Waypoint Tracking Control for a Quadrotor based on PID and Reinforcement Learning

Xurui Bao, Zhouhui Jing

School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China

Abstract: Waypoint tracking is a common task for quadrotors, which is usually achieved by the Inner-Outer loop PID control method. The outer loop is implemented by establishing PID control for the position and velocity errors of the quadrotor. This method requires the quadrotor's desired flight rate as an input. However, setting a fixed desired flight rate cannot match to changes in waypoints and the quadrotor's own state, which can lead to a large time consumption or even mission failure. To solve this problem, this paper uses a neural network trained by the DDPG algorithm to control the desired flight rate of the quadrotor during flight, so that the quadrotor can adjust the desired flight rate flexibly to achieve a better waypoint tracking performance. Simulation results show that the method proposed in this paper can improve the performance of the Inner-Outer loop PID control system through adjusting the desired flight rate.

Keywords: quadrotor, waypoint tracking, PID, reinforcement learning, neural network

1. INTRODUCTION

In recent years, tremendous progress has been made in researches related to quadrotors (L'Afflitto et al., 2018). And this leads to a wide range of applications, such as in the field of photography, transportation and rescue (Penicka and Scaramuzza, 2022). Waypoint tracking is one of the common tasks for quadrotors (Larsson et al., 2020; Qian et al., 2017).

For the quadrotor waypoint tracking task, the dominant solutions are derived from PID control, which consists of proportional, integral and differential components. Due to its simple structure, PID control has effective parameter tuning methods (Mohamad Ali Tousi et al., 2020; Kumar et al., 2021). Therefore, it is widely used in industry (Mahmud et al., 2020; Bo et al., 2016). The Inner-Outer loop PID control method derives from PID control and has been widely used for quadrotor control (Julkananusart and Nilkhamhang, 2015). (Saraf et al., 2020) approximately linearized the model of the quadrotor and presented a comparison between the PID method and LQR method acting on a quadrotor. However, the fixed PID parameters cannot adapt to complex path conditions and the quadrotor may generate large error signals along some complicated paths, causing PID controllers to fail. (Zhang et al., 2020) proposed a PID gain adjustment method based on the reinforcement learning algorithm, and improved the dynamic performance and the stability of the PID controller. (Cajo et al., 2018) used a fractional order PD control method to achieve the path-following control of the quadrotor. (Farid et al., 2018) used interpolation methods on linear trajectories to generate a series of waypoints to generate a smoother path in order to increase the possibility of completing the task. But this method does not reduce the time consumption of the task. (Song et al., 2022) introduced the fuzzy PID controller to improve the flight stability and the robustness of the quadrotor. But the simple fuzzy processing of the information of the control system leads to a reduction in the control accuracy, and the design of the fuzzy rule relies on experience.

With the development of neural networks and deep learning, people begin to seek for new control methods for quadrotors. (Lambert et al., 2019) used neural networks to predict changes of the state of a quadrotor with different control inputs and selected the one that brought the predicted velocity and acceleration closest to 0 to keep the quadrotor hovering in the air. (Rubí et al., 2020) used neural networks and the DDPG algorithm to calculate the expected yaw angle of the quadrotor. (Song et al., 2021) used neural networks trained by the PPO algorithm, one of the reinforcement learning algorithms (Schulman et al., 2017), to directly control the rotor voltage to complete a racing competition in a less time consumption than quadrotors controlled by professionals. This shows that it is feasible to control quadrotors through reinforcement learning algorithms and neural networks.

However, controlling quadrotors through neural networks faces two major challenges: a long training time consumption and difficulties in convergence (Hwangbo et al., 2017). These mainly come from two points: the control instability at the beginning of the learning process leading to a large time consumption for trial and error and the difficulties of designing a proper reward function to avoid local optima of the RL algorithm.

The contribution of this paper is using neural networks trained by the DDPG algorithm to control the desired flight rate, which is the input of the Inner-Outer loop PID control method of the control system. On the one hand, PID control guides the quadrotor in the right direction to ensure a reasonable reward during reinforcement learning, which largely reduces the training difficulties and time consumptions. On the other hand, the DDPG algorithm makes use of the nonlinearity and self-learning ability of neural networks to approximate a control policy of the desired flight rate. The simulation result proves that using neural networks trained by DDPG to control the input of PID controller improves the performance of the original Inner-Outer loop PID control method.

2. THE MODEL OF A QUADROTOR

2.1 Rigid Body Model of a Quadrotor

The model of a quadrotor used in this paper is based on the Section 4 of the work from (Zhang et al., 2014). The quadrotor is a 6 degree-of-freedom rigid body.

There are two reference frame concepts in this paper: the flat Earth reference frame $(O_eX_eY_eZ_e)$ and the body-fixed coordinate frame $(O_bX_bY_bZ_b)$. The schematic of the quadrotor and reference frames is shown in Fig. 1.

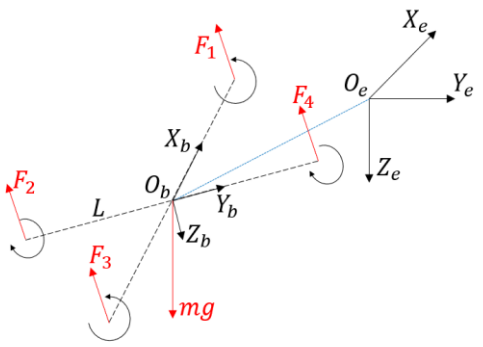


Fig. 1. The schematic of the quadrotor and reference frames.

Where m is the mass of the quadrotor, g is the gravitational constant, L is the arm length of the quadrotor, which is the distance from the center of the rotor to the center of mass of the quadrotor, and F_i is the thrust force generated by rotor i.

Let the inertia matrix be $J = \text{diag}(I_{xx}, I_{yy}, I_{zz})$, where I_{xx}, I_{yy}, I_{zz} are inertia components along x, y and z axis.

During the flight, the position p_e and the velocity v_e in the flat Earth reference frame satisfy the relationship:

$$\dot{p_e} = v_e \tag{1}$$

The body angular velocity ω_b and the attitude angles of the quadrotor satisfy the relationship:

$$\begin{aligned}\omega_b &= \begin{bmatrix}\dot{\varphi}\\0\\0\end{bmatrix} + \begin{bmatrix}1&0&0\\0&\cos\varphi&\sin\varphi\\0&-\sin\varphi&\cos\varphi\end{bmatrix}\begin{bmatrix}0\\\dot{\theta}\\0\end{bmatrix}\\ &+\begin{bmatrix}1&0&0\\0&\cos\varphi&\sin\varphi\\0&-\sin\varphi&\cos\varphi\end{bmatrix}\begin{bmatrix}\cos\theta&0&-\sin\theta\\0&1&0\\\sin\theta&0&\cos\theta\end{bmatrix}\begin{bmatrix}0\\0\\\dot{\psi}\end{bmatrix}\\ &=\begin{bmatrix}1&0&-\sin\theta\\0&\cos\varphi&\sin\varphi\cos\theta\\0&-\sin\varphi&\cos\varphi\sin\theta\end{bmatrix}\begin{bmatrix}\dot{\varphi}\\\dot{\theta}\\\dot{\psi}\end{bmatrix}\end{aligned} \tag{2}$$

Where φ is the roll angle, θ is the pitch angle and ψ is the yaw angle.

