drones

Article

Model-Free Guidance Method for Drones in Complex Environments Using Direct Policy Exploration and Optimization

Hongxun Liu and Satoshi Suzuki *

School of Science and Engineering, Chiba University, 1-33 Yayoi-cho, Inage-ku, Chiba 263-8522, Japan; 21wd4105@student.gs.chiba-u.jp
* Correspondence: suzuki-s@chiba-u.jp

Abstract: In the past few decades, drones have become lighter, with longer hang times, and exhibit more agile performance. To maximize their capabilities during flights in complex environments, researchers have proposed various model-based perception, planning, and control methods aimed at decomposing the problem into modules and collaboratively accomplishing the task in a sequential manner. However, in practical environments, it is extremely difficult to model both the drones and their environments, with very few existing model-based methods. In this study, we propose a novel model-free reinforcement-learning-based method that can learn the optimal planning and control policy from experienced flight data. During the training phase, the policy considers the complete state of the drones and environmental information as inputs. It then self-optimizes based on a predefined reward function. In practical implementations, the policy takes inputs from onboard and external sensors and outputs optimal control commands to low-level velocity controllers in an end-to-end manner. By capitalizing on this property, the planning and control policy can be improved without the need for an accurate system model and can drive drones to traverse complex environments at high speeds. The policy was trained and tested in a simulator, as well as in real-world flight experiments, demonstrating its practical applicability. The results show that this model-free method can learn to fly effectively and that it holds great potential to handle different tasks and environments.

Keywords: drones; reinforcement learning; policy optimization; model-free; traverse complex environments

check for updates

Citation: Liu, H.; Suzuki, S. Model-Free Guidance Method for Drones in Complex Environments Using Direct Policy Exploration and Optimization. *Drones* **2023**, *7*, 514. https://doi.org/10.3390/drones7080514

Academic Editor: Vaios Lappas

Received: 10 July 2023
Revised: 29 July 2023
Accepted: 1 August 2023
Published: 3 August 2023

1. Introduction

The recent advancements in drone technology have resulted in their widespread adoption across various industries worldwide. Drones offer favorable properties that make them highly suitable for complex mission environments, such as search and rescue operations in forests [1–3] and bridge inspections [4,5]. These environments often present challenges such as cluttered obstacles and weak communication and satellite signals. In search and rescue (SAR) tasks, it is crucial for drones to navigate around obstacles efficiently and reach their destinations quickly. In this study, we focus on the navigation problem of drones operating in unknown and complex environments, where they encounter various obstacles and reach their destinations quickly and safely. To maximize the maneuverability of drones while ensuring flight safety, researchers typically divide the controller into two layers: a low-level angular rate and attitude layer, analogous to the human epencephalon; and a high-level velocity, position, and navigation layer, resembling the human brain [6–8]. Among these parameters, the low-level control problem has been extensively analyzed, and accordingly, optimal solutions have been established. However, some issues still persist in high-level control methods, hindering the effective utilization of the high maneuverability of drones.

Based on extensive research, several factors have been identified as reasons for the improper utilization of the high maneuverability of drones. A recent study identified the latency of perception sensors, such as monocular and stereo cameras, as a contributing parameter [9]. It discussed the effect of perception latency in high-speed movement robot platforms. The study proposed a general analysis method to evaluate the maximum latency when operating the robot at the ultimate velocity. It highlighted that simply reducing the longitudinal velocity does not necessarily guarantee the security of the robot. However, this method can only adjust the latency tolerance, and the suggested solution is to use more advanced sensors, such as event cameras. In contrast, reducing the latency of software and algorithms is a more economical approach. Another study introduced a low-level multi-sensor fusion algorithm, where vision and three-dimensional (3D) lidar data were segmented [10]. Here, the core control system received and fused their results to reduce latency.

In addition to the factors mentioned above, the issue of coordination after modularization is a significant challenge. Traditional navigation pipelines for drones in unknown environments are typically divided based on perception, path planning, and control subtasks to solve the autonomous flight problem in a step-by-step manner [11–16]. A recent study proposed visual stereo simultaneous localization and mapping (SLAM) for drones, which accelerates the mapping process and thus provide an approach for map maintenance with a uniform distribution [17]. The problem of high-quality 3D mapping was addressed using only onboard, low-computing consumption, and imperfect measurements [18–20]. Some researchers focused on path planning using length analysis, search-based, and Euclidean signed distance field methods [21–23]. The control subtask has a long history, and there are many well-known and effective algorithms, such as PID, H-infinite, sliding mode control, and model predictive control (MPC) [24]. These methods enable drones to accomplish the tasks mentioned above through a parallel cooperation pipeline of components. Thus, it makes the entire system interpretable and facilitates the tuning of the parameters of each component. Although it may seem attractive from an engineering implementation perspective, due to the sequential nature, the communication latency between components and wastes computing resources on unnecessary functions is overlooked, leading to difficulties in achieving agile flight.

