minimizing position error or minimizing the magnitude of acceleration. Bangura *et al*. [30], Islam *et al*. [31], and Benotsmane *et al*. [32] proposed a quadrotor-specified MPC controller by formulating the quadrotor's linear dynamic model. However, MPC has a problem when there is noise in physical quantities because the accurate dynamics model must

be defined. In addition, as the state dimension of the dynamics increases, the computational time greatly increases. In order to handle the uncertainty of physical quantities that were not properly reflected in the Quadrotor model in advance, Bouffard *et al*. and Li *et al*. proposed learning-based MPC(LBMPC). LBMPC improves the performance of the controller by updating the model to minimize the predicted state and action error.

Although the authors mathematically modeled a controller, accurate physical parameters such as the moment-of-inertia, thrust-to-torque coefficient, and laborious fine-tuning to determine the gain values of the controller were still required.

2.3.2 Learning-based quadrotor controller

Neural networks began to be used to approximate complex non-linear modeling of quadrotor controllers. Mohajerin *et al*. [33, 34] trained neural networks that approximates a quadrotor dynamics model, which maps altitude changes according to motor speed using RNNs. However, the authors did not propose a quadrotor controller using the trained model. Maqbool *et al*. [35] and Tran *et al*. [36] proposed methods for altitude control of quadrotors using feed-forward neural networks. The neural networks, which map motor signals from a state of a single timestep of a quadrotor, were trained for minimizing attitude error. Khosravian *et al*. [37] proposed a PID controller that continuously updates PID gain parameters during flight using RNN. In each situation, the optimal PID gain values that minimize the positional error and attitude error were found and then trained through a supervised-learning fashion. However, these controllers focused on that can adaptively operate on a single fixed dynamics model. Furthermore, the non-linear controllers were approximated by neural networks but still had to mathematically define the dynamics model well.

Instead of modeling the dynamics model, various methods using RL that learns based on data obtained through interaction with environment have been proposed. Hwanbo *et al*. [1] and Duisterhof *et al*. [3] suggested methods to train a low-level

controller by model-free RL algorithms that can perform waypoint tracking as well as stabilization from harsh initialization. The authors trained the policy in the quadrotor simulator and then transferred it to a real drone without any adaptation process. Although RL has made it possible to avoid complex mathematical modeling and the fine-tuning process, these methodologies did not consider the changing dynamics of a quadrotor. Therefore, the control performance could not be maintained when the actual dynamics were changed. Lambert *et al*. [4] used model-based RL to generate a low-level control policy optimized for an individual quadrotor. Here, prior dynamics knowledge was not required because the dynamics transition model was directly trained from the demonstration data. However, this method needed actual quadrotor flying data, which requires either a human demonstration or an alternative low-level controller. Furthermore, the trained policy only focused on controlling one quadrotor that was used when acquiring the data, not the quadrotors with different dynamics.

Molchanov *et al*. [2] proposed a method for obtaining a single policy that can robustly operate in different quadrotors by applying the dynamics randomization technique. A PPO algorithm consisting of only the feed-forward layer was trained in a self-made quadrotor simulator that did not consider air drag effect. During the training phase, the trajectory of every episode was generated in the environment with randomly sampled five factors: mass, size, control frequency, thrust-to-weight ratio, and thrust-to-torque ratio. However, it was unclear whether this controller had any performance advantage compared to those that use dynamics information. Fei *et al*. [38] proposed a novel method to train a robust controller for drones to recover from unpredictable physical and cyber attacks. The physical attacks contained situations where the actuator signals and the sensor values were eighter missed or replaced with the wrong values. The aim was to obtain a more robust policy by applying a random attack in the training process instead of randomizing the dynamics. However, all of these controllers focused on the robustness of performance, regardless of changes in dynamics.

Peng *et al*. [6] demonstrated that using RNNs can help the robot controller respond

to unknown dynamics. The dynamics of robot arms were randomized in the simulation and an actor-critic model consisting of long short-term memory (LSTM) was trained using the recurrent deterministic policy gradient (RDPG) [5] algorithm. Fris *et al*. [39] applied RDPG to train a controller that can land a quadrotor on a slope, even when the mass and moment-of-inertia of the quadrotor are changed. However, a 2-D quadrotor simulator that was far from an actual quadrotor was used, and this method could not respond to changes in dynamics that are critical to the quadrotor, such as motor performance and propeller status.

In contrast to the previously mentioned research, we first use an RNN network that can extract dynamics information on a quadrotor. Then we use a 3-D quadrotor simulator with air drag effects that make it more likely to apply to real quadrotors. Finally, we improve learning stability through an auxiliary training process for the RNN module instead of an end-to-end approach to directly include RNN in the policy module proposed in [6].

Chapter 3

Memory-based robot controller

3.1 Introduction

In this section, the optimal structure for extracting dynamics model information using RNN is determined using a simpler robot environment than the quadrotor. The method of extracting unknown dynamics model information using RL is to add an RNN to both the actor and the critic [6]. At first, the structure that uses RNN only for actors without using RNN for the critic and various types of RNN are considered in Chapter 3.2. Finally, the most suitable type of RNN and the structure of actor-critic are experimentally shown in Chapter 3.3.

The simulation robot environment named Inverted-Pendulum [40] is used, as shown in Fig. 3.1. The position of a uniform rod with one end fixed to the motor is controlled by the torque of the motor. The dynamics model of the Inverted-Pendulum can be described as follows:

$$\frac{1}{12}mL^2\ddot{\theta} = mg\sin\theta + a\tau_{max}, \tag{3.1}$$

where a is the control action in the range [-1,1]. The goal of the environment is setting up the rod upside down as soon as possible without making the rotation speed and acceleration too large. Therefore, the reward function and the objective function can

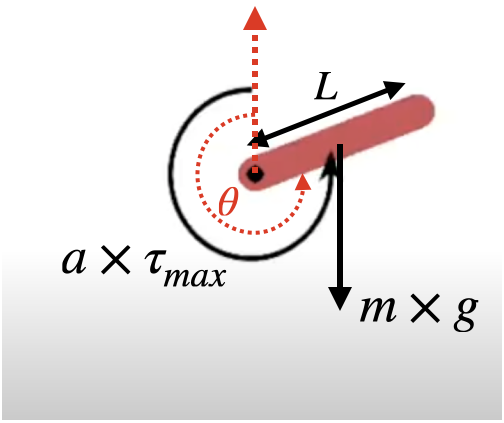


Figure 3.1: Inverted-Pendulum environment.

Parameter	Range	Original value
mass of the pole m [kg]	[0.33,3]	1
length of the pole L [m]	[0.33,3]	1
max torque of the motor τ_{max} [N· m]	[0.66,6]	2

Table 3.1: Dynamics parameters and their randomization range for Inverted-Pendulum environment.

be formulated as follows:

$$r_t = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001\ddot{\theta}^2), \tag{3.2}$$

$$\underset{\pi}{\text{maximize}}\, \mathbb{E}_{s_0,a_0,s_1,\ldots}[\sum_{t=0}^{\infty} \gamma^t r_t], \tag{3.3}$$

where s is the state which consists of $\{\theta, \dot{\theta}\}$.

In order to change the dynamics model, three parameters affecting robot control are randomized: mass of the pole, length of the pole, and maximum torque of the joint motor. Each parameter is uniformly sampled from the range displayed in Table 3.1. Since the state includes only the current angle and rotation speed, additional informa-

tion must be obtained through the state–action sequence to respond to changes in the dynamics model.