In accordance with Newton's second law, the linear motion of the quadrotor is generated by the combined external forces provided by four rotors, and satisfies the relationship:

$$m\dot{v_e} = \begin{bmatrix}0\\0\\mg\end{bmatrix} - R_{b2e}\begin{bmatrix}0\\0\\\sum_{i=1}^{4}F_i\end{bmatrix} \tag{3}$$

Where R_{b2e} is the rotation matrix from the body-fixed coordinate frame to the flat Earth reference frame.

$$R_{b2e} = \begin{bmatrix}\cos\theta\cos\psi & \sin\varphi\sin\theta\cos\psi-\cos\varphi\sin\psi & \sin\varphi\sin\psi+\cos\varphi\sin\theta\cos\psi\\ \cos\theta\sin\psi & \cos\varphi\cos\psi+\sin\varphi\sin\theta\sin\psi & \cos\varphi\sin\theta\sin\psi-\sin\varphi\cos\psi\\ -\sin\theta & \sin\varphi\cos\theta & \cos\varphi\cos\theta\end{bmatrix} \tag{4}$$

The angular motion of the drone is generated by the moments of four rotors, and the relationship between the moments and the body angular velocity can be obtained according to Euler's equation as:

$$J\dot{\omega_b} = -\omega_b \times (J\omega_b) + \begin{bmatrix}L(F_2-F_4)\\L(F_1-F_3)\\M_2+M_4-M_1-M_3\end{bmatrix} \tag{5}$$

Where M_i is the moment generated by rotor i.

2.2 Motor Dynamical Model

The thrust force and the moment generated by a rotor are proportional to the square of the rotor speed (Li, Qi et al., 2020). Assume each rotor has a rotational force coefficient of c_F and a rotational moment coefficient of c_M. The actual rotor speed of rotor i is n_i.

Then the thrust forces and moments generated by rotors can be obtained by following formulas:

$$F_i = c_F n_i^2, i = 1,2,3,4 \tag{6}$$

$$M_i = c_M n_i^2, i = 1,2,3,4 \tag{7}$$

The actual rotor speed and the desired rotor speed of rotor i satisfy the relationship:

$$\dot{n_i} = \frac{1}{T}\left(n_i^{des} - n_i\right) \tag{8}$$

Where T is the time constant of the first-order inertial system and n_i^{des} is the desired rotor speed of rotor i. The actual rotor speed is limited to the range of $[1200{,}7800]$rpm.

The desired rotor speeds can be broken down by the effect produced into different components: the component n_h that keeps the vehicle hovering, the component Δn_F that produces the lift, and the components Δn_φ, Δn_θ, Δn_ψ that cause the attitude angles to change. Thus, we can calculate the four desired rotor speeds by using the following equation:

$$\begin{bmatrix}n_1^{des}\\n_2^{des}\\n_3^{des}\\n_4^{des}\end{bmatrix} = \begin{bmatrix}1&0&1&-1\\1&1&0&1\\1&0&-1&-1\\1&-1&0&1\end{bmatrix}\begin{bmatrix}n_h+\Delta n_F\\\Delta n_\varphi\\\Delta n_\theta\\\Delta n_\psi\end{bmatrix} \tag{9}$$

From the force analysis of the hovering state, n_h satisfies the relationship:

$$4c_F n_h^2 = mg \tag{10}$$

Hence, n_h can be calculated as:

$$n_h = \sqrt{\frac{mg}{4c_F}} \tag{11}$$

The variables and parameters used to model the quadrotor in Section 2 are shown in Table 1 and Table 2 according to the order of appearance.

Table 1. Variables used to model the quadrotor.

Symbol (Unit)	Definition
$p_e(\mathrm{m})$	Position in the geodetic coordinate system
$v_e(\mathrm{m}\cdot\mathrm{s}^{-1})$	Velocity in the geodetic coordinate system
$\omega_b(\mathrm{rad}\cdot\mathrm{s}^{-1})$	Body angular velocity
$\varphi(\mathrm{rad})$	Roll angle
$\theta(\mathrm{rad})$	Pitch angle
$\psi(\mathrm{rad})$	Yaw angle
R_{b2e}	Rotation matrix
$F_i(\mathrm{N})$	Thrust force generated by rotor i
$M_i(\mathrm{N}\cdot\mathrm{m})$	Moment generated by rotor i
$n_i(\mathrm{rpm})$	Actual rotor speed of rotor i
$n_i^{des}(\mathrm{rpm})$	Desired rotor speed of rotor i
$n_h(\mathrm{rpm})$	Rotor speed component for hovering
$\Delta n_F(\mathrm{rpm})$	Rotor speed component to produce lift
$\Delta n_\varphi(\mathrm{rpm})$	Rotor speed component to change roll angle
$\Delta n_\theta(\mathrm{rpm})$	Rotor speed component to change pitch angle
$\Delta n_\psi(\mathrm{rpm})$	Rotor speed component to change yaw angle

3. PID CONTROL OF WAYPOINT TRACKING

The Inner-Outer loop PID control method is the most common control method for quadrotors, which consists of PID controllers. It is easy to understand what this method is trying to do. What's more, this method requires little computation time and memories which makes it possible for further improvements (Cao, 2016; Parivash et al., 2017).

The outer loop calculates the force needed by the quadrotor to track a path, and then converts the calculated force into the desired attitude angles and the thrust force for lift. The inner loop uses the PID control method to follow the desired attitude angles.

The structure of the Inner-Outer loop PID control method with constant desired flight rate is shown in Fig. 2 and the inputs to this system are a set of waypoints and the desired flight rate v^{des} of the quadrotor. The implementation of the control modules will be interpreted in Section 3.1 and Section 3.2.

Table 2. Parameters used to model the quadrotor.

Symbol (Unit)	Definition	Value
$m(\mathrm{kg})$	Mass of the quadrotor	0.5
$L(\mathrm{m})$	Arm length	0.2
$J(\mathrm{N}\cdot\mathrm{s}^2\cdot\mathrm{rad}^{-1})$	Inertia Matrix	$\mathrm{diag}(I_{xx}, I_{yy}, I_{zz})$
$I_{xx}(\mathrm{N}\cdot\mathrm{s}^2\cdot\mathrm{rad}^{-1})$	Inertia components along x axis	0.114
$I_{yy}(\mathrm{N}\cdot\mathrm{s}^2\cdot\mathrm{rad}^{-1})$	Inertia components along y axis	0.114
$I_{zz}(\mathrm{N}\cdot\mathrm{s}^2\cdot\mathrm{rad}^{-1})$	Inertia components along z axis	0.158
$g(\mathrm{m}\cdot\mathrm{s}^{-2})$	Gravitational constant	9.8
$c_F(\mathrm{N}\cdot\mathrm{rpm}^{-2})$	Rotational force coefficient	6.11×10^{-2}
$c_M(\mathrm{N}\cdot\mathrm{m}\cdot\mathrm{rpm}^{-2})$	Rotational moment coefficient	1.5×10^{-9}
$T(s)$	Time constant of the first-order inertial system	10
$n_h(\mathrm{rpm})$	Rotor speed component for hovering	$\sqrt{\frac{mg}{4c_F}}$