Thus, we propose a new guidance framework with a reinforcement-learning-based direct policy exploration and optimization method to maximize the maneuverability of drones and enhance flight speed in complex environments. In this framework, the low-level angular rate, attitude, and velocity are controlled by a well-tuned cascade PID, while the high-level path planning, position controller, and trajectory tracking controllers are merged into a control policy. The policy is a fully connected three-layer neural network that takes sensor measurements as inputs and outputs the distribution of the target velocity commands. Finally, the real control commands are sampled from the distribution and sent to a low-level controller, as shown in Figure 1. The main contributions of this study are summarized as follows:

1. A novel navigation framework is proposed to improve the flight speed of drones in complex environments. The framework integrates an RL-based policy to replace the traditional path planning and trajectory tracking controllers in a conventional drone guidance pipeline. The experimental outcomes in both the simulated and real-world environments show that it can push the maneuverability of drones to the physical limit.
2. A novel direct policy exploration and optimization method is proposed, which eliminates the need for a teacher–student structure commonly used in imitation learning and does not require reference trajectories. To train the policy, multiple environments (Figure 2) with varying obstacle distribution characteristics were created. Subsequently, the well-trained policy is directly applied to real-world drones for practical applications.
3. From a practical perspective, we designed a hybrid reward function to facilitate policy exploration and self-improvement. This function further comprises multiple continuous and discrete functions that can effectively evaluate the distance between the drone and the destination, energy consumption, collision risk with obstacles, and other such factors.

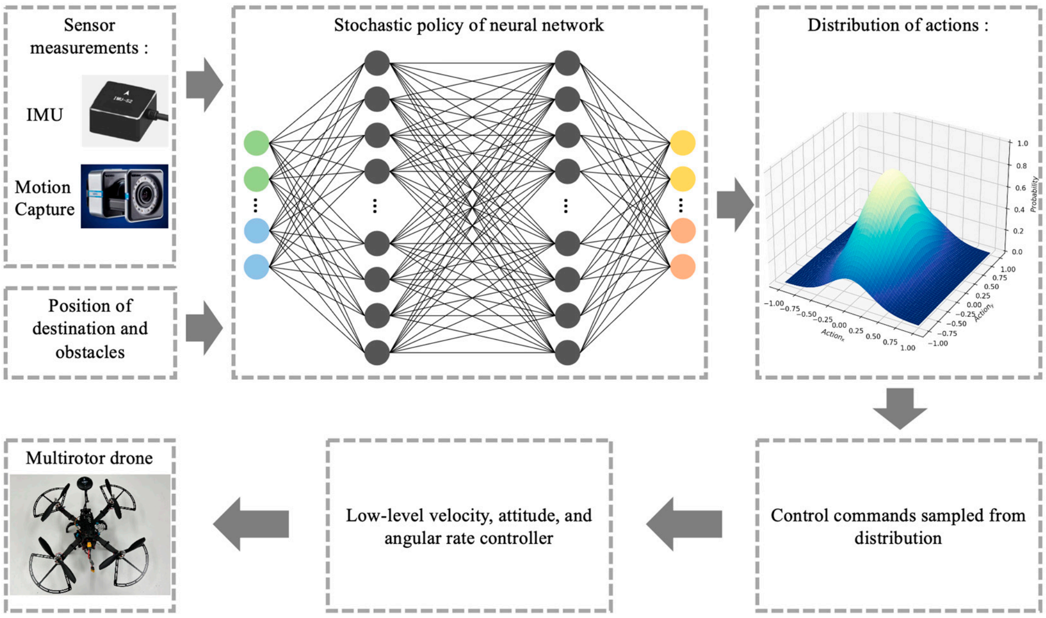

Sensor measurements :
IMU
Motion Capture
Position of destination and obstacles
Stochastic policy of neural network
Distribution of actions :
Probability
Action₁
Action₂
Control commands sampled from distribution
Low-level velocity, attitude, and angular rate controller
Multirotor drone

Figure 1. Proposed RL controller.

