Low-level controller in response to changes in quadrotor dynamics

Jae-Kyung Cho[1], Chan Kim[1], Mohamed Khalid M Jaffar[2], Michael W. Otte[2], and Seong-Woo Kim[1*]

***Abstract*— The dynamics of all real quadrotors inevitably differ even if they are the same product. In particular, the dynamics can change significantly during the flight due to additional device attachments or overheating motors. In this study, we focus on training a low-level controller, which operates in response to dynamics-changes without prior knowledge or fine-tuning of the parameters, using reinforcement learning. We randomize the dynamics of quadrotors in the simulator and train the policy based on dynamics information extracted from the state–action history through recurrent neural networks (RNNs). In addition, our experiment demonstrates the difficulties in applying existing actor-critic structures that extract dynamics information using end-to-end RNNs for unstable quadrotors; hence, we propose a novel structure with better performance. Finally, the excellent performance of the proposed controller is verified by testing experiments that stabilize quadrotors with different dynamics. The experiment videos and the code can be found at `https://github.com/jackyoung96/RNN-Quadrotor-controller`.**

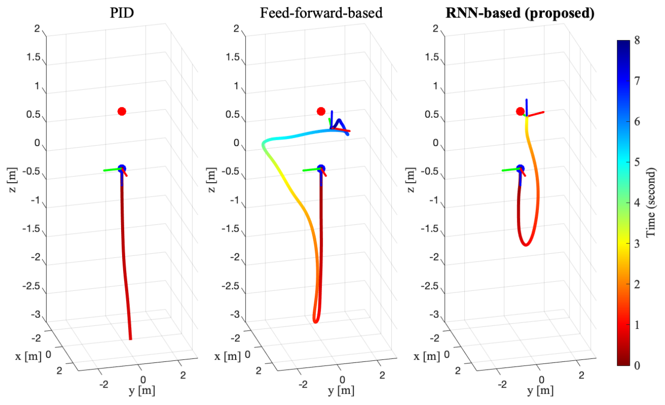

Fig. 1. Comparison of trajectories of controllers directing quadrotors to their target positions when the performance of one motor is reduced by 30%. Quadrotors are initialized as flipped status (blue dots) and aim to move to the goal (red dots).

I. INTRODUCTION

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, 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 of quadrotors by collecting data from interactions with the environment to replace the model-based controllers. 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. 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.

In this paper, 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 controller for a 3-D quadrotor. We divided the entire policy module into a dynamic extractor comprising the RNN and an actor composed of feed-forward networks, instead of using the conventional method of including an RNN in the actor for the end-to-end manner. The dynamics extractor module directly predicts dynamics parameters from state–action sequences using RNN and the feed-forward actor produces actions based on the predicted dynamics parameters. We trained the dynamics extractor and actor-critic separately using the ground truth of the dynamic parameters, which are explicit values in the simulator. With the proposed method, it is possible to maintain the controller performance in response to changes in dynamics. Fig. 1 displays the improved performance of the proposed method in an example scenario, which aims to recover from a flipped state and reach the

*Corresponding author

[1]The authors are with Seoul National University, Seoul, South Korea. {jackyoung96, chan_kim, snwoo}@snu.ac.kr

[2]The authors are with University of Maryland, Maryland, United States. {khalid26, otte}@umd.edu

desired final state (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 state and a shorter trajectory compared to the feed-forward networks-based controller.

The contributions of this study are summarized as follows:

- We propose a low-level quadrotor controller that operates in response to changes in dynamics without fine-tuning.
- We propose an RL structure that combines an auxiliary dynamics extraction module with an actor-critic model in a quadrotor environment where it would be difficult to learn using an end-to-end method.

II. Related Work

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.* [7] proposed a PID control method through quadrotor modeling and Ren *et al.* [8] presented a quadrotor-specified PID control method that performs position and altitude control in two steps. In addition, Faessler *et al.* [9] proposed a method to solve the problem of re-initializing drones when they are thrown or become unstable by an external intervention that was considered to require complex modeling. Due to the mathematical modeling of a controller, accurate physical parameters should be known and laborious fine-tuning was required.