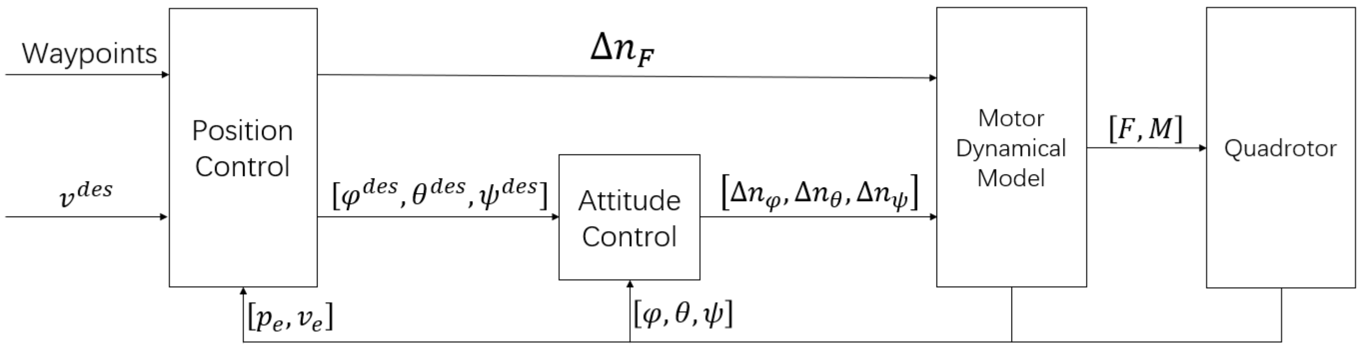

Waypoints
v^{des}
Position Control
Δn_F
$[\varphi^{des}, \theta^{des}, \psi^{des}]$
Attitude Control
$[\Delta n_\varphi, \Delta n_\theta, \Delta n_\psi]$
Motor Dynamical Model
$[F, M]$
Quadrotor
$[p_e, v_e]$
$[\varphi, \theta, \psi]$

Fig. 2. The structure of the original PID control method.

3.1 Attitude Control

According to (9), the control signals for different attitude angles are decoupled, so it's easy to establish PD control over attitude angles respectively. The control signals can be obtained by the following formulas:

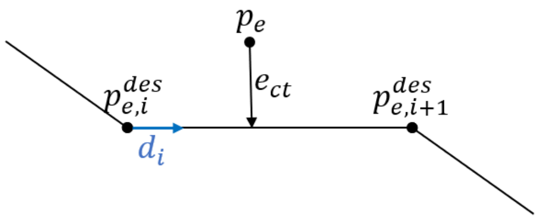
$$\begin{cases} \Delta n_\varphi = k_{p,\varphi}(\varphi^{des}-\varphi) + k_{d,\varphi}(\dot{\varphi}^{des}-\dot{\varphi}) \\ \Delta n_\theta = k_{p,\theta}(\theta^{des}-\theta) + k_{d,\theta}(\dot{\theta}^{des}-\dot{\theta}) \\ \Delta n_\psi = k_{p,\psi}(\psi^{des}-\psi) + k_{d,\psi}(\dot{\psi}^{des}-\dot{\psi}) \end{cases} \tag{12}$$

Where φ^{des}, θ^{des}, ψ^{des} are the desired attitude angles and $k_{i,j}, i = p, d, j = \varphi, \theta, \psi$ are the control parameters of PD controllers.

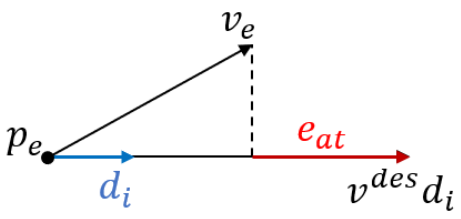
3.2 Position Control

(Hoffmann et al., 2008) proposed a control method for quadrotor waypoint tracking. They established PID control over the cross-segment position error and the velocity error. Section 3.2 is based on the work of Hoffmann.

The position error from the segment e_{ct} and the speed error e_{at} of the quadrotor during flight are shown in Fig. 3.



(a) Position error



(b) Speed error

Fig. 3(a), (b). The position error and the speed error of the quadrotor.

Where d_i is the unit vector pointing from the waypoint $p_{e,i}^{des}$ to the waypoint $p_{e,i+1}^{des}$. e_{ct} and e_{at} can be computed from (13) and (14).

$$e_{ct} = \left(\left(p_e - p_{e,i}^{des}\right) \cdot d_i\right) \cdot d_i - \left(p_e - p_{e,i}^{des}\right) \tag{13}$$

$$e_{at} = v^{des} \cdot d_i - (v_e \cdot d_i) \cdot d_i \tag{14}$$

PID control is established to e_{ct} and PI control to e_{at}. The control signals along x, y and z axis is obtained as follow:

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = k_{p,ct} e_{ct} + k_{d,ct} \dot{e_{ct}} + k_{i,ct} \int_0^t e_{ct} dt + k_{p,at} e_{at} + k_{i,at} \int_0^t e_{at} dt \tag{15}$$

Assume that the roll angle φ and the pitch angle θ of the quadrotor are very small, the yaw angle ψ changes in a small range and the total pull is approximately equal to the gravity of the quadrotor:

$$\begin{cases} \sin\varphi \approx \varphi, \cos\varphi \approx 0 \\ \sin\theta \approx \theta, \cos\theta \approx 0 \\ \psi \approx \psi^{des} \\ \sum_{i=1}^{4} F_i \approx mg \end{cases} \tag{16}$$

φ^{des}, θ^{des}, Δn_F can be calculated by substituting (16) into (3).

$$\begin{cases} \varphi^{des} = -\frac{1}{g}(u_x \sin\psi - u_y \cos\psi) \\ \theta^{des} = -\frac{1}{g}(u_x \cos\psi + u_y \sin\psi) \\ \Delta n_F = -\frac{m}{8 c_F n_h} u_z \end{cases} \tag{17}$$

By combining (9), (12) and (17), the desired motor speeds of the four rotors can be computed. Thus, the Inner-Outer loop PID control method for quadrotor waypoint tracking task is obtained.

Table 3. Variables used in PID controllers.

Symbol (Unit)	Definition
$v^{des}(\mathrm{m \cdot s^{-1}})$	Desired flight rate
$\varphi^{des}(\mathrm{rad})$	Desired roll angle
$\theta^{des}(\mathrm{rad})$	Desired pitch angle
$\psi^{des}(\mathrm{rad})$	Desired yaw angle
$e_{ct}(\mathrm{m})$	Position error from the segment
$e_{at}(\mathrm{m \cdot s^{-1}})$	Speed error
$p_{e,i}^{des}(\mathrm{m})$	Waypoint i
d_i	Unit vector pointing from $p_{e,i}^{des}$ to $p_{e,i+1}^{des}$
$u_x(\mathrm{m \cdot s^{-2}})$	Control signals along x axis
$u_y(\mathrm{m \cdot s^{-2}})$	Control signals along y axis
$u_z(\mathrm{m \cdot s^{-2}})$	Control signals along z axis

The variables and parameters of PID controllers in Section 3 are shown in Table 3 and Table 4 according to the order of appearance.

Table 4. Parameters of PID controllers.

		Proportionality Coefficient	Integration Coefficient	Differential Coefficient
Attitude Control	φ	2000	0	4000
	θ	2000	0	4000
	ψ	800	0	4000
Position Control	e_{ct}	0.5	3×10^{-4}	1
	e_{at}	0.5	3×10^{-4}	0

4. CONTROL OF DESIRED FLIGHT RATE THROUGH NEURAL NETWORKS

4.1 Task Model

The task was modelled on a Markov Decision Process (MDP) which is defined by the tuple (s_t, a_t, r_t, s_{t+1}).