3.2 Approach

In the conventional dynamic randomization method [6], which we refer to as RNN-full herein, a RNN for extracting dynamics information was used for both actor and critic, as shown in Fig. 3.2. Although the reason for using RNNs was to extract information about dynamics from the state–action history, this can cause a reduction in learning stability and learning speed [7]. In the actor-critic model, the actor needs to extract information about the dynamics model through the RNNs because the dynamics model is inaccessible directly in the test phase. However, Critic, which is used only for approximation of the value function to update the Actor, is used only in the training phase and not in the test phase. In the training phase, the ground truth value for the dynamics parameter μ of the simulator can be used to train the critic, because all information about dynamics is fully accessible. The dynamics information that can be obtained using RNN cannot be more than the true dynamics model. Therefore, there is no loss of information even if the RNN structure of the critic is excluded. In Fig. 3.3, we employed the RNNpolicy structure, which uses a feed-forward network and ground truth dynamics parameters for the critic and only uses RNNs for the actor. The following experiment in chapter 3.3 shows that the proposed structure improves the training performance and learning speed.

The performance and learning stability can be different depending on the type of RNN even with the same actor-critic structure. Different from [6] which uses only one RNN types, three types of RNNs are considered: vanilla Recurrent Neural Networks (RNN) [41], Long Short-Term Memory (LSTM) [42], and Gated Recurrent Unit (GRU) [43]. Vanilla RNN can be implement with the lightest model size and is effective when information can be extracted with only short term memory. LSTM is

effective when considering long-term memory together, but it has a large model size and amount of computation. In the case of GRU, the size of the model is reduced compared to LSTM while considering long-term memory. Chapter 3.3 shows which type of RNN can be used for stable and high-performance learning.

3.3 Experimantal results

To compare the performance according to the RNN type and structure, two metrics were used: the reward sum and the success rate. Evaluation is performed on 100 random dynamics models. The reward sum is the average reward summation of each dynamics model episode, and the success rate is the average of the success or failure of the episode. The success condition is judged if the angle between the pendulum rod and the vertical axis is maintained at less than 5 degrees for 2 seconds within the time limit. If the episode is judged as succeeds, it is immediately finished even if it does not reach the time limit of 7.5 s. It is noteworthy that the return increases when the length of the episode is shortened because the strictly negative reward function is used. In each evaluation step, 100 random dynamics models were tested to obtain the return and the success rate of the trained policy.

The training episode length of the Inverted-Pendulum is set to 7.5 s, and the control frequency is set to 20Hz. Therefore each episode is discretized for a total of 150 timesteps. A total of 20000 episodes are used for training, equivalent to data for 20000 random dynamics models. The SAC algorithm is used to train the controller, and the hyper-parameter details of the RNNpolicy actor-critic networks are shown in Table 3.2.

A total of seven types of controllers are compared experimentally:

- FF, which only uses feed-forward networks for both the actor and the critic, same as the original SAC [22].
- RNNfull, which uses vanilla RNN for both the actor and the critic.

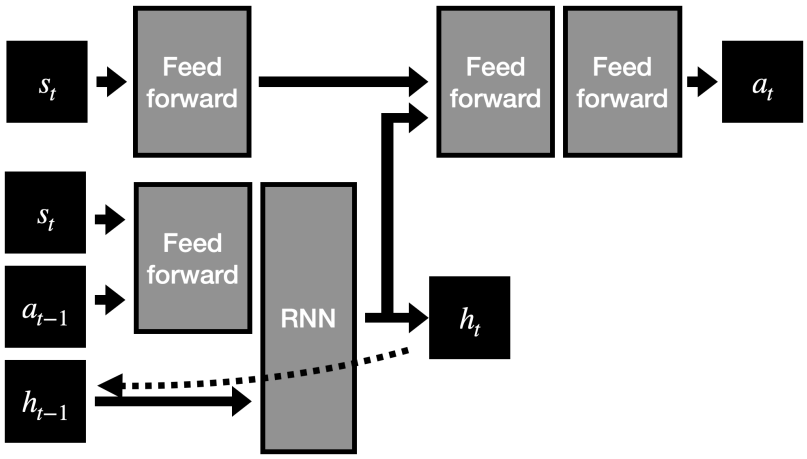
Table 3.2: RNNpolicy hyperparameters

Hyperparameter	Value	
	Actor	Critic
number of hidden layers	5 (4 linear, 1 RNN)	3
number of hidden units	64	64
Activation function	ReLU	ReLU
Last Activation function	Tanh	None
learning rate	$3e-5$	$3e-4$

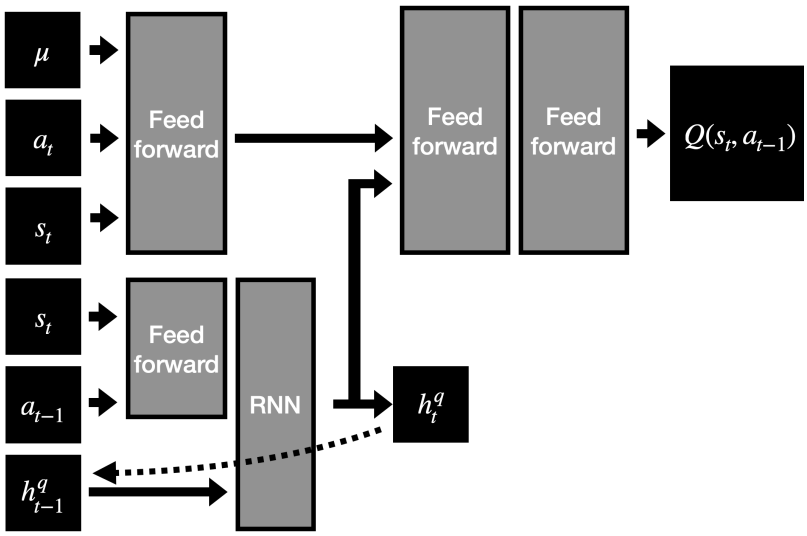
- LSTMfull, which uses LSTM for both the actor and the critic [6].
- GRUfull, which uses GRU for both the actor and the critic.
- RNNpolicy, which uses vanilla RNN for the actor and uses feed-forward networks for the critic.
- LSTMpolicy, which uses LSTM for the actor and uses feed-forward networks for the critic.
- GRUpolicy, which uses GRU for the actor and uses feed-forward networks for the critic.

Figure 3.4 displays the learning curve of average return and success rate for the Inverted-Pendulum experiment. Firstly, RNNpolicy, LSTMpolicy, and GRUpolicy show higher average returns and success rates than other methods. In other words, the proposed method in which the RNN structure is removed from the critic shows more effective performance than the method of extracting information on dynamics in an end-to-end manner using RNN in both the actor and the critic. Secondly, the low performance of FF shows that it is difficult to cope with unknown dynamics changes without using RNN. Lastly, LSTM or GRU showed better performance than vanilla RNN comparing the performance according to the type of RNN with the same structures. The vanilla

RNN has a gradient vanishing effect for long sequences. In the case of the Inverted-Pendulum task, the performance of vanilla RNN is inevitably degraded because of a long sequence consisting of 150 timesteps. Looking at both the average return and the success rate as a metric, there was no significant difference in performance between GRU and LSTM. However, similar performance can be achieved with fewer episodes when using GRU. Moreover, considering that the number of parameters of GRU is 75% of LSTM, using GRU can achieve better performance with a smaller model.

