Reinforcement Learning based Tuner for the Geometric Tracking Attitude Controller

Jongann Lee, *Dept. of Aerospace Engineering, Seoul National University,* Seoul, Republic of Korea
johnny3357@snu.ac.kr
H. Jin Kim, *Dept. of Aerospace Engineering, Seoul National University,* Seoul, Republic of Korea, hjin@snu.ac.kr

***Abstract*—The PID controllers are widely used across many applications including quadrotors. A variant of the PD controller is the geometric tracking controller, which utilizes the rotation matrix and the non-linear quadrotor dynamics. However, PID controllers require gain tuning, and their fixed gains render them incapable of responding to changes in the system in real-time. We propose a reinforcement learning based tuner for the attitude controller gains, which updates the gain in real time based on the history of the vehicle attitude error. The trained RL tuner is shown to be capable of stabilizing a vehicle with an unstable initial controller gain.**

***Index Terms*—PID tuning, Reinforcement Learning, Geometric Tracking Controller**

I. Introduction

Over the past decade, significant advancements have been made in drone technology. Progress in sensor technology, battery storage, and computing power has enabled the development of electrical drones capable of sustained stable flight. This technological evolution has facilitated the utilization of drones across a diverse range of applications, spanning from the agricultural industry to the military.

The multirotor has emerged as the dominant UAV form factor due to its superior accessibility, maneuverability, capacity for onboard sensors, and applicability to a wide range of applications [1]. Multirotors are characterized by the presence of multiple rotors arranged in the same plane and oriented vertically. This design enables the utilization of fixed-pitch propellers, with control input solely determined by the rotation speed of the propeller. This makes the platform mechanically very simple. It also means that the control properties are simple and well-understood. Notwithstanding their simplicity, they are very maneuverable and are capable of hovering stably in adverse conditions [2].

Most multirotors utilize a PID controller to stabilize and control the vehicle. PID controllers create a control input based on a linear combination of the error, the derivative of the error, and the integral of the error. The proportion of each of these terms are the PID parameters, and their value determines the behavior of the controlled system.

PID controllers require the tuning of their parameters to function correctly. While there are techniques for finding the appropriate parameters, it is still common to use intuition and gained knowledge to tune the PID gains.

A popular tuning technique is the Ziegler-Nichols tuning method. This method uses a concept called ultimate gain along with the oscillation period to assign the value of the PID parameters. Other ways to tune PID parameters include genetic algorithms, model matching, and pole assignment [3].

All of these techniques require a model or at the very least a step response of the system to work. However, models contain inaccuracies and accurate step responses are difficult to obtain. Therefore, on-the-spot adjustments to the parameters are still necessary. This is still mainly done by intuition based on the system's response to the input. It is therefore difficult for an inexperienced engineer to tune the PID gains correctly. Furthermore, once the tuning is complete, the PID controller is only capable of controlling the system it was tuned for. Its performance is greatly degraded if the dynamics of the plant changes over time.

A reinforcement learning(RL) based tuner would overcome all the previous issues, as a model-free RL algorithm does not require a model, and can be trained to update the controller gains in real-time based on the current state.

There has been previous research into combining RL with PID controllers. However, most of the previous approaches allow the RL algorithm to set the PID gains directly [4] [5] [6]. This makes the RL tuner inflexible as different systems have different responses to the same gain. Thus, to use the RL tuner on a new system, it will have to be trained again. Others have set the policy parameters themselves to be the PID gains, but this approach turns the RL algorithm into a very inefficient fixed-gain PID tuner [7].

The approach by Daesoo Lee, which is to multiply the previous control gains with the action, will allow an RL tuner to tune multiple quadrotors of different dimensions without retraining [8]. Our RL tuner also follows this approach, with a few modifications for improved performance.

The controller that will be tuned in this paper is the geometric tracking controller by Taeyoung Lee, which is a PD controller on the $SO(3)$ manifold that can track a desired trajectory [9]. The use of rotation matrices eliminates the singularity issue that affects Euler angles. Furthermore, the controller directly utilizes the non-linear quadrotor dynamics, making it free of linearization error.

The goal of this paper is to design an RL tuner for the geometric tracking attitude controller for the quadrotor that can tune the controller gains in real-time to stabilize the vehicle. To simplify the RL problem, only the attitude controller is tuned. Tuning the position controller once the attitude controller has been tuned is much easier as the vehicle is able to maintain the desired attitude.