Neural networks began to be used to approximate complex non-linear modeling of quadrotor controllers. Mohajerin *et al.* [10], [11] trained neural networks that approximate 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.* [12] and Tran *et al.* [13] 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.* [14] 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 in a supervised-learning fashion. However, these controllers focused on systems 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 learn based on data obtained through interaction with the 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 quadrotors with different dynamics.

Molchanov *et al.* [2] applied domain randomization to generate a robust low-level controller that operated successfully with various types of quadrotors when their physical quantities were unknown. In the data generation step, physical parameters such as mass, thrust-to-weight ratio, and the size of drones were sampled from a predefined physical property distribution for every episode in the simulator. Although the trained policy consisting of a feed-forward neural network produced robust actions regardless of the quadrotor's dynamics, customized actions were not produced based on quadrotor dynamics information that was not known in advance. Fei *et al.* [15] proposed a novel method to train a robust controller for drones to recover from unpredictable physical and cyber attacks. The physical attacks constituted of situations where the actuator signals and the sensor values were either 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.* [16] 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].

III. Preliminaries

In this section, we introduce a review and the notations of RL and the dynamics randomization technique. The standard RL problem can be described as finding a policy to maximize a return under the Markov decision process (MDP). Here, the MDP is defined as a 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 and action of the agent at timestep t are respectively denoted by $s_t \in \mathbb{S}$ and $a_t \in \mathbb{A}$. The transition probability $p_\mu(s_{t+1}|s_t, a_t)$ depends on parameter μ, which is the set of dynamics parameters, such as mass, body size, and thrust-to-weight ratio. The reward function $r : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ 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 policy $\pi : \mathbb{S} \to \mathbb{A}$ will return the action for a given state, which is trained to maximize the return $R_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$, where $\gamma \in [0, 1]$.

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

$$\pi^* = argmax_\pi J^\mu(\pi),$$
$$J^\mu(\pi) = \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)], \tag{1}$$

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

IV. 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 attitudes without any discontinuity or uniqueness issue. 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 in the 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:

$$\pi^*(a|s,h) = \underset{\pi}{\operatorname{argmax}} J^\mu(\pi), \quad \forall \mu \in M,$$
$$J^\mu(\pi) = \mathbb{E}_{s_0,a_0,s_1,\ldots} \sum_{t=0}^{\infty} -f(\|o_t\|, \|\omega_t\|), \tag{2}$$

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 a monotonically increasing function. It should be noted 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 $r(s_t) = -f(\|o_t\|, \|\omega_t\|)$.

TABLE I
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]	$\sim [-1,1]^3$

V. Method

A. Simulator setting and dynamics randomization

The Gym-pybullet-drones simulator [17] 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 two advantages: 1) it is possible to test many RL algorithm baselines easily because it is composed based on the Gym wrapper environment, which is a famous RL environment format; and 2) the reality gap is minimized by accounting for complex physical factors of quadrotors, such as air drag, downwash, and ground effects.

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 in the same quadrotor product. We independently randomized the XY coordinates of the center of mass, XYZ components 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 the 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:

$$f^i = k_f^i \times (p^i)^2, \ \tau^i = k_m^i \times (p^i)^2, \tag{3}$$

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

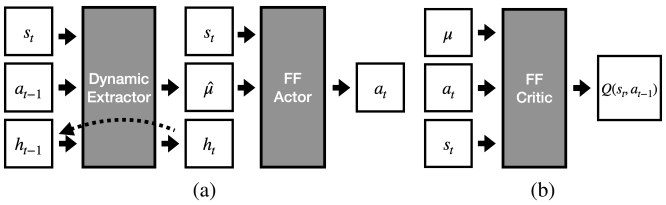

Fig. 2. (a) The actor of RNNparam consists of the dynamic extractor and the feed-forward actor. (b) The feed-forward critic of RNNparam uses the ground truth of dynamic parameters.

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 used as follows:

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

where dt is the control time interval, which was set to 10 ms in our experiment. Table I 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 I. In addition, state random noise was added to reflect the real-world environments.

B. Actor-critic model

Fig. 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) [18] algorithm combined with a gated recurrent unit (GRU) [19], 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. It is worth noting that a GRU can be simply replaced with vanilla RNN or LSTM; hence the general name RNN is used herein.