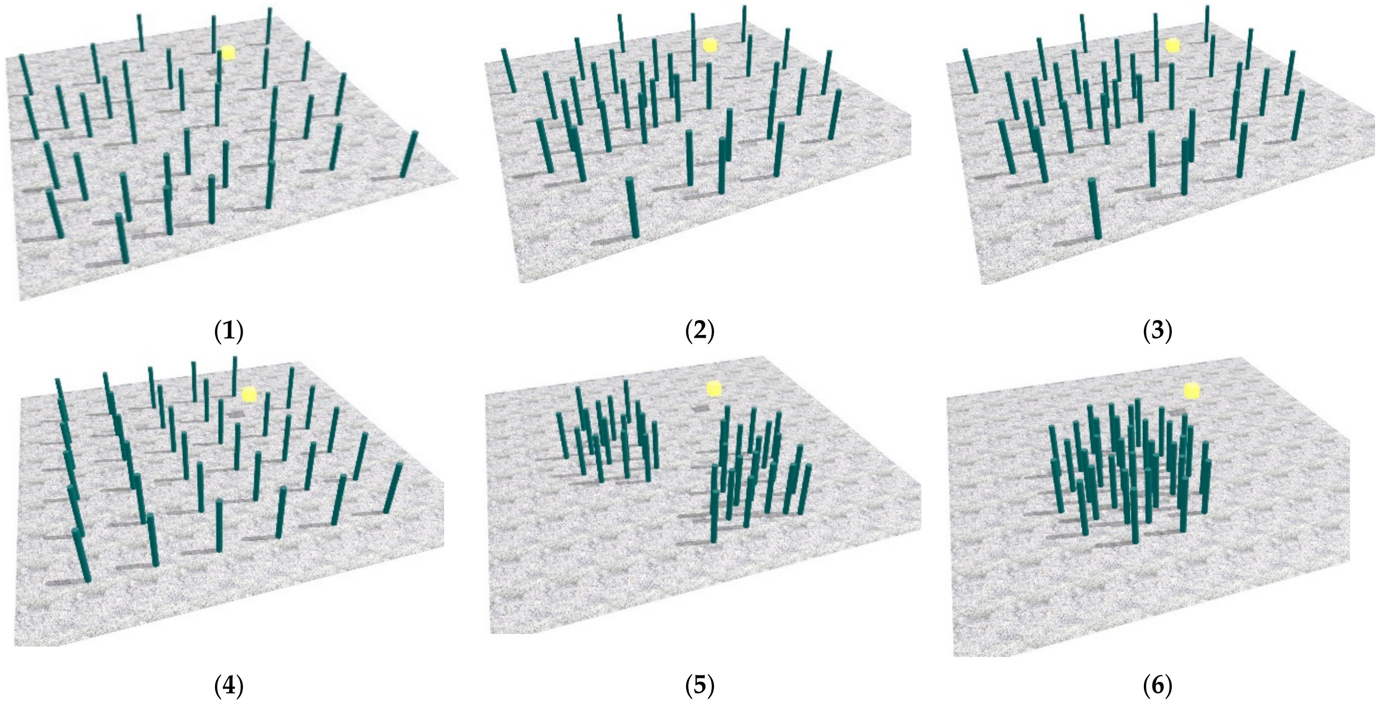


Figure 2. Part of designed clutter environments. Yellow boxes indicate destination. Drones takeoff on left bottom of floor. (**1**) has randomly distributed obstacles, (**2**,**3**) have designed routes to traverse. (**4**) has neatly arranged obstacles, while (**5**,**6**) have clusters of obstacles.

The remainder of this paper is structured as follows. Section 2 discusses the classical perception–planning–control pipeline and the latest RL works. Section 3 describes the low-level model-based attitude controller and high-level planning-tracking strategy in our approach, along with the designed reward function and the algorithm for direct policy exploration and optimization. Section 4 shows the experiments that were conducted to train and test the policy of the proposed structure in a simulator. We also applied this method directly to real-world drones and analyzed the observed advantages and limitations. Finally, the conclusions, opportunities, and challenges of future work are discussed in Section 5.

2. Related Works

2.1. Conventional Sequential Method

Flying drones in complex environments is a challenging task. The traditional approach involves breaking it down into specific problems, such as perception, path planning, and control, addressing them separately, and then arranging them in a sequential pipeline, as shown in Figure 3.

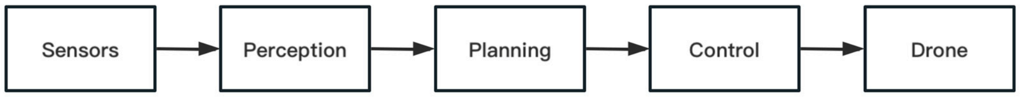


Figure 3. Pipeline of conventional method.

The main task of the perception module is to estimate the state of the drone and its surrounding environment using onboard or external sensor data. A few common and well-known perception algorithms include the SLAM and the visual–inertial odometry (VIO). The planning module generates a feasible and time-optimal trajectory that helps the drone accomplish its mission based on the state, environmental information, and task requirements. In a recent study, this module was classified into four types—sampling-based, search-based, optimization-based, and polynomial and spline trajectory-based. A control module typically operates at higher frequencies. This module calculates the corresponding control commands that enable the drone to follow the intended trajectory based on the state of the drone and target trajectory. Algorithms such as MPC can even predict the future states of a drone based on the motion model to search for optimal control inputs over the entire time window.

Although the conventional pipeline has enabled the autonomous flight of drones in complex environments, a few challenges still persist. For instance, although the VIO algorithms are lightweight, they require considerable computational resources. Planning algorithms may be affected by factors such as unmodeled system delays, state estimation drift, and discontinuities in perception modules, and this may result in suboptimal paths or even mission failures.

2.2. Reinforcement Learning

Reinforcement learning (RL) aims to learn how to make optimal decisions and maximize cumulative rewards through the interaction between an agent and its environment. Based on the optimization objective, RL can be divided into two types: value-based and policy-based methods. Different from the well-known deep learning, RL can update the knowledge of agents through trial and exploration in the world (a widespread structure like that in Figure 3) without a supervisor; this characteristic endows RL with vast potential to outperform human experts in computer games [25,26]. The primitive table-based RL algorithm can only deal with problems that only have discrete, limited states and actions. After this decade of developments, especially in relation to the combination of neural network technique, the RL can handle complex and nonlinear systems which have continuous states and actions; it is the foundation of applying RL to control and navigation tasks [27].

In recent years, learning-based methods have been extensively used in perception [28], planning [14], and control [29]. The neural networks can handle high- and low-dimensional information and are easy to deploy. Researchers have successfully replaced certain modules in classical pipelines with learning-based approaches, leading to significant advancements. The RL can update the knowledge of agents through trial and exploration worldwide (a common structure is shown in Figure 4) without a supervisor. Driven by this characteristic, it may potentially surpass human expert performance [30].

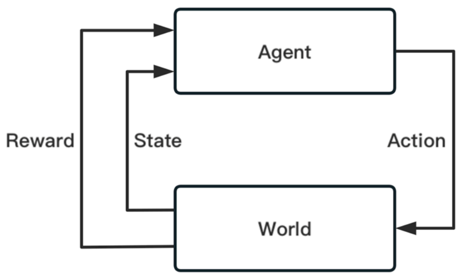


Figure 4. RL structure.