II. Dynamics and Control

A. Model and Dynamics

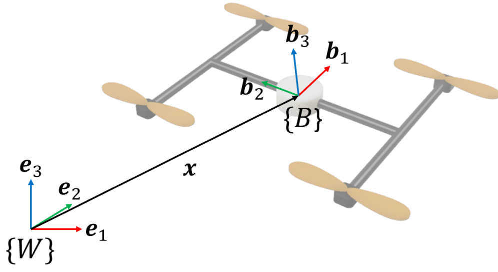

Fig. 1: The model of the vehicle

The vehicle model employs two coordinate frames: the world frame W and the body frame B. The center of the body frame B corresponds to the center of mass of the vehicle. The x-axis of the body frame $\boldsymbol{b}_1$ is the forward direction, the y-axis $\boldsymbol{b}_2$ is pointing left, and the z-axis $\boldsymbol{b}_3$ is pointing up. Thus, the body frame adopts a "North-West-Up" coordinate system. The world frame W also has its z-axis $\boldsymbol{e}_3$ pointing up, in the direction opposite to gravity.

The position of the center of mass is denoted as $\boldsymbol{x} \in \mathbb{R}^3$ and its velocity is denoted as $\boldsymbol{v} \in \mathbb{R}^3$. The rotation matrix of the vehicle with respect to the world frame is denoted as $R_{WB} \in \mathrm{SO}(3)$, where $R_{WB} = [\boldsymbol{b}_1 \;\; \boldsymbol{b}_2 \;\; \boldsymbol{b}_3]$. For trajectory generation, Z-Y-X Euler angles are utilized, with the z-rotation being yaw(ψ), y-rotation being pitch(θ), and x-rotation being roll(ϕ). The angular velocity of the vehicle with respect to the world frame is denoted as $\boldsymbol{\omega}_{BW}$.

The vehicle is modeled as a rigid body with mass m and moment of inertia J. The vehicle in question is fully actuated in its attitude and can generate thrust independently in two directions. The total moment in the body frame B is denoted as $\boldsymbol{M} \in \mathbb{R}^3$. The thrust, applied in the $\boldsymbol{b}_3$ direction, is denoted as $f > 0$.

With the above definitions, the dynamics of the vehicle is the following [10].

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{v} \\ m\dot{\boldsymbol{v}} &= -mg\boldsymbol{e}_3 + f\boldsymbol{b}_3 \\ \dot{R}_{WB} &= R_{WB}\hat{\boldsymbol{\omega}}_{BB} \\ \boldsymbol{M} &= J\dot{\boldsymbol{\omega}}_{BB} + \hat{\boldsymbol{\omega}}_{BB}J\boldsymbol{\omega}_{BB} \end{aligned} \tag{1}$$

where the hat map $\hat{\cdot} : \mathbb{R}^3 \to so(3)$ is defined such that $\hat{\boldsymbol{x}}\boldsymbol{y} = \boldsymbol{x} \times \boldsymbol{y}$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^3$.

B. Geometric Tracking Controller

The controller used is the geometric tracking controller by Taeyoung Lee which is based on the non-linear, $SE(3)$ dynamics of the vehicle and uses position and attitude error feedback [9]. It is designed to track a desired position trajectory $\boldsymbol{x}_d(t) \in \mathbb{R}^3$ and a desired attitude trajectory $R_d(t) \in \mathrm{SO}(3)$. However, due to the underactuated nature of the quadrotor dynamics, it is not possible to track an arbitrary position and attitude trajectory simultaneously. Typically, the desired attitude is defined using the desired yaw angle and the desired thrust direction [11].

From this point onward, the attitude of the vehicle R_{WB} will be simplified as R, and the angular velocity of the vehicle $\boldsymbol{\omega}_{BB}$ will be denoted as $\boldsymbol{\omega}$ in the body frame.

The tracking errors for position and velocity are defined as

$$\begin{aligned} e_x &= \boldsymbol{x} - \boldsymbol{x}_d \\ e_v &= \boldsymbol{v} - \boldsymbol{v}_d \end{aligned} \tag{2}$$

The error function on SO(3) is defined as

$$\Psi(R, R_d) = \frac{1}{2}\mathrm{tr}(I - R_d^T R) \tag{3}$$

This error function is locally positive definite about $R = R_d$ within the region where the rotation angle between R and R_d is less than 180° [12]. This region is equivalent to the set $L_2 = \{R_d, R \in \mathrm{SO}(3) | \Psi(R, R_d) < 2\}$, which is a sublevel set of Ψ and almost covers SO(3).