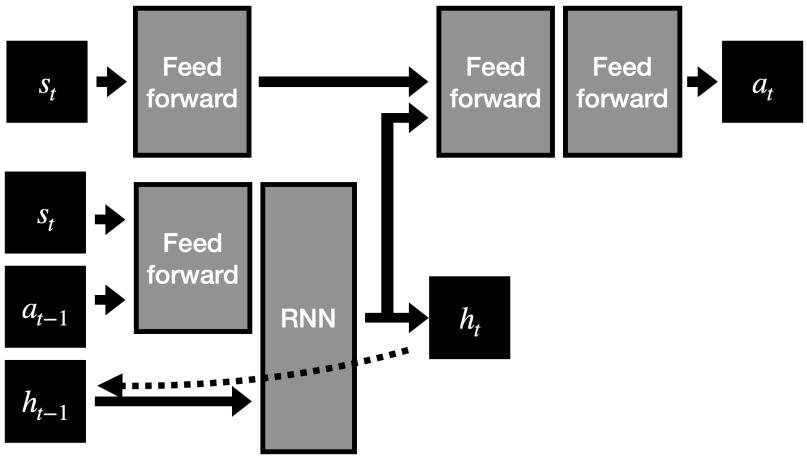


(a) Actor

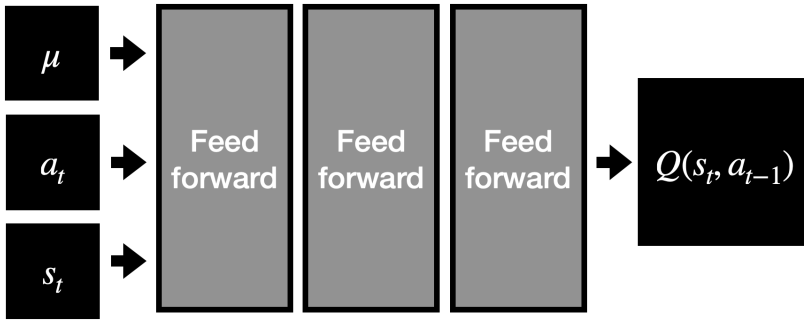


(b) Critic

Figure 3.2: Actor-critic structure of RNNfull. A RNN is included in both the actor and the critic for extracting dynamics model information from the state–action sequence.

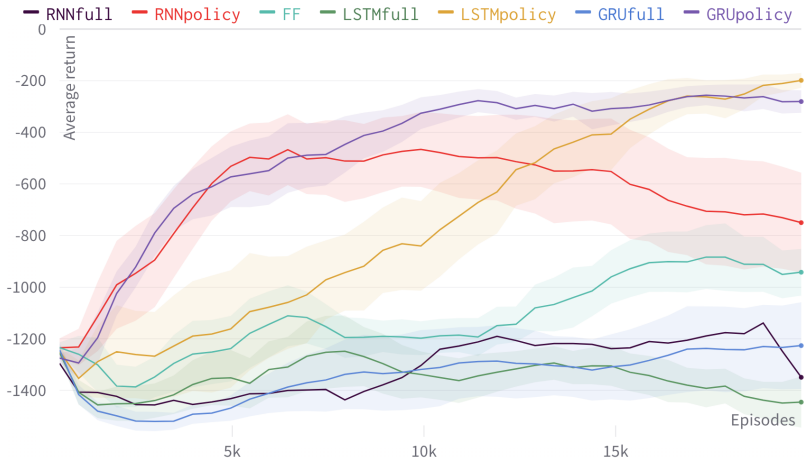


(a) Actor

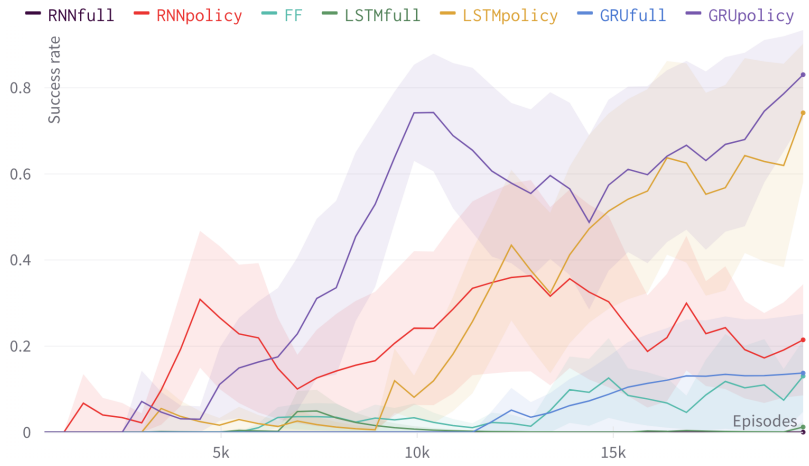


(b) Critic

Figure 3.3: Actor-critic structure of RNNpolicy. A RNN is included only in the actor for extracting dynamics model information from the state–action sequence. The critic receive sufficient information about the dynamics model by using the ground truth of the dynamics model μ from the simulator as an input.



(a)



(b)

Figure 3.4: (a) Average return learning curve of the Inverted-Pendulum experiment. (b) Success rate learning curve of the Inverted-Pendulum experiment. Return and success rate are measured by averaging the result obtained with 100 random dynamics models in each evaluation. The success is defined as the angle between the pendulum rod and the vertical axis is maintained at less than 5 degrees for 2 s.

Chapter 4

Memory-based quadrotor controller

4.1 Introduction

In this section, the RNN-based quadrotor controller is presented that can respond to changes in dynamics. At first, the problem that we solve is defined, and the simulator setting is described in Chapter 4.2. Also, a novel actor-critic structure specified for quadrotor controller learning is proposed. Finally, the experiment in Chapter 4.3 shows the proposed method outperforms existing methods in the stabilizing quadrotor task.

4.2 Approach

4.2.1 Problem definition

We focused on training a low-level controller that produces a raw motor signal from the current state of the quadrotor as well as the past state and motor signals. The input state $s_t \in \mathbb{R}^{18}$ consists of relative position from the goal $o_t \in \mathbb{R}^3$, all elements of rotation matrix $R_t \in \mathbb{R}^9$, linear velocity $v_t \in \mathbb{R}^3$, and angular velocity $\omega_t \in \mathbb{R}^3$. It should be noted that all positions used for input state used relative positions, regardless of the global origin. The reason why we used the rotation matrix $R_t \in SO(3)$ instead of using quaternions or Euler angles was that this can represent all altitudes without any

discontinuity. The action $a = \{a^1, a^2, a^3, a^4\} \in \mathbb{R}^4$ is the PWM signal of the four motors, which should be given as a discrete integer between range $[0, 2^{16} - 1]^4$. We scaled the action to a real value between $[-1, 1]$ and regarded it as a continuous action space.

The objective of this study was to train a low-level controller policy that minimizes the distance from the current position to the goal and the angular velocity in the shortest time according to the dynamics of the quadrotor, represented as follows:

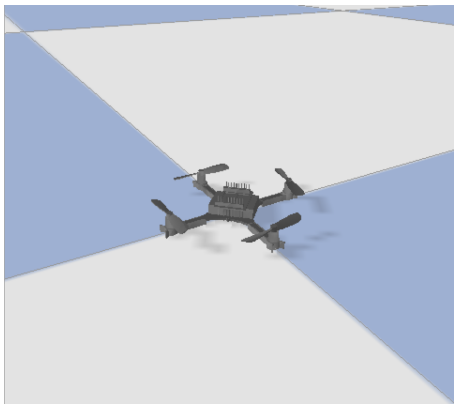
$$\begin{aligned} \pi^*(a|s,h) &= \underset{\pi}{\operatorname{argmax}}\, J^\mu(\pi), \quad \forall \mu \in M, \\ J^\mu(\pi) &= \mathbb{E}_{s_0,a_0,s_1,\ldots} \sum_{t=0}^{\infty} -f(\|o_t\|, \|\omega_t\|), \end{aligned} \tag{4.1}$$

where h is the embedding of past state–action sequence, μ is the dynamics parameter of the environment, M is the dynamics parameter space, and f is the monotonic increasing function. It should be noteworthy that the function f has the same meaning as the negative of the reward function r, which can be engineered with various function forms, such as $r(s_t) = -f(\|o_t\|, \|\omega_t\|)$.

4.2.2 Simulator setting and dynamics randomization

Gym-pybullet-drones simulator [44] was used to reproduce situations where the state transition model was changed, which can occur in the quadrotor, and train the RL controller. This environment has three advantages: 1) the proposed code presented in Chapter 3 can be used without modification because the input and output structures are the same as Inverted-Pendulum; 2) the reality gap is minimized because the complex physical factors of quadrotors, such as air drag, down washing, and ground effects; and 3) The dynamics model of the real model called Crazyflie [45] is implemented.