There are many variations in RL and value-based methods, including Q-learning, deep Q-network, policy-based deterministic policy gradient, trust region policy optimization [31], and proximal policy optimization (PPO) [32]. As shown in Figure 4, the agent interacts with the world over time, receives a state s_t in the state space S at timestep t, and selects an action a_t from the action space A; the mapping relationship from s_t to a_t is the policy $\pi(s_t|a_t)$. The world will transfer to a new state s_{t+1}, submit to the state transition probability $P(s_{t+1}|s_t, a_t)$, and present the agent with a scalar reward r_t according to the reward function $R(s, a)$, which is affected by the model of the system or environment dynamics. After a complete interaction process (one episode) between the agent and the environment, a series of observations, referred to as trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots)$, may be obtained. We assume that this series process satisfies the Markov property $P(S_{t+1} = s_{t+1}|S_{0:t} = s_{0:t}) = P(S_{t+1} = s_{t+1}|S_t = s_t)$, which means that for a stochastic process, given the current state and all past states, the conditional probability distribution of its future states depends only on the current state.

2.3. Value-Based Methods

During an episode of continuous state transition and action execution, an accumulated reward (referred to as return), with the discount factor $\gamma \in (0, 1]$, was defined as:

$$R_t = \sum_{j=0}^{\infty} \gamma^k r_{t+j} \tag{1}$$

The return can evaluate whether the entire process is good and helps the agent improve by maximizing the expectation of R_t. When a system model or state transition function is available, the optimization problem may be solved through dynamic programming. However, this is usually attainable. In this case, the complete trajectory cannot be obtained through model iterations, and the RL possesses the ability to solve this problem through trial and error. We define a value function to measure the value of each state, which is the expectation of the returns of trajectories starting from the current state and following policy π:

$$V_\pi(s) = E[R_t|s_t = s] \tag{2}$$

The action–value function for action a in state s following the policy π:

$$Q_\pi(s, a) = E[R_t|s_t = s, a_t = a] \tag{3}$$

The Bellman equation can be obtained by decomposing the expectation in V_π:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_\pi(s')\right] \tag{4}$$

The optimal value function is the maximum value of the current state s following an arbitrary policy:

$$V_*(s) = \underset{a}{max} \sum_{s',r} p(s', r|s, a) \left[r + \gamma V_*(s')\right] \tag{5}$$

The optimal action value function is:

$$Q_*(s,a) = \sum_{s',r} p(s',r|s,a)\left[r + \gamma \underset{a'}{max} Q_*(s',a')\right] \tag{6}$$

The intuitive solution to the RL problem is to find an optimal value function or optimal action–value function and always choose the action that motivates the system to transition to optimal states that can maximize the return of the trajectory. By estimating and updating these value functions, the agent can accurately evaluate the values of different states and actions and thus make optimal decisions through continuous learning. This approach is commonly referred to as value-based reinforcement learning. It also yields good results in application-based scenarios, such as aerial manipulator control [33], robot path planning [34], and lane following of unmanned cars [35]. However, as stated by Zhang in [36,37], the value-based method yields low training efficiency and data sufficiency; it requires several tricks and modifications to improve training speed and stability in practical and complex scenarios. We analyzed the reasons for this problem. The goal of the value-based method is to determine an optimal value function that defines the mapping relationship between actions and states once the algorithm converges. Each state has a corresponding action. However, the optimality of the action is questionable when the environment changes slightly. This is more apparent when the state and action spaces become significantly vast because the value function is unavailable for approximation.

2.4. Policy-Based Methods

Contrary to the approach of searching for an optimal value function, policy-based methods can directly optimize the policy $\pi(a|s;\theta)$ to enable it to choose the correct action in various states. Here, θ can be viewed as a parameter of a function approximation, and updating it through gradient ascent can improve the policy. This method usually performs better than value-based methods, particularly in complex scenarios with high-dimensional states and action spaces, and it possesses better convergence properties. The primary benefit of the policy-based method is that it can use a stochastic policy that exploits exploration. The value-based method has a deterministic policy that always chooses an action that can maximize the value function. This method only improves policy implicitly and does not possess exploration. Researchers usually solve this using ϵ-greedy exploration, which is insufficient and introduces extra hyperparameters. By assuming that the policy $\pi(a|s;\theta)$ is derivable and has analytical gradient, we obtain:

$$\nabla_\theta \pi(a|s;\theta) = \pi(a|s;\theta)\frac{\nabla_\theta \pi(a|s;\theta)}{\pi(a|s;\theta)} = \pi(a|s;\theta)\nabla_\theta \log\pi(a|s;\theta) \tag{7}$$

where, $\nabla_\theta \log\pi(a|s;\theta)$ is the likelihood ratio or score function. The policy gradient is defined as:

$$\nabla J_\theta = E_{\pi,\theta}[\nabla_\theta \log\pi(a|s;\theta) Q_{\pi,\theta}(s,a)] \tag{8}$$

We update policy parameters θ in k iteration step along the policy gradient. Here, α is the learning rate:

$$\theta_{k+1} = \theta_k + \alpha \nabla J_\theta \tag{9}$$

3. Method

As mentioned previously, we propose a new framework for guiding drones in complex environments. In the Section 1, we introduce a model-based attitude controller. The Section 2 presents the proposed method of direct policy exploration and optimization, which replaces the planning and trajectory-tracking modules in a classical pipeline. The Section 3 describes the designed reward function, which provides an effective evaluation criterion for policy optimization.

3.1. Low-Level Controller

Drones are typically agile, and they possess a six-dimensional motion capability, 3D position (denoted by $Pos = [x, y, z]^T$), and 3D orientation (denoted by $Rot = [\alpha, \beta, \psi]^T$), as shown in Figure 5. In our method, the low-level controller keeps the drone stable and agile, and precisely executes velocity commands from the high-level RL controller.