In the original dynamic randomization method [6], which we refer to as RNNfull herein, a recurrent network for extracting dynamics information was used for both actor and critic. 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. In the actor-critic model, the actor needs to extract information about the model dynamics through the RNNs because it is unknown information in the test phase. However, the critic is not used in the test phase to obtain the actual action. 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. Therefore, we employed the RNNpolicy structure, which uses a feed-forward network and ground truth dynamics parameters for the critic, as shown in Fig. 2(b), and only uses RNNs for the actor. This critic, which does not include the RNN structure, could increase the training performance and learning speed.

However, the RNN used in the actor still caused a learning instability problem. This was caused by the state–action history being too long to estimate dynamics, due to the control frequency of the quadrotor controller being high. To solve this problem, we proposed the RNNparam structure, which separates the entire policy networks into the dynamics extractor module and the feed-forward actor, as shown in Fig. 2(a). The feed-forward actor and the feed-forward critic were trained through a SAC algorithm. Since the actor uses the ground truth of the dynamics parameter μ as a given input, 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. 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 I 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 $\mathscr{B}$ (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

Algorithm 1 SAC RNNparam algorithm

```
1: 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 $\theta,\phi_1,\phi_2,\psi$
2: Initialize target networks $\bar{Q}_{\bar{\phi}_1}, \bar{Q}_{\bar{\phi}_2}$ $(\bar{\phi}_1 \leftarrow \phi_1, \bar{\phi}_2 \leftarrow \phi_2)$
3: Initialize empty replay buffer $\mathscr{B}$
4: for episode=1:E do
5:     Randomize dynamics parameter $\mu$, initial state $s_0$
6:     for t=0:T-1 do
7:         $s_{t+1} \sim p_\mu(\cdot|s_t,a_t)$ where $a_t \leftarrow \pi(s_t,\mu)$
8:     end for
9:     $\mathscr{B} \leftarrow \mathscr{B} \cup \{(s_0,a_0,r(s_0,a_0),\ldots,s_T,\mu)\}$
10:    Sample a minibatch of $|B|$ episodes
           $\{(s^i_0,a^i_0,r^i_0,\ldots,s^i_T,\mu^i)\}_{i=1,\ldots,|B|} \sim \mathscr{B}$
11:    Initialize losses $L_\pi = L_Q = L_D = 0$
12:    for i=b:$|B|$ do
13:        $t \sim \text{Uniform}(0,,1,\ldots,T-1)$
14:        $\hat{a}^i_t \leftarrow \pi_\theta(s^i_t,\mu^i)$
15:        $\hat{a}^i_{t+1} \leftarrow \pi_\theta(s^i_{t+1},\mu^i)$
16:        $q_t \leftarrow r_t +$
               $\gamma(\min_{i=1,2} \bar{Q}_{\bar{\phi}_i}(s_{t+1},\hat{a}_{t+1},\mu) - \alpha \log \pi_\theta(\hat{a}_{t+1}|s_{t+1}))$
17:        $L_\pi \leftarrow L_\pi + Q_{\phi_1}(s_t,\hat{a}_t,\mu) - \alpha \log \pi_\theta(\hat{a}_t|s_t)$
18:        $L_Q \leftarrow L_Q + (q_t - Q_{\phi_i}(s_t,a_t,\mu))^2$
19:        Initialize $a_{-1}$ and $h_{-1}$
20:        for t'=0:T-1 do
21:            $\hat{\mu}_{t'}, h_{t'} \leftarrow D_\psi(h_{t'-1}, s^i_{t'}, a^i_{t'-1})$
22:        end for
23:        $L_D \leftarrow L_D + \frac{1}{T}\sum_{t'=0}^{T-1}(\mu - \hat{\mu}_{t'})^2$
24:    end for
25:    $\phi_i \leftarrow \phi_i - \alpha_\phi \nabla_{\phi_i} \frac{1}{|B|} L_Q$
26:    $\bar{\phi}_i \leftarrow \tau\phi_i + (1-\tau)\bar{\phi}_i$
27:    $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta \frac{1}{|B|} L_\pi$
28:    $\psi \leftarrow \psi - \alpha_\psi \nabla_\psi \frac{1}{|B|} L_D$
29: end for
```