The attitude error is defined as

$$e_R = \frac{1}{2}(R_d^T R - R^T R_d)^\vee \tag{4}$$

The vee map $\vee : so(3) \to \mathbb{R}^3$ is defined as the inverse of the hat map.

The angular velocity error is defined as

$$e_\omega = \boldsymbol{\omega} - R^T R_d \boldsymbol{\omega}_d \tag{5}$$

Using the defined errors, the desired force $\boldsymbol{F}_{des}$ that should ideally be applied to the vehicle is

$$\boldsymbol{F}_{des} = -k_x e_x - k_v e_v + mg\boldsymbol{e}_3 + m\ddot{\boldsymbol{x}}_d \tag{6}$$

The actual, applied thrust f is defined as the projection of $\boldsymbol{F}_{des}$ on the $\boldsymbol{b}_3$y.

$$f = \boldsymbol{F}_{des} \cdot \boldsymbol{b}_3 \tag{7}$$

We now define the desired attitude R_d. $\boldsymbol{b}_{1d}$ is defined using the desired yaw and pitch, while $\boldsymbol{b}_{2d}, \boldsymbol{b}_{3d}$ are defined using the desired force $\boldsymbol{F}_{des}$.

$$\begin{aligned} \boldsymbol{b}_{1d} &= \mathrm{Rot}_z(\psi_d)\mathrm{Rot}_y(\theta_d)[1\ 0\ 0]^T \\ \boldsymbol{b}_{3d} &= \frac{\boldsymbol{F}_{des} - (\boldsymbol{F}_{des} \cdot \boldsymbol{b}_{1d})\boldsymbol{b}_{1d}}{\|\boldsymbol{F}_{des} - (\boldsymbol{F}_{des} \cdot \boldsymbol{b}_{1d})\boldsymbol{b}_{1d}\|} \\ \boldsymbol{b}_{2d} &= \boldsymbol{b}_{3d} \times \boldsymbol{b}_{1d} \end{aligned} \tag{8}$$

Here $\boldsymbol{F}_{des} - (\boldsymbol{F}_{des} \cdot \boldsymbol{b}_{1d})\boldsymbol{b}_{1d} \neq 0$ is assumed in order to properly define $\boldsymbol{b}_{3d}$.

The desired angular velocity $\boldsymbol{\omega}_d$ and desired angular acceleration $\dot{\boldsymbol{\omega}}_d$ are determined using the desired trajectory and its derivatives [11].

With all the necessary variables determined, the moment is defined as

$$\begin{aligned} \boldsymbol{M} = &-k_R e_R - k_\omega e_\omega + \boldsymbol{\omega} \times J\boldsymbol{\omega} \\ &- J(\hat{\boldsymbol{\omega}} R^T R_d \boldsymbol{\omega}_d - R^T R_d \dot{\boldsymbol{\omega}}_d) \end{aligned} \tag{9}$$

Note that the geometric tracking controller is essentially a PD controller with additional terms added for tracking the desired trajectory at the equilibrium. Looking at equation 9, the terms $-k_R e_R - k_\omega e_\omega$ correspond to the PD controller and the terms after that are the input required to follow the trajectory in the zero-error state.

III. Reinforcement Learning

Reinforcement learning(RL) is a subfield of machine learning where an agent learns to take the most optimal action to maximize the reward [13]. The RL problem consists of the agent and the environment. At a given timestep, the agent in state s_t takes action a_t, and receives reward r_t, forming a sequential transition $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \cdots)$. The action at each timestep is determined by a policy π which maps each state to a certain action or probability distribution of actions. Typically, the environment is assumed to be a Markov Decision Process(MDP), a requirement that holds true for quadrotor dynamics.

Q-learning is an RL technique that utilizes the Q-function to learn a policy. The Q-function is an action-value function that represents the expected reward given a certain state and action. For an agent at state s, with initial action a the optimal Q function is defined as

$$Q^*(s,a) = \max_{\pi} \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{10}$$

Here τ is an episode generated by a certain policy. The Q-function is typically expressed in its recursive form using the Bellman equation.

$$Q^*(s,a) = \mathbb{E}_{s'\sim P}[r(s,a) + \gamma \max_{a'} Q(s',a')] \tag{11}$$

Here $s \sim P$ means the state is sampled from the environment distribution, and $\gamma \in (0,1)$ is a decay rate set to give less reward to older experience.