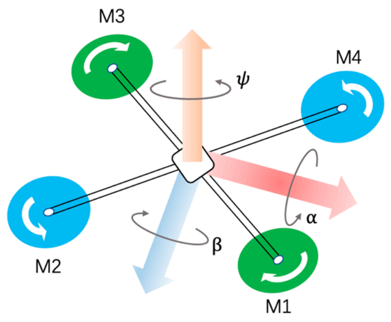


Figure 5. Drone model.

We proposed a cascaded control architecture that combines P and PID controllers with different loop frequencies to achieve a low-level control task. As shown in Figure 6, the velocity setpoint V_{sp} is input to the outer loop velocity control layer, which then outputs the expected acceleration A_{sp} to the compensator and distributor layer. This layer is not a controller but a simple function of $R_{sp} = \arctan\left(\frac{A_{sp}}{g}\right)$, where g is the acceleration of gravity, which is a simple decomposition of forces; it will not be further elaborated here. The necessity of this layer is that the thrust of the motors is limited, and our goal is to maximize the maneuverability of drones within the allowed range of power performance. The maximum horizontal acceleration and maximum target attitude angle should be restricted within a reasonable range to avoid crashing due to insufficient power.

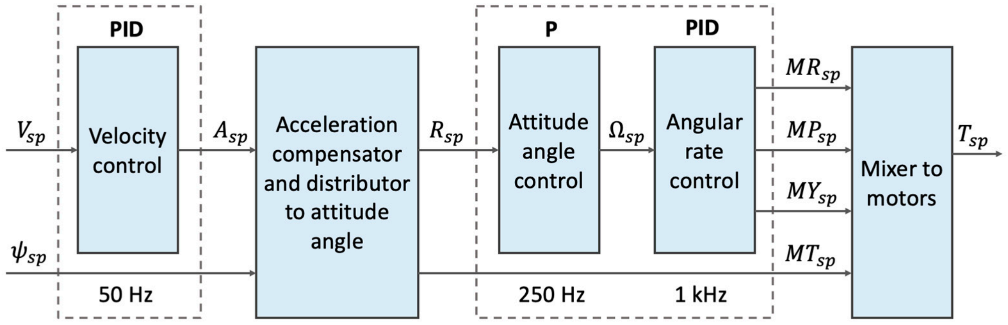


Figure 6. Complete cascaded low-level controller combining P and PID controllers in several control layers.

Then, the attitude angle setpoint R_{sp} is input to the cascaded inner loop attitude angle and angular rate controller, which can output the motor thrust in the roll, pitch, and yaw directions, denoted by MR_{sp}, MP_{sp}, and MY_{sp}, respectively. These three setpoints, along with the throttle setpoint MT_{sp}, are decomposed into four rotation speed setpoints T_{sp} according to Equation (10):

$$T_{sp}(M1, M2, M3, M4)^T = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} MT_{sp} \\ MR_{sp} \\ MP_{sp} \\ MY_{sp} \end{bmatrix} \tag{10}$$

Notably, the loop frequency of each controller in cascade control is different. The inner loop controllers for the attitude angle and angular rate operate at 250 and 1 kHz, respectively. This is because the system may respond faster to the angular rate than to the attitude angle. In contrast, the outer loop controller for the velocity operates at 50 Hz. This is because the velocity is indirectly controlled by changing the attitude angle and thus has a slower response.

3.2. High-Level Policy

As mentioned previously, we aim to fundamentally solve the issue of high latency in pipelines based on environmental perception through path planning for drones flying in complex environments, which further helps maximize their agile mobility. We utilized the PPO algorithm to handle the information gathered from both onboard and external sensors concerning the environment and drone states in parallel. The algorithm generates control commands that are fed directly to the low-level controller in an end-to-end scheme. The policy training framework is illustrated in Figure 7. The figure shows a partial set of state variables for the drone and its environment, obtained through a combination of onboard and external sensors.

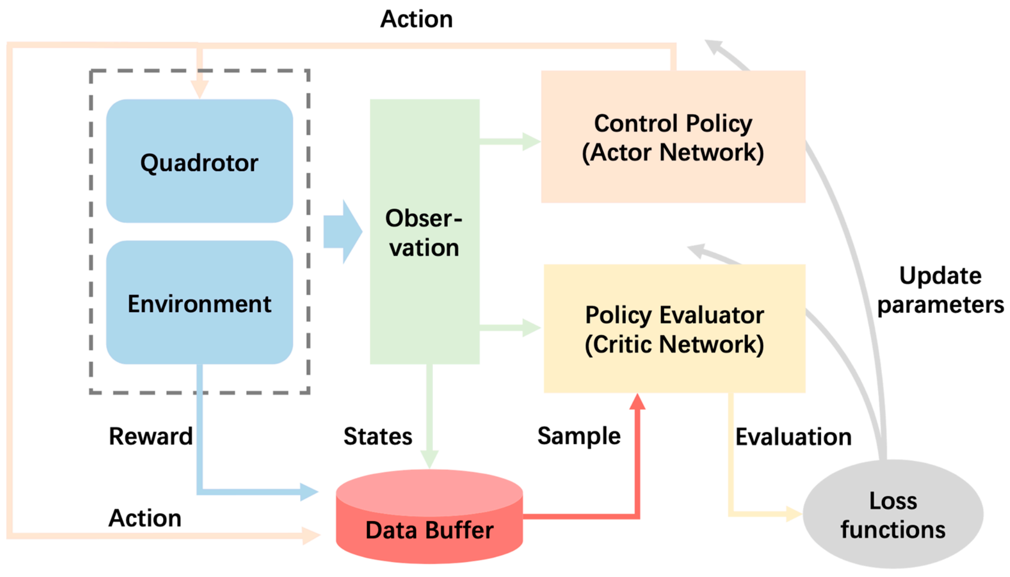


Figure 7. Policy training framework.