The observation space is $s_t = [p_o, v_e, \omega_b, R_{b2e}]$, where p_o is the relative positions to waypoints, and R_{b2e} is the expansion of the rotation matrix. The action space is $a_t = [v^{des}]$. The reward r_t is used to evaluate the performance of the action a_t taken in the observation space s_t.

4.2 Control Method

The structure of the control method proposed in this paper is shown in Fig.4. To change the desired flight rate of the quadrotor flexibly, the observation space of the quadrotor s_t is mapped onto the action space a_t using an Actor neural network. And this paper uses the DDPG algorithm, one of the reinforcement learning algorithms, to train the Actor network to give it proper weights for controlling the desired flight rate.

4.3 DDPG Algorithm

The DDPG algorithm is a reinforcement learning algorithm based on the AC architecture. It derives from the DQN algorithm and uses an Actor neural network to compensate for the inability of handling continuous control tasks (Lillicrap et al., 2015). The DDPG algorithm has been successfully applied in different environments (Zhang et al., 2020; Li, Ma et al., 2020). There are also precedents for using the DDPG algorithm in the field of quadrotor control (Rubí et al., 2020; Rodriguez-Ramos et al., 2019).

The Actor network in the DDPG algorithm is used to fit a policy function that maps the observation space directly to the action space, while the Critic network is used to fit a function that calculates the Q value, which is the expectation of the total future reward that the RL agent will receive. Under the observation space s_t, a better action a_t will get a higher Q value. When the Actor policy is μ, which means $\mu(s_t) = a_t$, the Q value of action a_t in s_t is as follow:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^{\mu}\left(s_{t+1}, \mu(s_{t+1})\right)\right] \quad (18)$$

Where Q^{μ} means the Q value is calculated based on the policy μ, $\mathbb{E}_{r_t, s_{t+1} \sim E}$ denotes the expected discounted future reward from taking a_t in s_t to the end of the task, $r(s_t, a_t)$ is the step reward received by the RL agent when taking a_t in s_t and $\gamma \in [0,1]$ represents the discount factor for the future reward.

The goal of the DDPG algorithm is to find the strategy μ^* that brings the greatest expected discounted reward in one episode.

$$\mu^* = \operatorname{argmax} \mathbb{E}_{\mu^*}\left[\sum_{t=0}^{t_f} \gamma^t r(t)\right] \quad (19)$$

Where t_f is the end time of the task.

Assume that there is an initialized Critic network $Q^{\mu}(s, a|\theta^Q)$ and an initialized Actor network $\mu(s|\theta^{\mu})$ where θ^Q and θ^{μ} are respectively the weights of the Critic network and the Actor network. From (18), it is clear that the calculation of $Q^{\mu}(s_t, a_t)$ requires the use of $Q^{\mu}\left(s_{t+1}, \mu(s_{t+1})\right)$. Therefore, updating the network parameters of $Q^{\mu}(s, a|\theta^Q)$ will produce a situation where the Critic network estimates itself, resulting in overestimation and oscillation of the Q value. To make the training process more stable, the DDPG algorithm introduces the target Critic network $Q'^{\mu'}(s, a|\theta^{Q'})$ and the target Actor network $\mu'(s|\theta^{\mu'})$ with the same initial weights as the original networks.

To get the greatest reward shown in (19), the Actor network is updated by maximizing $Q^{\mu}(s, a)$.

$$\max_{\theta^{\mu}} Q^{\mu}\left(s, \mu(s)\right) \quad (20)$$

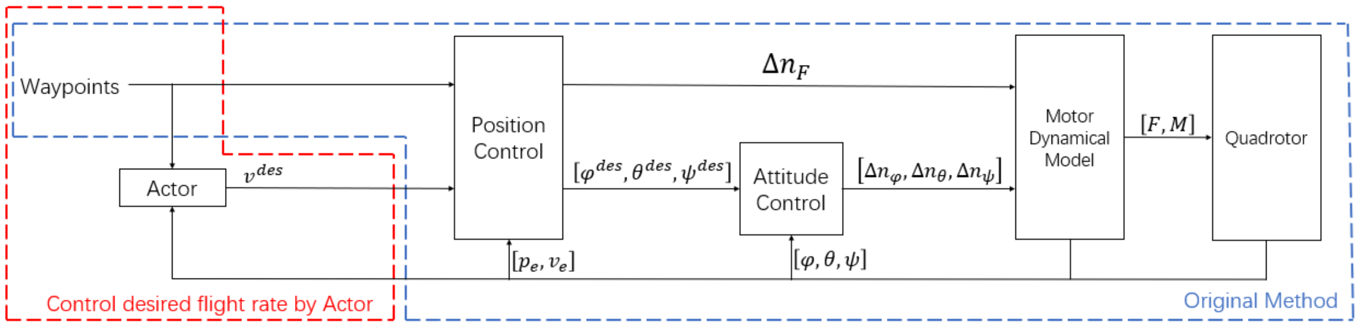
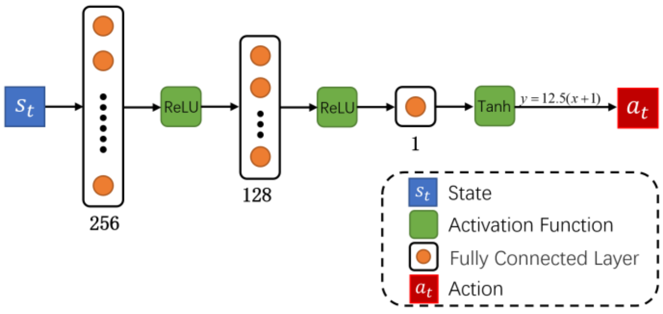

Waypoints
Actor
v^{des}
Position Control
Δn_F
$[\varphi^{des}, \theta^{des}, \psi^{des}]$
Attitude Control
$[\Delta n_\varphi, \Delta n_\theta, \Delta n_\psi]$
Motor Dynamical Model
$[F, M]$
Quadrotor
$[p_e, v_e]$
$[\varphi, \theta, \psi]$
Control desired flight rate by Actor
Original Method

Fig. 4. The structure of the control method proposed in this paper.

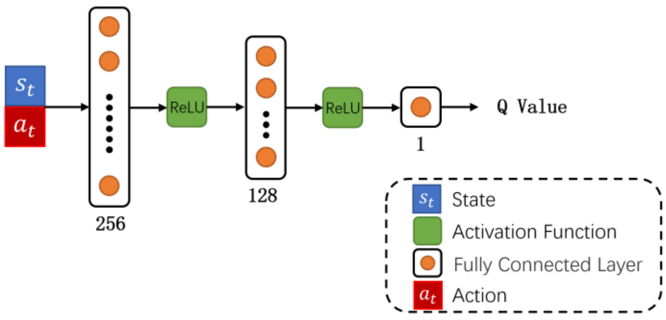
4.5 Reward Function

For the quadrotor waypoint tracking, the major task is to reduce the time consumption for a quadrotor to pass all waypoints in sequence. Once the waypoints are set, the quadrotor is expected to fly on the line connecting adjacent waypoints, and there is no additional effort for computing the reference path. The reward function consists of four components: the progress reward $r_p(t)$, the position error reward $r_e(t)$, the action reward $r_a(t)$ and the terminal reward $r_t(t)$.

The progress reward is set to measure the length of the route completed by the quadrotor per step. A greater progress reward means the quadrotor completes more distances in this step. As is not easy to measure the length of the completed route in three dimensions, this paper uses the projection of the quadrotor's position on the line connecting adjacent waypoints as an approximation.