Since the optimal Q-function takes both state and action as input, its value does not depend on the current policy. Therefore, algorithms that use Q^* are off-policy algorithms and can utilize a replay buffer $\mathcal{B}$ of previous experiences.

In deep RL,the Q-function is approximated using a deep neural network as Q_ϕ. This approximate is trained using the mean squared Bellman error, which roughly corresponds to how closely Q_ϕ comes to satisfying the Bellman equation.

$$L(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}\left[\left(Q_\phi(s,a) - (r + \gamma \max_{a'} Q_\phi(s',a'))\right)^2\right] \tag{12}$$

The parameters are then updated using stochastic gradient descent, optionally with a batch.

Deep Q-Network(DQN) is a deep Q-learning algorithm for a discrete action space [14]. It follows the above-mentioned structure and adds a target network for stable learning. In equation 12, the 'target' $r + \gamma \max_{a'} Q_\phi(s',a')$ is also dependent on the parameters ϕ, making the learning unstable. A target network $Q_{\phi'}$ is defined and is copied over from Q_ψ every fixed number of steps. With the target network, the loss becomes

$$L(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}\left[\left(Q_\phi(s,a) - (r + \gamma \max_{a'} Q_{\phi'}(s',a'))\right)^2\right] \tag{13}$$

While DQN has shown to be successful in discrete action spaces, extending it to the continuous action space is not straightforward. The optimal policy π^* is defined as the action that maximizes the optimal Q value given the state $\pi^*(s) = \max_a Q^*(s,a)$. Unlike a discrete action space, where finding the optimal action is as simple as trying all the possible actions, determining the action that maximizes the Q-function in a continuous action space is not trivial.

Deep Deterministic Policy Gradient(DDPG) solves this problem by employing a second neural network $\mu_\theta(s)$ that approximately maximizes Q_ϕ [15]. For DDPG, the action space is assumed to be continuous, and the Q-function is assumed to be differentiable with respect to the action. This allows for the training of $\mu_\theta(s)$ by performing gradient ascent using the reward

$$\max_{\theta} \mathbb{E}_{s\sim\mathcal{B}}[Q_\phi(s, \mu_\theta(s))] \tag{14}$$

Combined, this means the loss function for DDPG is

$$L(\phi) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{B}}\left[(Q_\phi(s,a) - (r + \gamma Q_{\phi'}(s', \mu_\theta(s'))))^2\right] \tag{15}$$

DDPG uses polyak averaging to update both the target network and the target policy network, to make the target update process more gradual. Every time the main network is updated, the target network and target policy are updated as

$$\begin{aligned} \phi' &\leftarrow \rho\phi' + (1-\rho)\phi \\ \theta' &\leftarrow \rho\theta' + (1-\rho)\theta \end{aligned} \tag{16}$$

where ρ is a hyperparameter between 0 and 1.

Twin Delayed DDPG(TD3) is a modified version of DDPG that exhibits desirable training characteristics [16]. While DDPG is capable of training optimal policies, it is very sensitive to hyperparameter tuning, making it difficult to train. TD3 overcomes this issue by employing three modifications.

First, TD3 deploys target policy smoothing to prevent incorrect peaks in $\mu_\theta(s)$. This is achieved by adding a clipped noise to the action and then clipping the action inside a valid action range.

$$a'(s') = \mathrm{clip}(\mu_{\theta'}(s') + \mathrm{clip}(\epsilon, -c, c), a_{low}, a_{high}), \quad \epsilon \sim \mathcal{N}(0,\sigma) \tag{17}$$

Next, a double Q-learning approach is used to prevent overestimation in the Q-function. Two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, are set up and are trained to the same target which is calculated using the smaller Q-function.

$$y(r,s') = r + \gamma \min_{i=1,2} Q_{\phi'_i}(s', \mu_{\theta'}(s)) \tag{18}$$

Finally, the optimal policy estimation $\mu_\theta(s)$ is learned using only Q_{ϕ_1}. Compared to DDPG, the policy is updated less frequently than the Q-function to further stabilize training.

IV. RL tuner

We will now present the methods used to create the RL tuner for the geometric attitude controller.

The overall schematics of the RL tuner can be seen in figure 2. The geometric tracking controller gives the input f, M to the quadrotor at a frequency of 120Hz. The RL tuner gathers the vehicle attitude as a state, then updates the parameters k_R, k_ω at a frequency of 10Hz. Thus, the parameters are updated every 12 control steps.