The framework consists of two neural networks. The first one is the actor network with parameters θ, which is evaluated to find an approximate optimal control policy, which provides a control command (referred to as an action in RL) to the low-level controller end-to-end based on the observation. At the beginning of the training process, the actions are chosen randomly, the drone transfers to a new state after the actions take effect, and the environment provides a reward that indicates the value of the actions. We refer to each complete navigation task process as an episode, with the sequence of states, actions, and rewards obtained during the process referred to as the trajectory τ. After several episodes, all trajectory data were stored in data buffer D, and these data were periodically sampled and used to calculate the value of each state and action based on Equation (1).

Conversely, the value function $V_{\phi k}$ from Equation (2) reveals the value of a certain state, s, on the k–th iteration, which is obtained through sampling and calculation from the data buffer. The action–value function Q^{π} from Equation (3) represents the value of a certain action in that state, which is approximated using a critic network with the parameter ϕ. The advantage function representing the advantage of action a in state s can then be obtained as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V_{\phi k}(s) \tag{11}$$

For the actor and critic neural networks, we used two different loss functions to calculate the parameter gradients and used stochastic gradient ascent to optimize their performance. The actor network parameters are updated by maximizing the PPO clip objective.

$$\theta_{k+1} = \arg \underset{\theta}{max} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g\left(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)\right) \right) \tag{12}$$

The set of trajectories collected by applying the control policy $\pi_k = \pi(\theta_k)$ in the training environment is denoted as $D_k = \{\tau_i\}$. Here, T represents the time step length of a trajectory randomly sampled from D_k. Equation (12) shows two surrogate losses, $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$. $A^{\pi_{\theta_k}}(s_t, a_t)$ indicates the step length of parameter updates along the policy gradient, but it will lead to an unexpectedly large update of policy [32]. To ensure a stable and convergent policy optimization, policy updates must be constrained within a reasonable range. The PPO introduced a clip function $g\left(\epsilon, A^{\pi}_{\theta_k}(s_t, a_t)\right)$ with a hyperparameter ϵ to limit the step length of policy update:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases} \tag{13}$$

In contrast to the actor network, the parameter update of the critic network uses a gradient descent, and the objective function is the mean-squared error:

$$\phi_{k+1} = \arg \underset{\phi}{min} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2 \tag{14}$$

The $\hat{R}_t$ is the discount accumulated reward from current state s_t to the end of episode, and it can be calculated according to Equation (1) on the trajectory τ, which was sampled from the dataset D_k. Here, $V_\phi(s_t)$ is the evaluation of current state s_t predicted using critic network with current parameters ϕ_k.

We then obtain the learning algorithm by synchronously optimizing the actor and critic networks stepwise:

3.3. Environment Setting and Reward Function

As shown in Figure 2, there are varying numbers of cylindrical obstacles in the envisioned environments. The goal of this study is to enable drones to traverse cluttered environments safely and quickly. Without any mechanical protection, it may crash after collision with an obstacle. Further, without protective measures, the delicate and sensitive components of a drone, such as its propellers and motors, are vulnerable to damage from collisions with obstacles.

The reward function is the criterion for measuring the quality of a single step of an action, and both the value function V and the action–value function are Q built on this basis. Therefore, the design of reward function must enable the control policy to avoid obstacles and fly to the destination as quickly as possible. Because our learning algorithm uses a clip function and the magnitude of a single policy update is constrained by a hyperparameter ϵ, the reward function should be as continuous and smooth as possible in order to ensure stable convergence and avoid local optima. In accordance with these requirements, we designed a composite reward function with five components and their weights $\mathcal{W}$ as follows:

$$r_t = \mathcal{W} * \begin{pmatrix} \Delta\, D\, i\, s_t \\ E\_Cost_t \\ |Over\_speed| \\ Symbol(collide\ or\ arrive) \\ Soft\_constrains \end{pmatrix} \tag{15}$$

The first term $\Delta Dis_t = Dis_{t-1} - Dis_t$ is a continuous reward, which presents a positive reward when the drone approaches the goal and a negative reward otherwise, and Dis_t is the distance between the drone and the destination. The physical interpretation of the second term, $E\_Cost_t = |U_t^2 - U_{t-1}^2|$, is the energy consumed by the current action. Because the control input U_t is the flight velocity of the drone and $\frac{mU^2}{2}$ is the kinetic energy of the drone, we moved the constant parameter $\frac{m}{2}$ into weights for conciseness. This term can reduce meaningless actions, determine the shortest path, and suppress the action noise of a drone during a mission. The third term (Equation (16)) is a soft constraint on the flight speed. In reality, the ability of the drones is restricted due to the performance of their power systems and maneuverability. Long-term high loads may lead to serious accidents such as motor failures, which will provide a negative reward when the drone exceeds the Max_speed, to relieve the burden on the power system.

$$|Over\_speed| = \begin{cases} 0, & U_t \le Max\_speed \\ |U_t - Max\_speed|, & U_t > Max\_speed \end{cases} \tag{16}$$

The fourth and final terms were designed to address sparse rewards in specific scenarios. When a drone collides with an obstacle, the symbol function provides a large negative reward as a punishment, whereas reaching the destination provides an extra positive reward. Further, the $Soft\_constrains$ can add points of interest or avoid additional dangerous areas in the task.