(a) The structure of the Actor neural network



(b) The structure of the Critic neural network

Fig. 5(a), (b). The structures of the Actor neural network and the Critic neural network.

According to Fig. 3(a), at time t, the projection of the quadrotor on the line connecting the adjacent waypoints $s_P(t)$ is defined as follow:

$$s_P(t) = \frac{\left(p_e - p_{e,i}^{des}\right)\cdot\left(p_{e,i+1}^{des} - p_{e,i}^{des}\right)}{\left\|p_{e,i+1}^{des} - p_{e,i}^{des}\right\|} \tag{24}$$

This gives the progress reward for the quadrotor at time t as follow:

$$r_p(t) = s_P(t+1) - s_P(t) \tag{25}$$

The position error reward $r_e(t)$ is used to penalize the position error from the flight path. Because the position error is the input of the PID controllers according to (15), and the PID controller will fail if the input is too large. The position error reward is determined by the positional error of (13):

$$r_e(t) = -\|e_{ct}\| \tag{26}$$

The action reward $r_a(t)$ is used to avoid a large desired flight rate when the quadrotor is close to waypoints or a small desired flight rate when the quadrotor is away from a waypoint. The PID controller outputs the control signal when the error signal appears and it takes some time for the integrator in the PID controller to "ramp up" to eliminate the error signal. However, if the quadrotor is moving too fast when approaching waypoints, the position error increases quickly before the quadrotor slows down and the PID controller will fall due to the large error input. In fact, it is hard for the RL agent to learn to slow down when closing a waypoint, as it has to give up some step rewards to gain a greater total reward, like jumping out of the local maximum to find the global maximum (Kamar et al, 2020). Thus, $r_a(t)$ can reduce the quadrotor crash and accelerate the training process of the RL agent.

At time t, the distance between the projection of the quadrotor and the next waypoint $s_R(t)$ can be calculated as follow according to Fig. 3(a):

$$s_R(t) = \frac{\left(p_{e,i+1}^{des} - p_{e,i}^{des}\right)}{\left\|p_{e,i+1}^{des} - p_{e,i}^{des}\right\|} - s_P(t) \tag{27}$$

Introduce Δ as follow:

$$\Delta = \begin{cases} a_t \cdot t_R - s_R(t), a_t \cdot t_R - s_R(t) \geq 0 \\ a_t \cdot t_R - \min(s_R(t), a_t^u t_R), a_t \cdot t_R - s_R(t) < 0 \end{cases} \tag{28}$$

Where t_R is a parameter for time and a_t^u is the upper bound of the action a_t (which is set to $20\text{m}\cdot\text{s}^{-1}$ in this paper), and $\min(s_R(t), a_t^u t_R)$ limits the range of Δ when $a_t \cdot t_R - s_R(t) < 0$. The action reward $r_a(t)$ is defined as follow:

$$r_a(t) = -1 + 2\cdot e^{-\left(\frac{\Delta}{\sigma}\right)^2} \tag{29}$$

Where σ is a hyperparameter to control the range of $\frac{\Delta}{\sigma}$ so $r_a(t)$ will change significantly while Δ changes. Equation (29) is derived from the equation as follow:

$$y = -1 + 2\cdot e^{-x^2} \tag{30}$$

Equation (30) has a large gradient when $x \in [-2.5, 2.5]$, so, when Δ is away from 0, which means the action a_t is either too big or too small for $s_R(t)$, the action reward $r_a(t)$ will decrease quickly. The action reward is big when $|\Delta|$ is small. The image of (30) is shown in Fig. 6 and can help to understand the action reward $r_a(t)$.

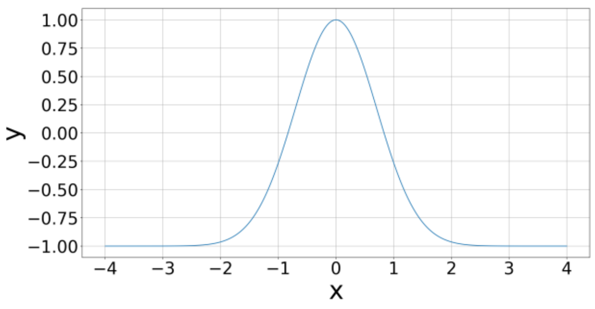


Fig. 6. The image of $y = -1 + 2\cdot e^{-x^2}$.

The terminal reward $r_t(t)$ is used for penalizing the quadrotor crash. The quadrotor crashes when $|e_{ct}|$ is too big to eliminate. $r_t(t)$ makes the quadrotor easier and faster to learn to avoid the crash. The terminal reward $r_t(t)$ can be calculated as follow:

$$r_t(t) = \begin{cases} r_t, if\ crash \\ 0, \quad else \end{cases} \tag{31}$$

Where $r_t < 0$. Combining (25), (26), (29) and (31) gives the reward function as follow:

$$r(t) = a_1 r_p(t) + a_2 r_e(t) + a_3 r_a(t) + r_t(t) \tag{32}$$

a_1, a_2, a_3 are hyperparameters that balance the order of magnitude of different rewards.

5. SIMULATION AND DISCUSSIONS

The quadrotor model is built in the Simulink environment according to the formulas in Section 2 and Section 3, and the DDPG algorithm is implemented by PyTorch. The interaction between Simulink and Python is achieved with the MATLAB Engine API for Python.

A time metric and position error metrics are set in order to describe the control performance.

The time metric t_f is the time it takes for the quadrotor to pass all waypoints in sequence, which is the main goal of the task.

The position error metrics are used to describe the extent to which the quadrotor deviates from the path. During flight, the largest position errors occur when switching waypoints and the quadrotor has the biggest possibility to get out of control. In this paper, the maximum position error when switching waypoint d_m and the cumulative position error during the whole flight d_c are used to describe the local and overall position errors of the quadrotor. A smaller d_m means the quadrotor will be less likely to get out of control. A smaller d_c means the quadrotor will fly more accurately along the path. The cumulative position error d_c is given by:

$$d_c = \int_{t=0}^{t_f} \|e_{ct}\| \, dt (\mathrm{m} \cdot \mathrm{s}) \tag{33}$$

The starting point of the quadrotor is set to $[0,0,0](\mathrm{m})$. The randomly generated set of waypoints is shown in Table 5.

Table 5. The set of waypoints.

No.	Position(m)
1	[244.42,83.55,19.82]
2	[271.74,164.06,19.51]
3	[38.10,287.25,32.68]
4	[274.01,289.47,88.03]
5	[189.71,47.28,47.11]

5.1 Policy Training

The main challenge during training is that the experiences stored in the replay buffer have an uneven distribution, as there are more experiences when the quadrotor is away from waypoints than close to waypoints. Therefore, the RL agent has less chance to learn how to pass a waypoint as the experiences in the replay buffer have equal importance. (Hou et al., 2017) proposed a prioritized replay buffer method using prioritized sampling. In this paper, the replay buffer is divided into two parts, respectively storing the experiences when the quadrotor is far from the waypoints and close to the waypoints. This helps the quadrotor learn to pass waypoints and reduces the training time.

The policy training process with the DDPG algorithm in this paper is shown below.