Six types of dynamics parameters were randomized in this simulator: mass, location of the center of mass, moment-of-inertia, thrust-to-force coefficient k_f, thrust-to-momentum coefficient k_m, and motor delay time constant T. The following parameters were those that could be changed during the flight or could have a different value, even



(a) Gym-pybullet-drone simulation

(b) Crazyflie

Figure 4.1: (a) Gym-pybullet-drone simulator environment [44]. (b) Crazyflie, a real drone model implemented in Gym-pybullet-drone [45].

in the same quadrotor product. We independently randomized the XY coordinates of the center of mass, XYZ coordinates of the moment of inertia, and the k_f and k_m parameters for all four motors. Therefore, 15 parameters were randomized and consisted of values representing $\mu \in \mathbb{R}^{15}$. All the parameters, except for motor delay T, are already modeled in the gym-pybullet-drone simulator, so the values could be easily and randomly modified. In the case of mass and moment-of-inertia, we changed the values inside the URDF file, and forward kinematics calculation was performed through the pybullet physical engine. Terms k_f and k_m represent the thrust and rotational force generated ratio proportional to the motor speed, respectively. The final action-to-force and action-to-momentum values were modeled as follows:

$$f^i = k_f^i \times (p^i)^2,\ \tau^i = k_m^i \times (p^i)^2, \tag{4.2}$$

where p^i is the RPM of each motor and i is the index of each motor $i = 1, 2, 3, 4$. According to [2], motor delays can occur in real-world motors because the motor signal cannot be immediately reflected in the RPM, which is an important factor that can affect performance in real-world applications. The motor delay time constant T was

Table 4.1: The randomization range of dynamics parameters and initial state in Gym-pybullet-drones environment

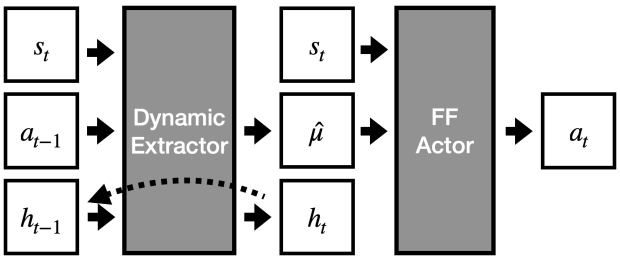
Parameter	**Range** ($\beta = 0.3$)
Mass (m)	[1-β,1+β] $\times$ $m^{original}$
Center of mass (x_{cm},y_{cm})	[-β,β] $\times$ body length of the drone
Moment-of-inertia (I'_{xx},I'_{yy},I'_{zz})	[1-β,1+β] $\times$ $I^{original}$
$k_f = \{k_f^i\}, i = 1, 2, 3, 4$	[1-β,1+β] $\times$ $k_f^{original}$
$k_m = \{k_m^i\}, i = 1, 2, 3, 4$	[1-β,1+β] $\times$ $k_m^{original}$
T ($\sim$motor delay time constant)	[1-β,1+β] $\times$ 0.15
Initial state	**Range**
Linear velocity [m/s]	$\sim [-1, 1]^3$
Angular velocity [rad/s]	$\sim [-\pi, \pi]^3$
Rotational matrix	$\sim SO(3)$
goal position [m]	$\sim [-1, 1]^3$

used as follows:

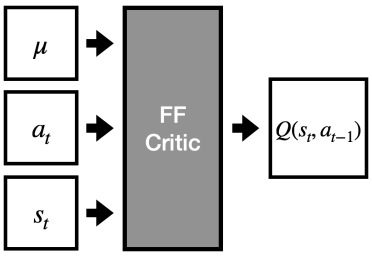
$$a_{t+1} = \frac{4dt}{T}(a_{t+1} - a_t) + a_t, \tag{4.3}$$

where dt is the control time interval, which was set to 10 ms in our experiment. Table 4.1 shows the randomization range for each parameter.

To train the policy that can acquire information about dynamics parameters and stabilize the quadrotor in harsh configurations, we randomly sampled initial attitude, linear velocity, and angular velocity at the simulation reset step. In the case of attitude, random sampling was performed in the entire $SO(3)$. The initial position was set to a point far from the ground to avoid a situation where the velocity and angular velocity were zero due to being hit by the ground. The target positions were randomly sampled from a cube having the initial position as the center. The range of the initialized state is demonstrated in Table 4.1. In addition, state random noise was added to reflect the



(a) Feed-forward actor with dynamic extractor



(b) Feed-forward critic

Figure 4.2: (a) The policy network of RNNparam consists of the dynamic extractor and the feed-forward actor. (b) The feed-forward critic of RNNparam uses the ground truth of dynamic parameters.

real-world environments.

4.2.3 Actor-critic model

Fig. 4.2 displays the proposed actor-critic network structure. In a study that focused on training a policy to handle the unknown dynamics in the robot arm [6], the RDPG algorithm was used, which combined LSTM and DDPG. Instead, we used a soft actor-critic (SAC) [22] algorithm combined with a gated recurrent unit (GRU) [43], which is a type of RNN. The reason for using a GRU was that it has higher learning stability than vanilla RNN and has a smaller model size than LSTM, as verified in Chapter 3. It is worth noting that GRU can be simply replaced with various recurrent neural network cells including vanilla RNN or LSTM; hence the general name RNN is used herein.

Recurrent networks for extracting dynamics information are used for both actor

and critic in the RNNfull structure, which is the conventional dynamic randomization method [6]. However, the RNN structure inevitably has the problem of learning instability and a decrease in learning speed. In Chapter 3, the effectiveness of not using the RNN structure in the critic was confirmed because the ground truth about the dynamics model information can be obtained when using the simulation. Therefore, the RNNpolicy structure was proposed that consists of the actor including recurrent neural networks, and the critic using feed-forward neural networks and ground truth of dynamics model parameters, as shown in Fig. 4.2(b). Although the RNNpolicy structure successfully learned the controller that can respond to changes in the dynamics model in the inverted pendulum task, the problem of learning instability has still been shown in the drone environment. First of all, it is difficult to obtain good trajectory data through random search in the early stage of learning because the drone loses stability and falls when a random action is applied. Secondly, the control frequency of the quadrotor is at least 100 Hz which is very high compared to the pendulum robot. The higher control frequency of the quadrotor increases the length of the state-action sequence and worsens the learning instability of the RNN. To solve instability issues, the RNNparam structure is proposed that can use the information extracted from the state–action sequence without using RNN for the actor-critic structure in the RL loop learning phase.

As shown in Fig. 4.2, the RNNparam structure consists of an actor and a critic composed of feed-forward networks, and a dynamic extractor using RNN. The feed-forward actor and the feed-forward critic were trained through a SAC algorithm. The difference from the existing controller using feed-forward networks which shows robust performance to dynamics changes [2] is that the actor uses a dynamics parameter. Since the actor uses the ground truth of the dynamics parameter μ as a given input in the training phase, the trained actor would produce actions by reflecting changes in dynamics information. The problem of learning instability with RNNs was also solved because the actor and critic only consist of feed-forward networks. However, the dy-

namics ground truth parameters are inaccessible in the execution phase in the real world. Therefore, the dynamics extractor module that predicts dynamics parameters directly from the state–action sequence is required. The dynamic extractor comprises a combination of a linear layer for embedding, a GRU layer, and a linear layer for predicting values. Training of the dynamic extractor is conducted in a supervised-learning manner that predicts the dynamic parameters from the state–action sequence through RNNs separately from the RL training process. The details of the update process of the RNNparam structure are described in Algorithm 1.

First, the initial state s_0 and dynamic parameters μ are randomized in the range decided in Table 4.1 for the beginning of every episode (line 5). An episode of length T is rolled out by interacting with the environment, which has the dynamics model μ (lines 6–8). It should be noted that the action is created by using the feed-forward actor with the ground truth dynamics parameters μ. This helps to accumulate high-quality data in the replay buffer compared to using a less-trained RNN-structured actor. The created trajectory is stored in the replay buffer $\mathcal{B}$ (line 9).