4. Experiment

This chapter is divided into four parts to provide a detailed explanation of the experimental results. Section 4.1 showcases the configuration of the simulator environment, in addition to the method and parameters used for policy training. Section 4.2 presents the training results, where the average reward demonstrates the convergence of the algorithm in all the designed environments. Sections 4.3 and 4.4 describe the performance of the trained controller in a simulator and the real world, respectively. The results indicate that the proposed algorithm performs well in different environments, with both the average and maximum flight speeds surpassing those of the PID algorithm.

4.1. Policy Training Frame in Simulator

As described in Algorithm 1, the RL training process involves interacting with the environment based on the current policy, accumulating experience data, calculating the advantage function and the gradient of the policy, and updating the parameters of the actor and critic networks—with this process repeated iteratively. To improve the efficiency of collecting the experience data and ensure the safety of the interaction process, we used the Gazebo system to simulate the state of a drone in the real world. We used the model (Figure 8) from [38], which was designed for simulation in a robot operation system (ROS) and Gazebo, whose frame parameters are listed in Table 1.

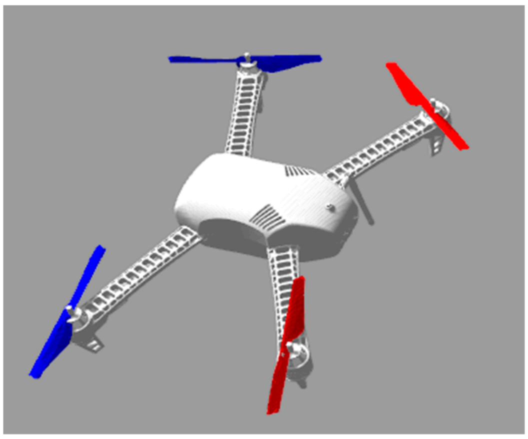


Figure 8. Frame of drone in simulator.

Algorithm 1 Clipped Advantage Policy Optimization Algorithm.

```
1: Initialize the actor and critic network using random parameters $\theta_0$ and $\phi_0$.
2: Initialize the dataset $D_k$ to store trajectories $\tau$.
3. for $k$ = 0, 1, 2, . . ., (max episode) do
4.   Initialize the environment and drone states, obtain initial observation, set $i$ to 0.
5.   for $t$ = 0, 1, 2, . . ., (max step per episode) do
6.     Get action $a_t$ and log probability $loga_t$ of it according to $s_t$ and policy $\pi(\theta_k)$.
7.     Implement $a_t$, get new state $s_{t+1}$ and reward $r_t$ from the environment.
8.     Record the tuple $(s_t, a_t, r_t, s_{t+1}, loga_t)$ to trajectory $\tau$.
9.     if crashed or arrive the destination or reach max timestep then
10.      Reset the environment and drone states.
11.      Store the trajectory $\tau_i$ to $D_k$, $i + 1$.
12.   end for
13.   Calculate rewards-to-go $\hat{R}_t$ using Equation (1) for every trajectory.
14.   Calculate the estimation of advantage $\hat{A}$ according to Equation (11) based on the current value function $V_{\phi_k}$.
15.   Update the actor network using Equation (12).
16.   Update the critic network using Equation (14).
17. end for
```

Table 1. Frame parameters of drone model.

Parameter	Value	Unit
Mass	1.5	kg
ixx	0.0347563	kg·m^2
iyy	0.0458929	kg·m^2
izz	0.0977	kg·m^2
Wheel base	0.511	m
Body width	0.47	m
Body height	0.11	m
Mass of rotor	0.005	kg
Length of rotor	0.2	m
Rotor max rotation velocity	838	rad/s
Rotor drag coefficient	1×10^{-6}	--

We tackled two structurally identical neural networks as actor and critic networks, which possess the same three-layer network structure and number of parameters, as shown in Figure 9. Both networks use the ReLU function as the activation layer, connected behind the input layer and fully connected layer. The only differences were in the input and output layers. The input layer of both networks has the same dimensions as the system state. However, the output layer of the critic network had q = 1, whereas that of the actor network was q = 2.

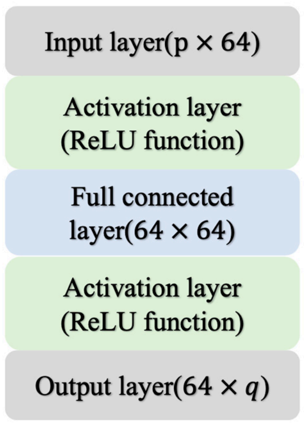

Input layer(p × 64)
Activation layer (ReLU function)
Full connected layer(64 × 64)
Activation layer (ReLU function)
Output layer(64 × q)

Figure 9. Structure of tackled neural networks.