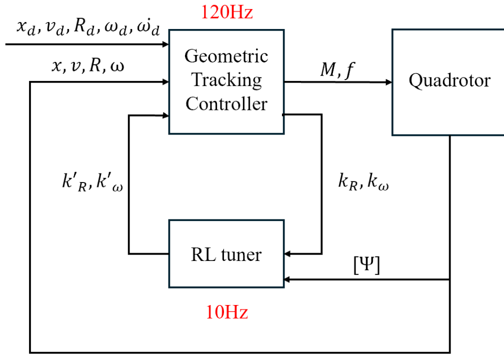

Fig. 2: Diagram of the RL tuner

A. The RL problem setup

All RL problems need to define three things: the state, the action, and the reward.

The state must contain enough information for the agent to perform the appropriate action. The goal of the RL tuner is to modify the attitude controller gains to stabilize the vehicle. Therefore the attitude error function $\Psi = \frac{1}{2}\mathrm{tr}(I - R_d^T R)$ is utilized for the state. However, using only the current attitude error is unsuitable for this problem. The geometric tracking controller is essentially a PD controller, and PD gains are often tuned by observing a step response that shows the state of the vehicle over time. Thus, a history of states is necessary for the successful operation of the agent. We set the state as a history of the states between each action, which in this case is the previous 12 attitude error measurements.

$$s_t = [\Psi_{t-11},\ ,\Psi_{t-10}, \cdots,\ \Psi_t] \tag{19}$$

The action in this problem is an update to the attitude controller gains. The RL tuner does not directly set the gain values as the optimal gain, which is approximately proportional to the moment of inertia, will be different for every vehicle. Thus, a direct setting of the gains would make the RL tuner only capable of tuning one specific quadrotor. Instead, the action is used as a multiplier to the existing gain, setting the product as the new gain. However, most RL algorithms require normalizing the action to $[-1, 1]$, as well as a symmetric action space if the action space is continuous. Therefore, the output of the policy is mapped to the action in the following manner.

$$\begin{aligned} \pi(s_t) &= a_t = [a_{t,0},\ a_{t,1}] \\ k'_R &= k_R \cdot 10^{a_{t,0}} \\ k'_\omega &= k_\omega \cdot 10^{a_{t,1}} \end{aligned} \tag{20}$$

The reward is set using the position error e_x and the action a_t. The reward is calculated at every step of the tuner, which comes with 12 simulation steps. The position error is added to the reward at every simulation step, while the absolute value of the action is added only once. The reward is then set as equation 21, with t_s being the time elapsed in the simulation.

$$r_t = \sum_{i=0}^{11} \left(\frac{1}{12}(1 - |(e_x)_{t-i}|) \right) - \frac{\lfloor t_s \rfloor}{2} |a_t| \tag{21}$$

The attitude error Ψ, used for the state, is not used in the reward as the attitude error is not dramatically large even for a very badly tuned controller. The position error is much more sensitive to controller gain, making it more suitable as a reward. The fact that the goal of the overall controller is to track a desired position adds to the suitability. The product of the simulation time and the absolute value of the action is subtracted to discourage large actions in later stages of the simulation, as the desired behavior is to quickly converge to a given gain and stay there.

B. RL Algorithm

The RL algorithm used to create the RL tuner is TD3. While DDPG is a popular choice for similar problems, TD3 is chosen as it exhibits more stable training characteristics. The algorithm is implemented in Python using the stable-baselines3 library [17]. The hyperparameters used for training are listed in table I. Any hyperparameters not listed in the table used the default value in stable baselines3.

TABLE I: Hyperparameters used to train the RL tuner

buffer size	200,000
learning starts	10,000
learning rate	1E-3
action noise	$\mathcal{N}(0, 0.1^2)$
batch size	256
gradient steps	1
train freq	1
gamma	0.98
tau	0.001

The state and the reward are normalized using the values stored in the replay buffer $\mathcal{B}$.

$$\begin{aligned} \dot{s}_t &= \frac{s_t - \mathrm{mean}(s_t|\mathcal{B})}{\mathrm{std}(s_t|\mathcal{B})} \\ \dot{r}_t &= \frac{r_t - \mathrm{mean}(r_t|\mathcal{B})}{\mathrm{std}(r_t|\mathcal{B})} \end{aligned} \tag{22}$$

The policy network μ_θ and the Q-function network Q_ϕ are both implemented as a multilayer perceptron(MLP). The MLP consists of two layers: a first layer of size 400 and a second layer of size 300. The activation function used is ReLU, with the exception being the output layer of the policy network which uses a tanh activation function.