To update the networks, a minibatch consisting of $|B|$ number of full episodes is sampled from the replay buffer (line 9) and loss values for the actor, the critics, and the dynamics extractor are initialized as zero (line 11). We then uniformly sample a single timestep t for each episode in the minibatch (line 13). The reason for the additional random sampling of single timestep data instead of using the entire episodes is to maintain the i.i.d condition for training the actor and the critic networks, which are comprised of the feed-forward neural networks. It is noteworthy that every episode contains the ground truth of dynamics parameters μ that were randomized for each episode in the simulator. The SAC algorithm is applied to update the actor and the critic, considering the dynamics information μ (lines 14–18 and 25–27). The dynamics extractor D_ψ, which is composed of the RNN network, predicts the dynamics parameter $\hat{\mu}$ from every sequence of state–action pairs in the episode (lines 20–22). The dynamics parameters $\hat{\mu}_{t'}$ of every timestep t' are predicted based on the hidden output of RNN $h_{t'-1}$ and

an input state–action pair $(s_{t'}, a_{t'-1})$. The hidden output of RNN $h_{t'-1}$ is an output latent vector for the given previous state–action sequences $(s_0, a_{-1}, \ldots, s_{t'-1}, a_{t'-2})$. The dynamic extractor is updated to minimize the mean square error between all predictions $\{\hat{\mu}_{t'}\}$ and the ground truth parameter μ (lines 23 and 28). By using the trained actor and dynamics extractor module, a low-level controller produces an action as follows:

$$a_t = \pi_\theta(s_t, D_\psi(h_{-1}, s_0, a_{-1}, \ldots, s_t, a_{-1})), \tag{4.4}$$

which predicts the dynamics parameters by dynamics extractors and uses them as a given input to produce an action.

4.3 Experimental results

In this section, we experimentally demonstrate that the proposed controller achieves good performance, even for quadrotor dynamics that are changed, compared to existing methods and other RNN actor-critic structures. We compared the three RNN-based controllers and three conventional controllers experimentally:

- PID controller, which needed to fine-tune 18 PID gain values in advance [44].
- FF-rand, which only used feed-forward networks with dynamics randomization [2].
- FF-norand, which only used feed-forward networks without dynamics randomization.
- RNNfull, which used RNNs for both the actor and the critic and trained in an end-to-end manner [6].
- RNNpolicy, which only used RNNs for the actor and trained in an end-to-end manner.

Table 4.2: RNNparam hyperparameters

Hyperparameter	Value		
	Actor	Critic	Dynamic extractor
number of hidden layers	5	5	3 (2 linear, 1 GRU)
number of hidden units	64	128	64
Activation function	ReLU	ReLU	ReLU
Last Activation function	Tanh	None	Tanh
learning rate	10^{-4}	10^{-4}	10^{-4}

- RNNparam, which is the proposed method that trains an auxiliary dynamics extraction module consisting of RNNs.

All learning-based controllers were basically trained using the SAC algorithm. To satisfy the possibility of use in a real-world quadrotor, the following hardware constraints of the real model of the quadrotor used in the gym-pybullet-drone were satisfied: operating $\leq$10 ms at 168 MHz Cortex-M4 MCU, and limiting model size limit of $\leq$192 KB. The model size limit should only be respected by the actor because the critic is only used during the training process. A larger critic model was used to sufficiently reflect the complexity of the environment. This reason that a bigger critic model can be used is based on the study of Mysore *et al*. [46], where even a larger critic than the actor did not affect the final performance. Therefore, we constructed and trained the model using the hyperparameters, as shown in Table 4.2.

We used 20,000 trajectories with a step length of 800, which is 8 s in the simulator running at 100 Hz. The same following reward function was used to ensure a fair comparison:

$$r(s_t) = -(\|o_t\|^2 + 0.5\|\omega_t^{yaw}\|^2), \tag{4.5}$$

where $\|o_t\|$ is the Euclidean distance from the current position to the goal position, and ω_t^{yaw} is the yaw rate. The quadrotor had to change the direction of thrust by changing

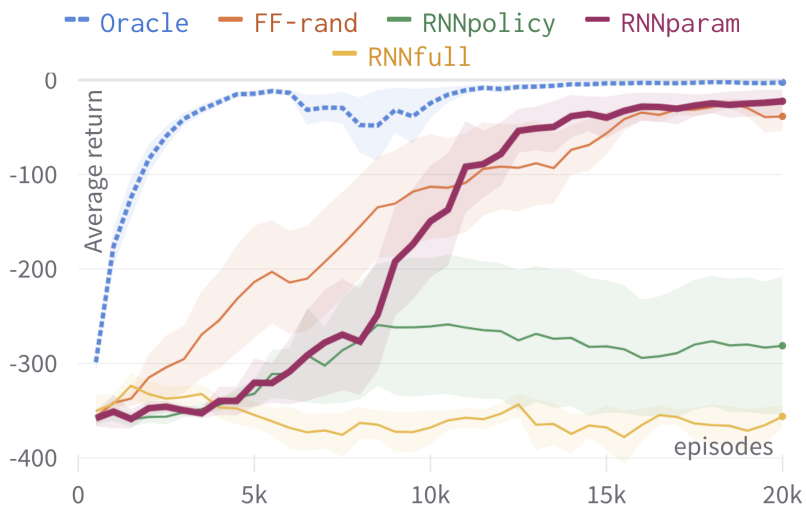


Figure 4.3: Learning curve of five different actor-critic structures runs with five different seeds. The Oracle is the learning curve of FF-norand because the training was not interfered with by dynamics randomization.

the roll and pitch to control the XY position. In other words, there was a trade-off relationship in which the roll and pitch rates had to be increased to reduce the positional errors. We judged that this trade-off relationship would make learning difficult. Hence, we designed the reward function to minimize position error more efficiently by only considering only the yaw rate at angular velocity.

We trained five policies for each method with varying seeds, and the learning curve of the average return of evaluation is shown in Fig 4.3. It should be noted that FF-norand is the same as the oracle learning curve because it is the only one learned in the absence of dynamics randomization. It will be shown in later experiment results that the performance of FF-norand is worse in the dynamics randomization environment, even if the average return is high in the training process.

It can be observed that the final return of RNNparam was the closest to the oracle compared to the other methods. In the case of RNN-included structures, it was evident that the more RNN was not used in the RL learning process, the better the learning became. First, it was more stable to only use the RNN structure for the actor (RNNpolicy)

than to use the RNN structure for both the actor and critic (RNNfull). Moreover, the RNNpolicy still exhibited a less average return and high variance, compared to the proposed RNNparam method. We confirmed that the end-to-end learning method using the RNN structure for both actor and critic was not suitable for unstable quadrotor environments. This is because it is difficult to obtain data close to stable flying by random exploration. In the case of FF-rand, the average return increased quickly during the early stage of the training. However, the final return of FF-rand was lower than that of RNNparam.

4.3.1 Stabilization experiment

We conducted the stabilizing experiment using the controller with the highest average return among them trained by various seeds. However, RNNfull was excluded from the experiment because it was not even trained, as shown in Fig. 4.3. In the stabilizing experiment, the aim was to control the quadrotor to a random goal position from the randomized initial state in 8 s, as in Table 4.1. The average position error to the goal e_p and the average magnitude of the yaw rate $\|\omega_{\text{yaw}}\|$ were measured as a metric to represent control and stabilizing performance. Each controller was tested in an environment with different degrees of dynamic randomization on 100 random seeds. Table 4.3 displays the result of the stabilizing experiment.