Algorithm 1 Policy Training with the DDPG algorithm

```
1: Randomly initialize Actor network and Critic network
2: Initialize target networks with the same weights
3: Initialize two replay buffers to store experiences when
   the quadrotor is far from waypoints and close to
   waypoints respectively
4: **for** episode ← 1 to M **do**
5:     Initialize the quadrotor
6:     **for** t ← 1 to T **do**
7:         Compute $a_t$ according to the current actor network
           and exploration noise
8:         Execute $a_t$ in the Simulink environment and
           compute $s_{t+1}$ and $r_t$
9:         Store tuple $(s_t, a_t, r_t, s_{t+1})$ in one of two replay
           buffers according to the distance between the quadrotor
           and waypoints
10:        Sample a random minibatch of $\frac{N}{2}$ in each replay
           buffer
11:        Update the Actor network using (20)
12:        Update the Critic network using (21) and (22)
13:        Update target networks according to the soft
           update strategy
14:    **end for**
15: **end for**
```

(Song et al., 2021) used a parallelized implementation simulating hundreds of quadrotors in parallel to collect up to 25000 environment interactions per second, which was even larger than the memory capacity of the replay buffers in this paper. With the help of the Inner-Outer loop control method, the control instability at the beginning of learning is reduced, and thus reducing the time consumptions and the cost of computation needed for training. It takes 3 hours for training 50 episodes.

The average step reward gained by the RL agent is shown in Fig. 7. In the first four episodes, a_t increased very fast and caused the quadrotor to crash, therefore, the average reward reduced at the first 4 episodes. After that, the quadrotor didn't crash anymore with the training set of waypoints.

The loss curves of the Actor network and the Critic network are shown in Fig. 8.

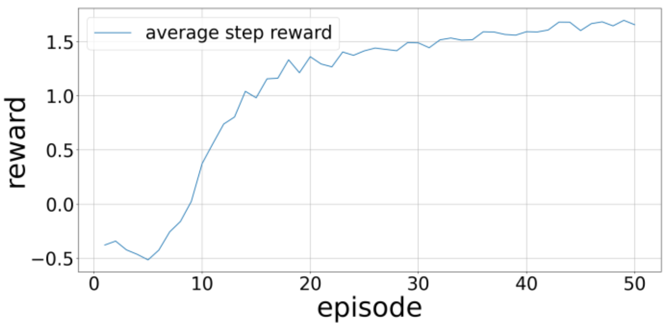
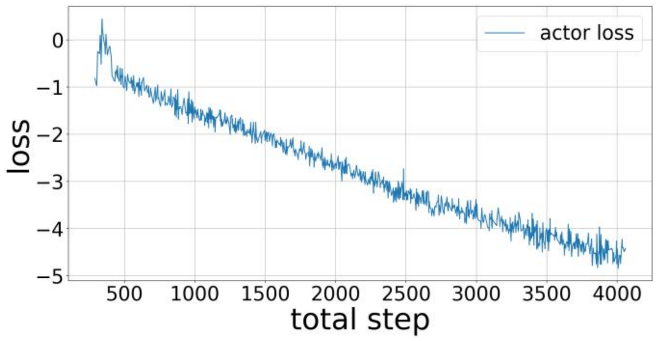
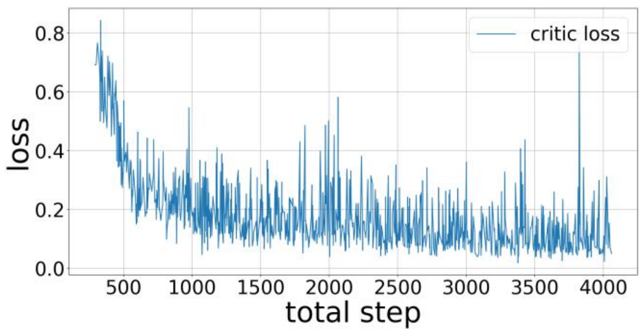


Fig. 7. Average step reward gained by the RL agent.



(a) Loss curve for the Actor network



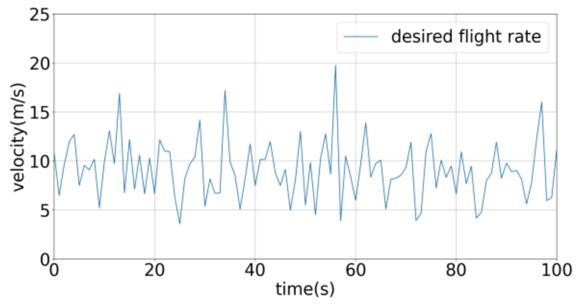
(b) Loss curve for the Critic network

Fig. 8(a), (b). The loss curves of the Actor network and the Critic network.

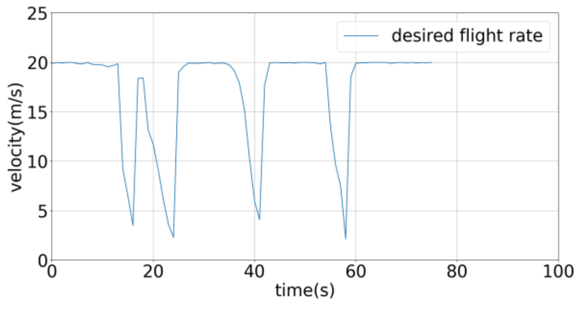
The metrics during training are shown in Table 6. Fig. 9 shows the outputs of the Actor network before and after training. Fig. 10 shows the quadrotor's trajectories after different episodes. This can give an intuitive view of the learning process of the RL agent.

Table 6. The metrics during training.

	$t_f(\mathrm{s})$	$d_m(\mathrm{m})$	$d_c(\mathrm{m \cdot s})$
1 episode	140.14	8.30	139.52
4 episodes		Crash	
30 episodes	76.26	14.36	219.53
50 episodes	74.78	14.06	206.89



(a) a_t before training



(b) a_t after training

Fig. 9(a), (b). The output of the Actor network.

5.2 Training Results

To evaluate the performance of the trained Actor network, this paper introduces three methods as baselines.

Our method is the method proposed in this paper.

Method A is setting the desired flight rate $v^{des} = 13\mathrm{m \cdot s^{-1}}$, which is set as large as possible while finishing the training route in this paper successfully.

Method B is gained from the shape of the output of the Actor network, and is defined by $s_R(t)$ in (27):

$$v^{des} = \begin{cases} 20\mathrm{m \cdot s^{-1}}, s_R(t) \geq 100\mathrm{m} \\ 10\mathrm{m \cdot s^{-1}}, s_R(t) < 100\mathrm{m} \end{cases} \quad (34)$$

Method C is the sliding mode control method based on the backstepping approach (Bouadi et al., 2007).

The metrics of different control methods are shown in Table 7.

Table 7. The metrics of different control methods.

	$t_f(\mathrm{s})$	$d_m(\mathrm{m})$	$d_c(\mathrm{m \cdot s})$
Ours	74.78	14.06	206.89
Method A	102.12	14.66	208.32
Method B	78.40	14.19	208.45
Method C	79.97	5.33	138.63

The quadrotor's real trajectories using different control methods are shown in Fig. 11.

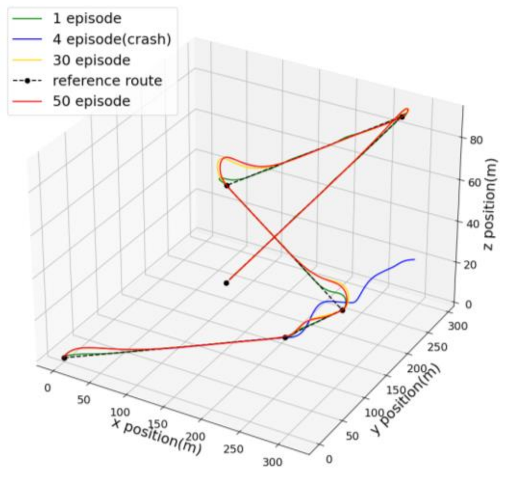


Fig. 10. The quadrotor's trajectories after different episodes.