The training is conducted for 500,000 timesteps. An `EvalCallBack` callback function is used to evaluate the performance of the policy every 1000 timesteps. The algorithm that obtains the highest evaluation reward during the training process is saved.

C. Simulation Environment

As reinforcement learning requires millions of iterations to converge to a useful policy, it is difficult to train them using only real data. A good simulation capable of generating adequate amounts of data is crucial for the successful training of an RL algorithm.

The stable-baselines3 library uses the gymnasium open source library to define the RL environment [18]. The gym environment provides a standard API to communicate between

the RL algorithm and the environment. The quadrotor is then simulated using the pybullet physics engine in the gym-pybullet-drones environment [19]. The physical properties of the simulated quadrotor are as follows.

$$\begin{aligned} &m = 0.027; \mathrm{kg} \\ &J = diag(1.4 \cdot 10^{-5},\ 1.4 \cdot 10^{-5},\ 2.17 \cdot 10^{-5}) \mathrm{kg} \cdot \mathrm{m}^2 \end{aligned} \tag{23}$$

The thrust and moment generated by the propeller are modeled as a quadratic function of the motor speed.

$$\begin{aligned} F_i &= k_F \cdot P_i^2,\ k_F = 3.16 \cdot 10^{-10} \\ M_z &= \sum_{i=0}^{3} (-1)^{i+1} k_T \cdot P_i^2,\ k_T = 7.94 \cdot 10^{-12} \end{aligned} \tag{24}$$

The RPM of the motor is capped to a value of 21,700.

The geometric tracking controller was implemented and tested in the simulation environment prior to training the RL tuner. The controller was manually tuned and the optimal gains were found to be the following.

$$k_x^* = 0.05 \qquad k_R^* = 0.05 \tag{25}$$

$$k_v^* = 0.04 \qquad k_\omega^* = 0.001 \tag{26}$$

For the training process, two environments are used: the training environment and the evaluation environment. Both environments have an episode length of 2,1 seconds. Both are set to hover around the point $\boldsymbol{x}_d = [0\ 0\ 1]^T$ with an initial position and attitude of $\boldsymbol{x}(0) = [0\ 0\ 1]^T, R(0) = \mathrm{Rot}_y(\pi/3)\mathrm{Rot}_x(\pi/3)$. The difference between the two environments is the initial gains used for the geometric tracking controller. For the training environment, a random gain sampled from a uniform distribution is used.

$$\begin{aligned} k_R(0) &= 5 \cdot 10^{z_R} \cdot 0.01,\ z_R \sim \mathcal{U}(-1, 1) \\ k_\omega(0) &= 0.1 \cdot 4^{z_\omega} \cdot 0.01,\ z_\omega \sim \mathcal{U}(-1, 1) \end{aligned} \tag{27}$$

The controller gains for the evaluation environment are set to be the optimal gains. The position controller gains k_x, k_v are set to their optimal values. Thus the RL tuner is trained on a variety of different controller gains, while evaluated on the fixed optimal gain for consistency.

V. Simulation Results

The trained RL tuner is tested in the same environment as the training environment: the desired trajectory is $\boldsymbol{x}_d(t) = [0\ 0\ 1]^T$ and the initial position and attitude is $\boldsymbol{x}(0) = [0\ 0\ 1]^T, R(0) = \mathrm{Rot}_y(\pi/3)\mathrm{Rot}_x(\pi/3)$. The simulation length is 2 seconds. 4 simulations were conducted, with the initial controller gains set as specified in table II.

TABLE II: Initial control gains used for each simulation

	k_R	k_ω
Simulation 1	$50 \cdot 0.01$	$0.2 \cdot 0.001$
Simulation 2	$0.5 \cdot 0.01$	$5 \cdot 0.001$
Simulation 3	$50 \cdot 0.01$	$5 \cdot 0.001$
Simulation 4	$0.5 \cdot 0.01$	$0.2 \cdot 0.001$

The two graphs on the left side of figure 3 showcase the absolute position error $|e_x|$ over time, and the trajectory of the vehicle. The data for the geometric tracking controller with gains updated from the RL tuner is plotted in blue, while the data for the controller not being tuned is plotted in red. The absolute error is shown to peak and then decrease for the controller with the RL tuner, while the controller without the tuner struggles to reduce its position error. A look at the trajectory reveals that the vehicle with the RL tuner is able to regain stability and heads toward the desired position, while the vehicle without the tuner hits the ground and fails to regain stability.