It can be deduced from the large positional error that the PID controller could not recover a quadrotor from harsh initialization, even when the dynamics were not randomized. However, the learning-based controllers exhibited some response, regardless of the dynamics. The FF-norand controller achieved the best performance when the dynamics were fixed. However, it did not respond appropriately to changes in dynamics (*i.e.,* $\beta \neq 0$). For the FF-rand controller, the larger the value of β, the larger the change in performance degradation compared to the RNNparam controller. In other words, the RNNparam controller achieved superior performance to the FF-rand when responding to situations in which the dynamics changed significantly. Moreover, the

Table 4.3: Experiment for stabilizing from random initial state

dynamics randomize range ($1\pm\beta$)		PID	FNN-norand	FNN-rand	RNNpolicy	**RNNparam**
β = 0.0	e_p	73.52	**0.75**	0.91	6.12	0.90
	$\|\omega_{yaw}\|$	0.95	**0.01**	0.21	10.73	6.71
β = 0.1	e_p	84.30	5.83	1.55	10.09	**1.04**
	$\|\omega_{yaw}\|$	9.02	5.37	**0.70**	17.45	2.40
β = 0.2	e_p	87.88	28.84	5.55	20.89	**3.22**
	$\|\omega_{yaw}\|$	14.21	21.79	**1.25**	19.80	2.08
β = 0.3	e_p	86.65	58.56	23.54	36.31	**11.06**
	$\|\omega_{yaw}\|$	21.97	41.52	11.77	18.27	**6.89**

RNNparam controller exhibited much smaller positional errors and angular velocity compared to the RNNpolicy controller using RNN in the end-to-end manner.

In order to verify the performance of the proposed controller quantitatively, the control trajectory is examined in certain scenarios that the quadrotor could face. Fig. 4.4 displays the controlled trajectory of the quadrotor to a target position when the quadrotor was flipped in 6 scenarios: (a) motor performance degradation, (b) center of gravity shift, (c) motor delay change, (d) momentum-of-inertia change, (e) mass change (f) nothing change.

Fig. 4.4(a) shows the motor performance degradation scenario that was implemented by reducing the values of k_f and k_m by 20%, which is equivalent to generating less thrust and torque. Fig. 4.4(b) displays the scenario when the center of gravity is shifted by 10% of the total body length from the geometric center. The PID, FF-norand, and RNNpolicy controllers demonstrated that they could not recover the quadrotor from the flipped state in these two cases. Although the FF-rand controller success to stabilize from the flipped status and control the quadrotor to the target po-

sition, it took a relatively long time to stabilize and reach the target position compared to the RNNparam controller. Using the proposed RNNparam method, the quadrotor quickly recovered to a stable state and the quadrotor was controlled in response to the degradation of one motor.

Fig. 4.4(c-f) shows the results of the situation respectively: when the motor delay coefficient decreased by 20%, when the momentum-of-inertia in the x direction decreased by 20%, when the mass decreased by 20%, and when nothing changed. As well as the previous motor degradation and the center of gravity shift scenarios, the RNNparam controller generates an efficient trajectory that reaches the target position in a shorter time than the FF-norand controller. Although the RNNpolicy controller cannot still make the quadrotor reach the target position, it shows possible to recover from the flipped status. It should be noted in the four scenarios that the FF-norand controller generates the most efficient control trajectory.

The qualitative evaluation results confirm that the performance increase using RNN for the learning-based controller is remarkable in terms of motor performance degradation and center of gravity shift. The change in the performance of a single motor or the location of the center of gravity deviating from the center means a situation in which the symmetric assumed by the quadrotor dynamics model is broken. In the case of mass change, momentum-of-inertia change, and motor delay coefficient change scenarios, the physical quantities are changed while maintaining the symmetricity of the dynamics model. Therefore, the FF-norand controller, which has the advantage of operating robustly, produces a good performance. However, using a controller using an RNN structure shows better performance when the dynamics model changes asymmetrically.

Algorithm 1 SAC RNNparam algorithm

Initialize actor $\pi_\theta(s_t, \mu)$, critics $Q_{\phi_1}(s_t, a_t), Q_{\phi_2}(s_t, a_t)$, dynamic extractor $D_\psi(s_t, a_{t-1}, h_{t-1})$ with parameters $\theta, \phi_1, \phi_2, \psi$

Initialize target networks $\bar{Q}_{\bar{\phi}_1}, \bar{Q}_{\bar{\phi}_2}$ with parameters $\bar{\phi}_1 \leftarrow \phi_1, \bar{\phi}_2 \leftarrow \phi_2$

Initialize empty replay buffer $\mathcal{B}$

for episode=1:E **do**

  Randomize dynamics parameter μ, initial state s_0

  for t=0:T-1 **do**

    $s_{t+1} \sim p_\mu(\cdot|s_t, a_t)$ where $a_t \leftarrow \pi(s_t, \mu)$

  end for

  $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_0, a_0, r(s_0, a_0), \ldots, s_T, \mu)\}$

  Sample a minibatch of $|B|$ episodes

$$\{(s_0^i, a_0^i, r_0^i, \ldots, s_T^i, \mu^i)\}_{i=1,\ldots,|B|} \sim \mathcal{B}$$

  Initialize losses $L_\pi = L_Q = L_D = 0$

  for i=b:$|B|$ **do**

    $t \sim$ Uniform$(0, , 1, \ldots, T-1)$

    $\hat{a}_t^i \leftarrow \pi_\theta(s_t^i, \mu^i)$

    $\hat{a}_{t+1}^i \leftarrow \pi_\theta(s_{t+1}^i, \mu^i)$

    $q_t \leftarrow r_t +$

$$\gamma(\min_{i=1,2} \bar{Q}_{\bar{\phi}_i}(s_{t+1}, \hat{a}_{t+1}, \mu) - \alpha \log \pi_\theta(\hat{a}_{t+1}|s_{t+1}))$$

    $L_\pi \leftarrow L_\pi + (Q_{\phi_1}(s_t, \hat{a}_t, \mu) - \alpha \log \pi_\theta(\hat{a}_t|s_t))^2$

    $L_Q \leftarrow L_Q + (q_t - Q_{\phi_i}(s_t, a_t, \mu))^2$

    Initialize a_{-1} and h_{-1}

    for t'=0:T-1 **do**

      $\hat{\mu}_{t'}, h_{t'} \leftarrow D_\psi(h_{t'-1}, s_{t'}^i, a_{t'-1}^i)$

    end for

    $L_D \leftarrow L_D + \frac{1}{T} \sum_{t'=0}^{T-1} (\mu - \hat{\mu}_{t'})^2$

  end for

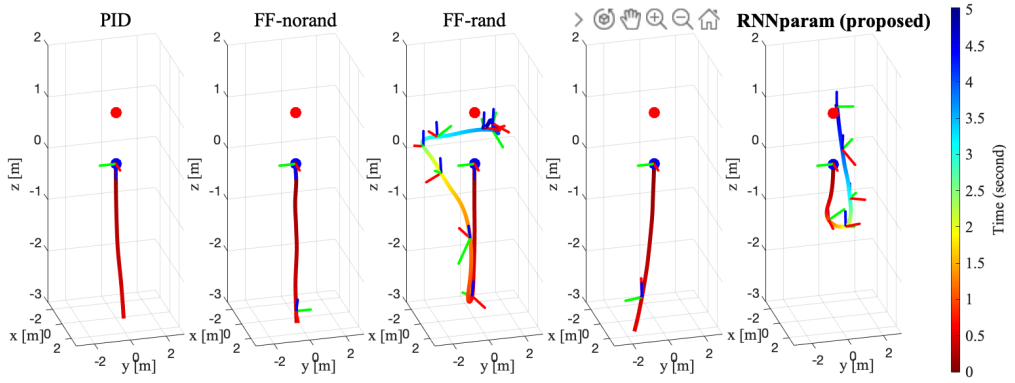
  $\phi_i \leftarrow \phi_i - \alpha_\phi \nabla_{\phi_i} \frac{1}{|B|} L_Q$