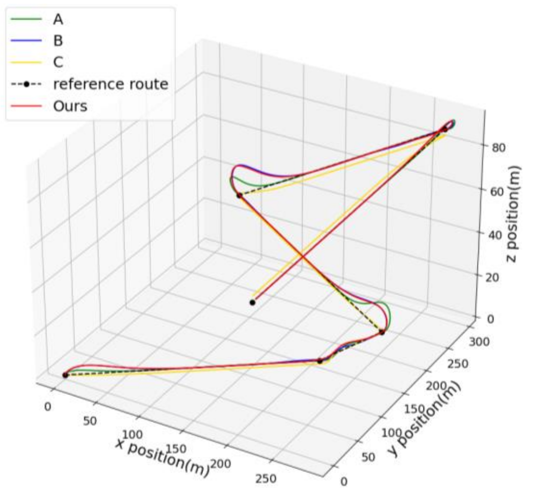
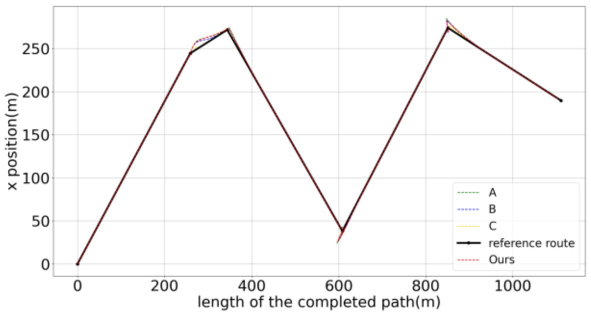


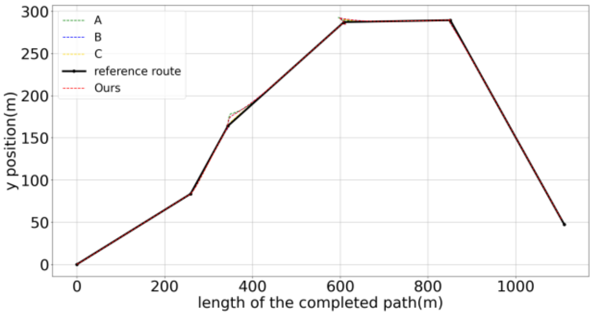
Fig. 11. The quadrotor's trajectories using different control methods.

To compare the performances directly, this paper uses the length of the path completed by the quadrotor as the x axis of the figures, which is equal in different control methods.

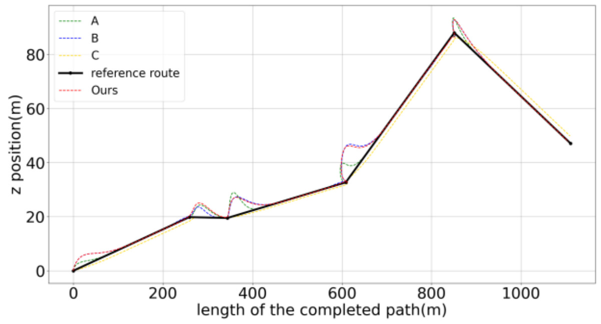
For four control methods, the quadrotor's motions along x, y and z axis are shown in Fig. 12.



(a) the quadrotor's motions along x axis



(b) the quadrotor's motions along y axis



(c) the quadrotor's motions along z axis

Fig. 12(a), (b) and (c). The quadrotor's motions along x, y and z axis.

5.3 Discussions

Method A and method B are Inner-Outer loop PID control methods that are the same as the method proposed in this paper. According to Table 7, the method proposed in this paper outperforms both Method A and method B in terms of all three metrics. This demonstrates that the method proposed in this paper is able to control the desired flight rate of the quadrotor properly and improve the performance of the Inner-Outer loop PID control system. The performance of the method B is similar to the method proposed in this paper, and this is because the output of the Actor network after training in this paper is easy to imitate due to the upper bound on the desired flight rate.

Method C is another common type of control method for quadrotors. According to Table 7, the method proposed in this paper does better in t_f while is not as good as the method C in terms of d_m and d_c. As shown in Fig. 13, the trajectory of the method C presents a permanent tracking error which also occurs in the paper of the method C.

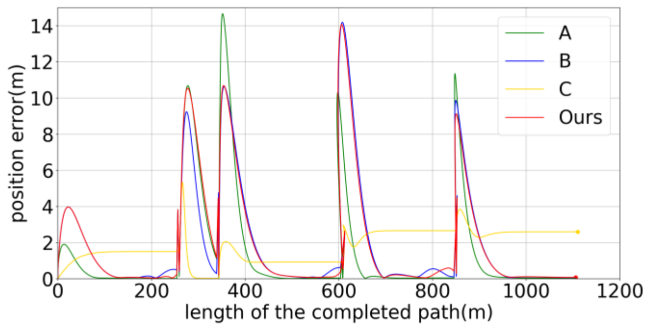


Fig. 13. The position errors of the quadrotor using different control method.

As shown in Fig. 14, the velocity of the quadrotor using the method C changes drastically compared with the quadrotor using the method proposed in this paper, and this leads to higher energy consumption which will reduce the quadrotor's endurance. Additionally, there is chatter when using the method C based on the sliding mode control in Fig. 14 that does damage to the quadrotor and increases the risk of quadrotor crash if applied in reality (Lin et al., 2022). What's more, the proper parameters of the sliding mode control (method C) are more difficult to find than those of the PID control method (method proposed in this paper). Therefore, the method proposed in this paper is more suitable for practical application with less time consumption.

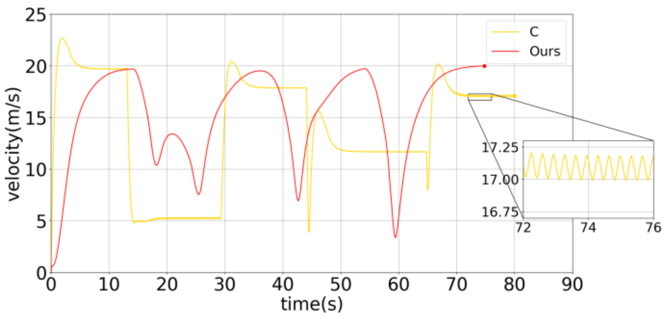


Fig. 14. The speed curves of the quadrotor.

6. CONCLUSIONS

This paper uses neural networks to control the input of the Inner-Outer loop PID control method, the desired flight rate, to accomplish the quadrotor waypoint tracking task. Firstly, this paper presents the Inner-Outer loop PID control method to complete the waypoint tracking task. In order to make the quadrotor adjust its flight rate flexibly, the Actor network is introduced to control the desired flight rate and is trained by the DDPG algorithm. Simulation results prove that the proposed method is able to adjust the desired flight rate successfully and improve the performance of the Inner-Outer loop PID control system.

In future, there is room for improvements of the control system in this paper. Using fractional order PID controllers can improve the performance of PID control (Trivedi and Padhy, 2021). Using more advanced reinforcement algorithms like twin delayed deep deterministic policy gradient algorithm (Fujimoto et al., 2018) and changing the structures of the networks like using multilinear map of state and action as a new input (Long et al., 2018) are possible improvements from a reinforcement learning perspective.