The changes in the controller gains over time are plotted in the right graphs of figure 3. The optimal gains have been plotted as a green dashed line and the changes in controller gains have been plotted in blue. While the values may not converge explicitly, they do change in the correct direction. It is also important to note that, unlike a traditional geometric tracking controller which has static gains, we are allowing for a changing gain, which may mean the above gain changes are more optimal. Lastly, near the equilibrium, the system is less sensitive to gain which is why it may not converge to the optimal gain at the end.

The results of the other three simulations are presented in figures 4 - 6. All three simulations show that the controller with the RL tuner outperforms the untuned controller. The control gain history graphs show that while final gains may not converge to the optimal value, the gains do move in the correct direction, decreasing the overly large gains and increasing the small gains.

VI. Conclusion

In this paper, an RL tuner for the geometric tracking attitude controller is developed for the purpose of correcting the controller gains of untuned controllers in real-time. The tuner is developed using the TD3 algorithm, which is a modified DDPG algorithm for better training stability. The tuner is designed to run at 10Hz, updating the controller gains by multiplying the previous gain with the action to the tenth power. Therefore, the action space is normalized and symmetric. The state is set to be the history of the attitude error between the previous action and the current action. Since the simulation ran at 120Hz, the state contained 12 attitude error measurements. The reward was set to be the position error with the magnitude of the action also added to discourage large actions later in the simulation. The trained RL tuner was deployed on a simulated quadrotor where it demonstrated the ability to modify the controller gains such that previously unstable controllers were made to be stable. The RL tuner modified the gains correctly, reducing larger-than-optimal gains and increasing the smaller gains.

References

[1] S. Tang and V. Kumar, "Autonomous flight," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 29–52, 2018.

[2] M. W. Mueller, S. J. Lee, and R. D'Andrea, "Design and control of drones," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 161–177, 2022.

[3] R. P. Borase, D. Maghade, S. Sondkar, and S. Pawar, "A review of pid control, tuning methods and applications," *International Journal of Dynamics and Control*, vol. 9, pp. 818–827, 2021.