  $\bar{\phi}_i \leftarrow \tau \phi_i + (1-\tau)\bar{\phi}_i$

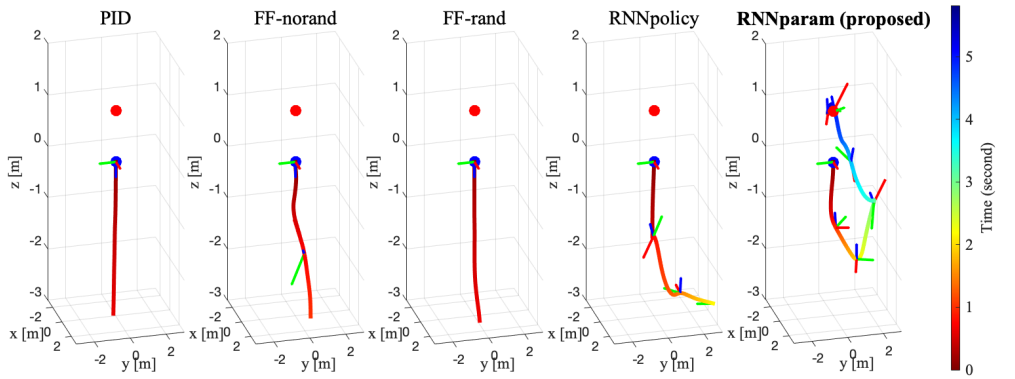
  $\theta \leftarrow \theta - \alpha_\theta \nabla_\theta \frac{1}{|B|} L_\pi$

  $\psi \leftarrow \psi - \alpha_\psi \nabla_\psi \frac{1}{|B|} L_D$

end for

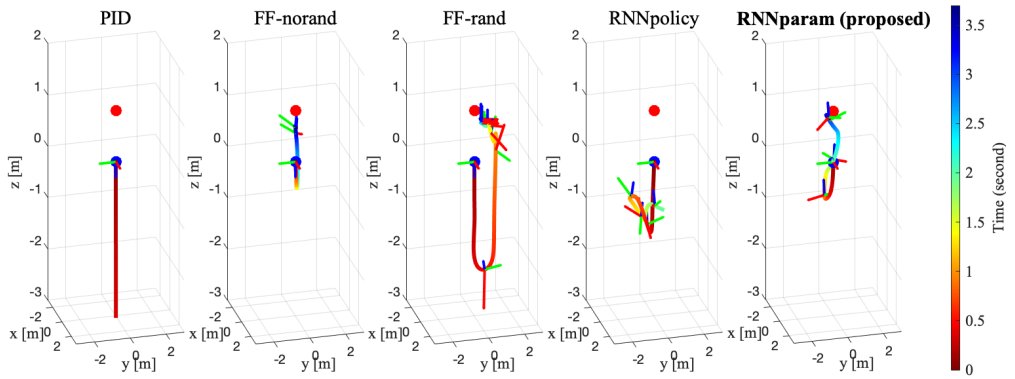


(a)

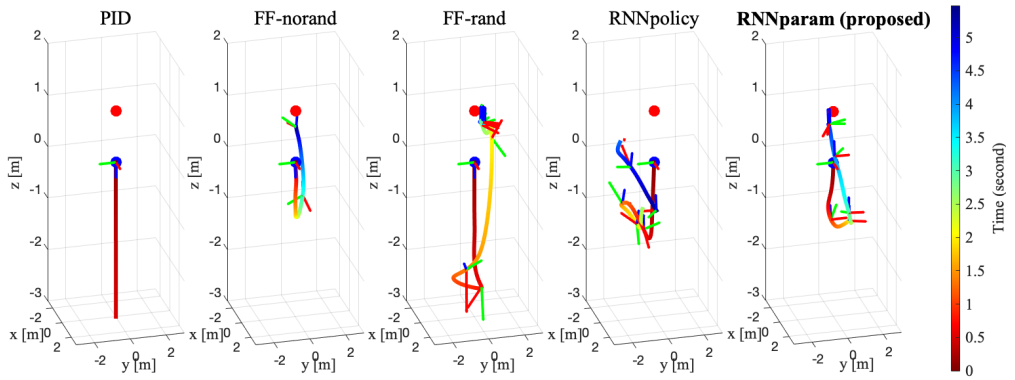


(b)

Figure 4.4: Trajectory of each controller that stabilized the quadrotor from a flipped state and controlled it to the target position (a red dot) in specific dynamics model change cases: (a) When the performance of one motor has reduced by 20%, (b) when the position of the center of gravity displaced by 10% of the total body length.

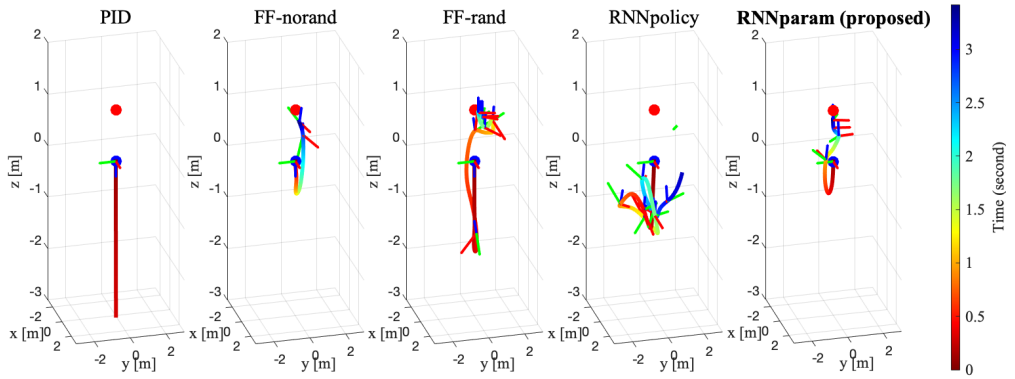


(c)

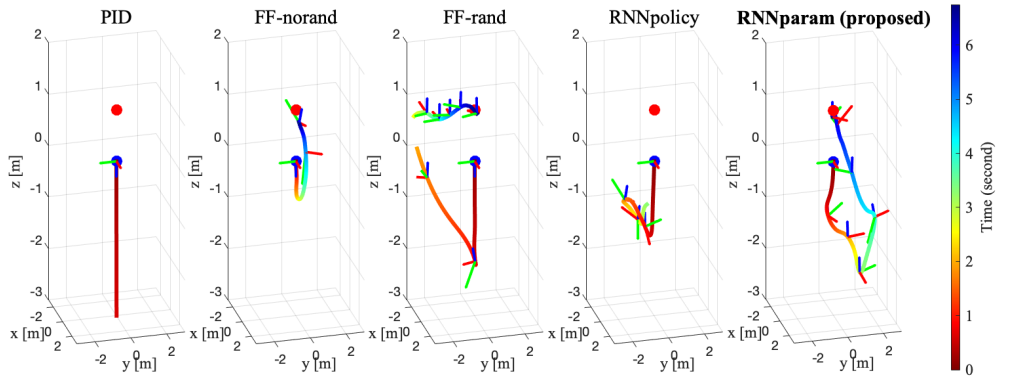


(d)

(c) when the motor delay coefficient has changed from 0.15 s to 0.12 s, (d) when the x-direction of the momentum-of-inertia has reduced by 20%.



(e)



(f)

(e) when the mass has reduced by 20%, (f) when nothing has changed.

Chapter 5

Conclusion

5.1 Conclusion

In this study, a new RNN-based actor-critic structure for learning a low-level controller of a quadrotor that can operate in response to unknown dynamics changes is proposed. Although there have been studies in which the RNNs implicitly extract dynamics information in an end-to-end method, we experimentally demonstrated that this method is unsuitable for unstable drones. Hence, a new possible structure is suggested. The dynamics in the simulator are randomly changed for every episode and the feed-forward network-based actor-critic model is trained using the SAC algorithm. Furthermore, dynamics information is extracted from the state–action sequence using the RNN-based dynamic extractor which is trained in a supervised learning manner. Using the proposed RNN-based method, the performance improvement especially for asymmetric changes such as motor performance degradation or center of gravity shift was experimentally verified. Through the proposed controller, it will be possible to control in response to situations where the dynamics change unexpectedly during flight, such as when the motors overheat or the propeller is damaged.