REFERENCES

Bo, G., Xin, L., Hui, Z., and Ling, W. (2016). Quadrotor helicopter Attitude Control using cascade PID. *2016 Chinese Control and Decision Conference*, 5158-5163.

Bouadi, H., Bouchoucha, M., Tadjine, M. (2007). Sliding mode control based on backstepping approach for an UAV type-quadrotor. *World Academy of Science, Engineering and Technology*, 26(5): 22-27.

Cajo, R., Copot, C., Ionescu, C.M., De Keyser, R., and Plaza, D. (2018). Fractional Order PD Path-Following Control of an AR. Drone Quadrotor. *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics*, 291-296.

Cao, N., and Lynch, A. F. (2016). Inner–Outer Loop Control for Quadrotor UAVs With Input and State Constraints. *IEEE Transactions on Control Systems Technology*, vol (24), 5:797-1804.

Farid, G., Mo, H., Zahoor, M.I., and Liwei, Q. (2018). Computationally efficient algorithm to generate a waypoints-based trajectory for a quadrotor UAV. *2018 Chinese Control And Decision Conference*, 4414-4419.

Fujimoto, S., Hoof, H., Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International conference on machine learning*, 1587-1596.

Julkananusart, A., and Nilkhamhang, I. (2015). Quadrotor tuning for attitude control based on double-loop PID controller using fictitious reference iterative tuning (FRIT). *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, 4865-4870.

Hoffmann, G., Waslander, S., and Tomlin, C. (2008). Quadrotor helicopter trajectory tracking control. *AIAA guidance, navigation and control conference and exhibit*, 7410.

Hou, Y., Liu, L., Wei, Q., Xu, X., and Chen, C. (2017). A novel DDPG method with prioritized experience replay. *2017 IEEE International Conference on Systems, Man, and Cybernetics*, 316-321.

Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. (2017). Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4), 2096-2103.

Kamar, D., Akyol, G., Mertan, a., and İnceoğlu, A. (2020). Comparative Analysis of Reinforcement Learning Algorithms on TORCS Environment. *2020 28th Signal Processing and Communications Applications Conference*, 1-4.

Kumar, V. B., Sampath, D., Siva Praneeth, V. N., and Pavan Kumar, Y. V. (2021). Error Performance Index Based PID Tuning Methods for Temperature Control of Heat Exchanger System. *2021 IEEE International IOT, Electronics and Mechatronics Conference*, 1-6.

L'Afflitto, A., Anderson, R. B., and Mohammadi, K. (2018). An Introduction to Nonlinear Robust Control for Unmanned Quadrotor Aircraft: How to Design Control Algorithms for Quadrotors Using Sliding Mode Control and Adaptive Control Techniques [Focus on Education]. *IEEE Control Systems Magazine*, 38(3), 102-121.

Lambert, N.O., Drew, D.S., Yaconelli, J., Levine, S., Calandra, R., and Pister, K.S.J. (2019). Low-Level Control of a Quadrotor With Deep Model-Based Reinforcement Learning. *IEEE Robotics and Automation Letters*, 4(4), 4224-4230.

Larsson, D.T., Nguyen, C. H., and Artemiadis, P. (2020). Modeling and Control of Mid-flight Coupling of Quadrotors: A new concept for Quadrotor cooperation. *2020 International Conference on Unmanned Aircraft Systems*, 310-315.

Li, X., Qi, G., Guo, X., and Ma, S. (2020). Trajectory Tracking of a Quadrotor UAV based on High-Order Differential Feedback Control. *2020 IEEE 9th Data Driven Control and Learning Systems Conference*, 201-206.

Li, Z., Ma, H., Ding, Y., Wang, C., and Jin, Y. (2020). Motion Planning of Six-DOF Arm Robot Based on Improved DDPG Algorithm. *2020 39th Chinese Control Conference*, 3954-3959.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., et al. (2015). Continuous control with deep reinforcement learning. *ArXiv*, 1509.02971.

Lin, C.H., Ho, C.W., Hu, G.H., Sreeramaneni, B., Yan, J.J. (2022). Robust chaos suppression of uncertain unified chaotic systems based on chattering-free sliding mode control. *Measurement and Control*. 55(5-6):321-329.

Long, M., Cao, Z., Wang, J., et al. (2018). Conditional adversarial domain adaptation. *Advances in neural information processing systems*, 31.

Mohamad Ali Tousi, S., Mostafanasab, A., and Teshnehlab, M. (2020). Design of Self Tuning PID Controller Based on Competitional PSO. *2020 4th Conference on Swarm Intelligence and Evolutionary Computation*, 22-26.

Mahmud, M., Motakabber, S.M.A., Zahirul Alam, A.H.M., and Nordin, A.N. (2020). Adaptive PID Controller Using for Speed Control of the BLDC Motor. *2020 IEEE International Conference on Semiconductor Electronics*, 168-171.

Parivash, F., and Ghasemi, A. (2017). Trajectory tracking control for a quadrotor using fuzzy PID control scheme. *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation*, 0553-0558.

Penicka, R., and Scaramuzza, D. (2022). Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments. *IEEE Robotics and Automation Letters*, 7(2), 5719-5726.

Qian, G.M., Pebrianti, D., Chun, Y.W., Hao, Y.H., and Bayuaji, L. (2017). Waypoint navigation of quad-rotor MAV. *2017 7th IEEE International Conference on System Engineering and Technology*, 38-42.

Rodriguez-Ramos, A., Sampedro, C., Bavle, H., Puente, P., and Campoy, P. (2019). A deep reinforcement learning strategy for uav autonomous landing on a moving platform. *Journal of Intelligent & Robotic Systems*, 93(1), 351-366.

Rubí, B., Morcego, B., and Pérez, R. (2020). A Deep Reinforcement Learning Approach for Path Following on a Quadrotor. *2020 European Control Conference*, 1092-1098.

Saraf, P., Gupta, M., and Parimi, a.m. (2020). A Comparative Study Between a Classical and Optimal Controller for a Quadrotor. *2020 IEEE 17th India Council International Conference*, 1-6.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *ArXiv*, 1707.06347.

Song, W., Li, Z., Xu, B., Wang S., and Meng, X. (2022). Research on Improved Control Algorithm of Quadrotor UAV based on Fuzzy PID. *2022 IEEE International Conference on Artificial Intelligence and Computer Applications*, 361-365.

Song, Y., Steinweg, M., Kaufmann, E., and Scaramuzza, D. (2021). Autonomous Drone Racing with Deep Reinforcement Learning. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1205-1212.

Trivedi, R., and Padhy, P.K. (2021). Design of Indirect Fractional Order IMC Controller for Fractional Order Processes. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(3): 968-972.

Zhang, Q., Tang, R., Gou, S., and Zhang, W. (2020). A PID Gain Adjustment Scheme Based on Reinforcement Learning Algorithm for a Quadrotor. *2020 39th Chinese Control Conference*, 6756-6761.

Zhang, M., Zhang, Y., Gao, Z., and He, X. (2020). An Improved DDPG and Its Application Based on the Double-Layer BP Neural Network. *IEEE Access*, 8, 177734-177744.

Zhang, X., Li, X., Wang, K., et al. (2014). A survey of modelling and identification of quadrotor robot. *Abstract and Applied Analysis*, vol (2014).