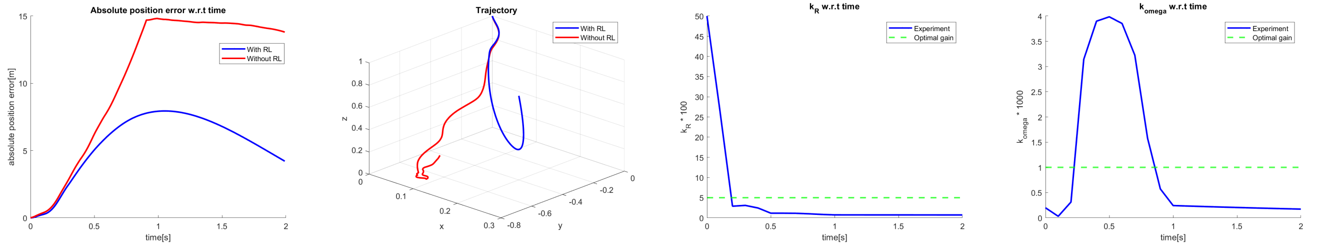

Fig. 3: Results of simulation 1

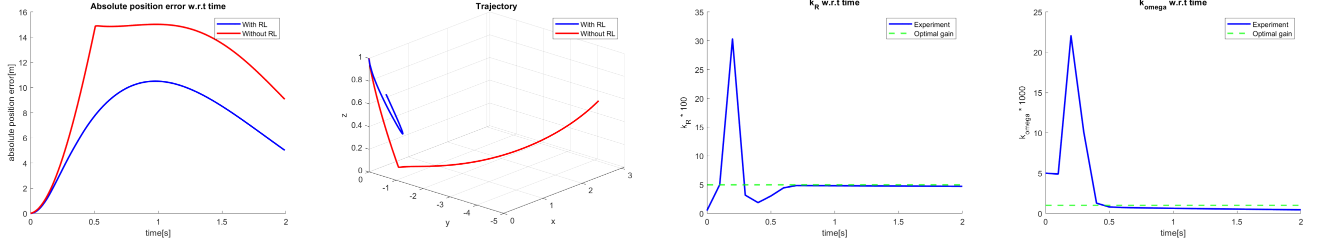

Fig. 4: Results of simulation 2

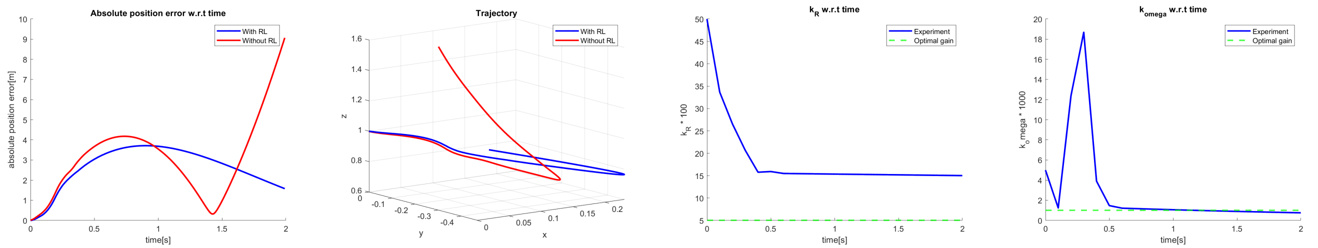

Fig. 5: Results of simulation 3

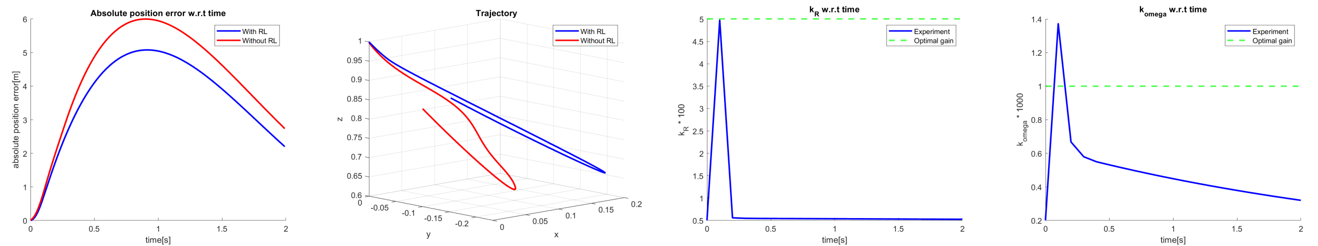

Fig. 6: Results of simulation 4

[4] O. Dogru, K. Velswamy, F. Ibrahim, Y. Wu, A. S. Sundaramoorthy, B. Huang, S. Xu, M. Nixon, and N. Bell, "Reinforcement learning approach to autonomous pid tuning," *Computers & Chemical Engineering*, vol. 161, p. 107760, 2022.

[5] M. Sedighizadeh and A. Rezazadeh, "Adaptive pid controller based on reinforcement learning for wind turbine control," in *Proceedings of world academy of science, engineering and technology*, vol. 27, pp. 257–262, Citeseer, 2008.

[6] T. Shuprajhaa, S. K. Sujit, and K. Srinivasan, "Reinforcement learning based adaptive pid controller design for control of linear/nonlinear unstable processes," *Applied Soft Computing*, vol. 128, p. 109450, 2022.

[7] N. P. Lawrence, M. G. Forbes, P. D. Loewen, D. G. McClement, J. U. Backström, and R. B. Gopaluni, "Deep reinforcement learning with shallow controllers: An experimental application to pid tuning," *Control Engineering Practice*, vol. 121, p. 105046, 2022.

[8] D. Lee, S. J. Lee, and S. C. Yim, "Reinforcement learning-based adaptive pid controller for dps," *Ocean Engineering*, vol. 216, p. 108053, 2020.

[9] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*, pp. 5420–5425, IEEE, 2010.

[10] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE robotics & automation magazine*, vol. 19, no. 3, pp. 20–32, 2012.

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

[12] F. Bullo and A. D. Lewis, *Geometric control of mechanical systems: modeling, analysis, and design for simple mechanical control systems*, vol. 49. Springer, 2019.

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

[14] 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.

[15] 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.

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

[17] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-baselines3: Reliable reinforcement learning implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[18] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, “Gymnasium,” Mar. 2023.

[19] 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)*, pp. 7512–7519, IEEE, 2021.