This study includes several limitations that require improvement in future research. Firstly, the size of the model was limited because of the hardware constraints to demon-

strate the possibilities of real-world applications. Different methods that perform better with limited model size should be explored, such as training with a larger model and then using distillation. Secondly, there were too many trade-off relationships in the quadrotor reward engineering process, so it could not be considered optimal engineering. Using goal-conditioned RL using binary goal-achieved rewards, the laborious reward engineering process could be omitted.

Bibliography

[1] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[2] A. Molchanov, T. Chen, W. Hönig, J. A. Preiss, N. Ayanian, and G. S. Sukhatme, "Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 59–66.

[3] B. P. Duisterhof, S. Krishnan, J. J. Cruz, C. R. Banbury, W. Fu, A. Faust, G. C. de Croon, and V. J. Reddi, "Tiny robot learning (tinyrl) for source seeking on a nano quadcopter," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7242–7248.

[4] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[5] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," *arXiv preprint arXiv:1512.04455*, 2015.

[6] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "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.

[7] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.

[8] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[12] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[13] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[14] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[15] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[16] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[17] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[20] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "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.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[24] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[25] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[26] J. Li and Y. Li, "Dynamic analysis and pid control for a quadrotor," in *2011 IEEE International Conference on Mechatronics and Automation*. IEEE, 2011, pp. 573–578.

[27] J. Ren, D.-X. Liu, K. Li, J. Liu, Y. Feng, and X. Lin, "Cascade pid controller for quadrotor," in *2016 IEEE International Conference on Information and Automation (ICIA)*. IEEE, 2016, pp. 120–124.

[28] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 2520–2525.

[29] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[30] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 773–11 780, 2014.

[31] M. Islam, M. Okasha, and M. Idres, "Dynamics and control of quadcopter using linear model predictive control approach," in *IOP Conference Series: Materials Science and Engineering*, vol. 270, no. 1. IOP Publishing, 2017, p. 012007.

[32] R. Benotsmane, A. Reda, and J. Vásárhelyi, "Model predictive control for autonomous quadrotor trajectory tracking," in *2022 23rd International Carpathian Control Conference (ICCC)*. IEEE, 2022, pp. 215–220.

[33] N. Mohajerin and S. L. Waslander, "Modular deep recurrent neural network: Application to quadrotors," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2014, pp. 1374–1379.

[34] ——, "Modelling a quadrotor vehicle using a modular deep recurrent neural network," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2015, pp. 376–381.

[35] U. Maqbool, T. Nomani, and H. Talat, "Neural network controller for attitude control of quadrotor," in *2019 Second International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT)*. IEEE, 2019, pp. 1–8.

[36] T.-T. Tran and C. Ha, "Self-tuning proportional double derivative-like neural network controller for a quadrotor," *International Journal of Aeronautical and Space Sciences*, vol. 19, no. 4, pp. 976–985, 2018.

[37] E. Khosravian and H. Maghsoudi, "Design of an intelligent controller for station keeping, attitude control, and path tracking of a quadrotor using recursive neural networks," *International Journal of Engineering*, vol. 32, no. 5, pp. 747–758, 2019.

[38] F. Fei, Z. Tu, D. Xu, and X. Deng, "Learn-to-recover: Retrofitting uavs with reinforcement learning-assisted flight control under cyber-physical attacks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7358–7364.

[39] R. Fris, "The landing of a quadcopter on inclined surfaces using reinforcement learning," 2020.

[40] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[44] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7512–7519.

[45] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 2017, pp. 37–42.

[46] S. Mysore, B. El Mabsout, R. Mancuso, and K. Saenko, "Honey. i shrunk the actor: A case study on preserving performance with smaller actors in actor-critic rl," in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 01–08.

초 록

본 논문에서는 쿼드로터의 물리량을 모르는 상태에서도 강건하게 동작할 수 있는 강화 학습을 이용해 학습한 심층 순환 신경망 기반의 쿼드로터 제어기를 제안한다. 로봇 제어기는 로봇의 상태를 기반으로 모터 제어 신호를 생성하는 역할로 정의할 수 있는데, 이는 제어하려는 로봇의 동역학 모델에 맞게 미세조정해야 한다. 동역학 모델은 특정 모터 제어 신호를 가했을 때 로봇의 상태가 변하는 정도를 결정하는 것으로, 로봇의 물리량들에 따라 달라진다. 모든 실제 쿼드로터는 질량이나 회전 관성 등의 물리량이 불확실하기 때문에 같은 제품이라도 동역학 모델이 다를 수밖에 없다. 특히 쿼드로터는 비행 중 모터 과열이나 프로펠러 손상으로 인해 역학 모델이 바뀔 가능성이 높다.

이 문제를 해결하기 위해 순환 신경망 구조를 포함하는 깅화학습 기반 제어기 학습 기법을 제안한다. 강화학습은 쿼드로터 역학모델을 수학적으로 모델링하는 대신, 환경에서 얻은 데이터를 이용해 제어기를 훈련하는데 사용된다. 순환 신경망은 쿼드로터의 연속적인 상태들과 모터 신호들로부터 역학 모델에 대한 정보를 추출한다. 그러나 쿼드로터는 초당 제어 빈도수가 높고 컴퓨팅 장치의 성능이 제한적이기 때문에, end-to-end 방식으로 순환 신경망 구조를 사용했을 때 학습이 불안정해지는 문제가 발생한다. 따라서 강화학습 과정에서 심층 순환 신경망 구조를 포함하는 역학 모델 추출기 모듈을 따로 분리하는 방식을 제안한다. 역학 모델 추출기는 지도학습 방식으로 쿼드로터 상태들과 모터 신호들로부터 역학 정보를 예측되도록 훈련되며, 강화학습의 액터-크리틱 구조는 시뮬레이터에서 제공하는 역학 모델의 참값을 기반으로 훈련된다.

제안된 방법은 3차원 쿼드로터 제어기에 심층 순환 신경망을 적용한 최초의 연

구이며, Gym-pubullet-drone이라는 시뮬레이션 환경을 이용해 학습을 진행하였다. 시뮬레이터에서 학습이 진행될 때에도 실제 드론에 대한 적용 가능성을 검증하기 위해 Crazyflie 라는 실제 쿼드로터의 모든 하드웨어 제약조건을 만족하는 심층 네트워크 모델을 설계하였다. 제안된 제어기는 무작위 역학 모델을 가지는 쿼드로터 안정화 실험에서 기존의 모델 기반 제어기와 심층 신경망 기반 제어기보다 나은 성능을 보였고, 특히 모터 성능 저하나 무게중심 이동 등의 비대칭적 변화에 대해 더 효과적인 성능 개선을 검증하였다. 이러한 방식을 통해 비행 중 모터가 과열되거나 프로펠러가 손상되는 등 동역학 모델이 변화할 수 있는 상황에 대응해 제어가 가능할 것이다.

본 연구는 추후 몇 가지 개선 가능성을 시사하고 있다. 첫째로, 실제 쿼드로터 모델에 적용하기 위한 하드웨어 제약조건 때문에 제어기 모델의 사이즈가 제한되었다. 더 커다란 모델을 사용하여 학습한 후 network distillation이나 quantization을 사용하여, 동일한 성능을 보이면서 모델의 사이즈를 축소하는 방법론들을 적용할 수 있다. 둘째로, 보상 함수 디자인 과정에서 너무 많은 상충관계가 있기 때문에, 본 연구에서 사용한 보상함수가 최적이라는 보장이 없다. 추후 연구에서 바이너리 목표 도달 보상 함수를 사용하여, 보상 함수 디자인 과정을 생략할 수 있을 것이다.

주요어: 쿼드로터 제어, 순환 신경망, 강화학습, 로봇, 딥러닝

학번: 2020-27508