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:

$$a_t = \pi_\theta(s_t, D_\psi(h_{-1},s_0,a_{-1},\ldots,s_t,a_{-1})), \quad (5)$$

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

VI. Experimental results

In this section, we experimentally demonstrate that the proposed controller achieves good performance, even for

TABLE II
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}

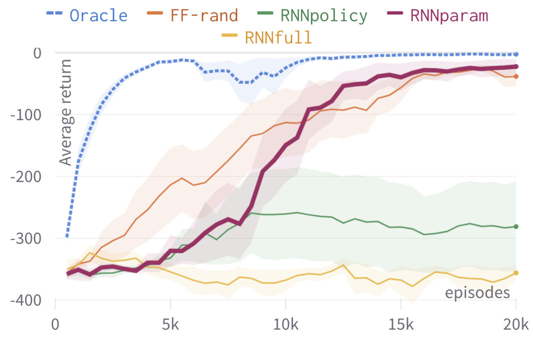

Fig. 3. Learning curve of five different actor-critic structures runs with five different seeds. The Oracle is the learning curve of FF-norand because the training was not interfered with by dynamics randomization.

quadrotor dynamics that are changed, compared to existing methods and other RNN actor-critic structures. We compared the three RNN-based controllers mentioned in Section V and three conventional controllers experimentally:

- PID controller, which needed to fine-tune 18 PID gain values in advance [17].
- 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.
- 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 ≤10 ms at 168 MHz Cortex-M4 MCU, and limiting model size limit of ≤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.* [20], 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 II.

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

$$r(s_t) = -(\|o_t\|^2 + 0.5\|\omega_t^z\|^2), \tag{6}$$

where $\|o_t\|$ is the Euclidean distance from the current position to the goal position, and ω_t^z is the yaw rate. The weighting for the yaw rate was set at 0.5 as that value showed the best performance through several experiments. The quadrotor had to change the direction of thrust by changing 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 considering only the yaw rate. We trained five policies for each method with varying seeds, and the learning curve of the average return of evaluation is shown in Fig 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. Thus, we conclude that end-to-end learning method using an RNN structure for both actor and critic is not suitable for unstable quadrotor settings. 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.

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. 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 I. The average position error to the goal e_p and the average magnitude of the yaw rate $\|\omega_{yaw}\|$ 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 III 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

TABLE III
EXPERIMENT FOR STABILIZING FROM RANDOM INITIAL STATE

dynamics randomize range $(1\pm\beta)$		PID	FF-norand	FF-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 = 0.0	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**

achieved the best performance when the dynamics were fixed (*i.e.,* $\beta = 0$). 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 RNNparam controller exhibited much smaller positional errors and angular velocity compared to the RNNpolicy controller using RNN in the end-to-end manner.

VII. CONCLUSION

In this study, we proposed a new RNN-based actor-critic structure for learning a low-level controller of a quadrotor that can operate in response to the implicit dynamics changes. We randomly altered the dynamics in the simulator and trained an actor-critic model using the SAC algorithm to extract dynamics information from state–action history using a RNNs. 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, we proposed a new possible structure. Through the proposed controller, it will be possible to control the quadrotor even in situations where its dynamics change unexpectedly during flight, such as when the motors overheat or the propellers get damaged.

ACKNOWLEDGEMENTS

This research was supported by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government (MOTIE) (P0017304, Human Resource Development Program for Industrial Innovation), in part by the National Research Foundation of Korea (NRF) through the Ministry of Science and ICT under Grant 2021R1A2C1093957, and in part by the BK21 FOUR (Fostering Outstanding Universities for Research) funded by the Ministry of Education(MOE, Korea) and National Research Foundation of Korea (NRF). The first and corresponding authors have participated in Integrated Major in Smart City Global Convergence, Seoul National University. The Institute of Engineering Research at Seoul National University provided research facilities for this work.

References

[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] 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.

[8] 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.

[9] M. Faessler, F. Fontana, C. Forster, and D. Scaramuzza, "Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 1722–1729.

[10] 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.

[11] ——, "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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.