4.2. Policy Training Process

We then used the hyperparameters in Table 2 to train the control policy in all the designed environments. In the experiments, we found that adjusting some of the hyperparameters significantly influenced the training results. For example, an exceptionally small "Max step per episode" will lead to insufficient time for the algorithm to find a complete optimal trajectory, and so more steps are required in a larger scenario. And larger "Timesteps per batch" can improve data utilization efficiency but will also increase data collection time. Our policy optimization algorithm updates the policy, and after each update, the old data become ineffective. Therefore, we chose a parameter value that strikes a balance between efficiency and data collection time. Similar to gradient descent algorithms, the learning rate determines the step size of parameter updates. An excessively small learning rate can lead to slow convergence or becoming stuck in local optima, while an excessively large learning rate can cause oscillations or even divergence. Finally, the "Covariance matrix element value" controls the shape of the action distribution. A larger value will increase the exploration of the policy in the environment, making it less likely to become stuck in local optima. However, it may also slow down the convergence speed of the policy.

Table 2. Hyperparameters of policy training.

Parameter Name	Value
Control period	0.02 s
Max steps	2×10^8
Max step per episode	400
Dimension of states	4
Dimension of actions	2
Timesteps per batch	2048
Discount rate of reward	0.99
Parameter update times per iteration	10
Learning rate	3×10^{-4}
ϵ of clip function	0.2
Covariance matrix element value	0.5

The Gazebo simulator, ROS, and reinforcement learning algorithm were deployed together on a desktop computer running Ubuntu 18.04. The main hardware configuration included an Intel 11,900 k CPU, an Nvidia 3090 GPU, 16-GB RAM, and a 512-GB hard drive. Figure 10 shows the total return for each episode during training and the average of 10 episodes. The y-axis in the figure represents the accumulated reward that is obtained through the policy within an episode, where a higher reward indicates a better performance. Notably, different environments have different upper bounds. As shown in the figure, the proposed algorithm gradually improves the accumulated reward in all environments and finally converges.

It is worth noting that the training process exhibits different characteristics in environments with different obstacle distribution styles. In environments with more scattered obstacle distributions, such as env $1-3$, the training process of the algorithm exhibited higher randomness during convergence. Further, sudden drops in returns may occur over several cycles as the average return increases. This is because the algorithm searches for the optimal policy, and the cluttered environments comprise random obstacles when exploring a certain direction. The returns obtained during these explorations were relatively low, owing to the penalty for collisions in the reward function. After several cycles of optimization, the optimization algorithm finds the optimal policy gradient again, and the average return resumes its upward trend.

In env3, where the obstacles are distributed in an obvious pattern (lined), the exploration–optimization process of the policy is a continuous upward trend. The performance of the algorithm in env5 is impressive. In the initial exploration process, the return value hovers around a low value and even decreases slightly. However, after exploring the gap between the two clusters of obstacles that were designed, the average return quickly rises to near-optimal levels in a few episodes. Similarly, in env6, after exploring the cluster of obstacles, the policy can be quickly optimized to a good level.

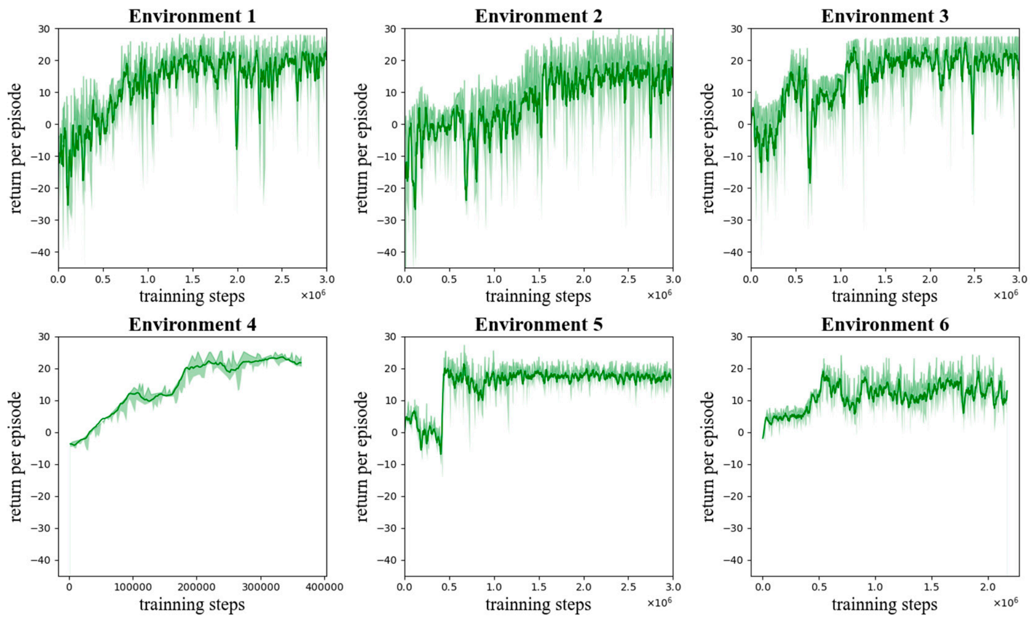


Figure 10. Reward during training.

In summary, our proposed algorithm demonstrates a strong adaptability to different environments, successfully determines the correct policy gradient, and optimizes the policy to improve the return value in environments with cluttered, uniform, and concentrated obstacle distributions.

4.3. Flight Test in Designed Environments

During the training process of the control policy, because exploration and exploitation must be balanced, actions are sampled from the policy network, and this does not guarantee that the optimal action is always sampled. Therefore, during testing, we designed a simplified structure, as shown in Figure 11, by removing components such as the critic network, loss function, optimizer, and data buffer and retaining only the policy network. Moreover, the control command received by the drone is selected directly based on the output of the policy network, which represents the optimal action. We tested the well-trained policy in all the designed environments in the Gazebo simulator. Figure 12 shows the flight trajectories and projectories in the X–Y plane.

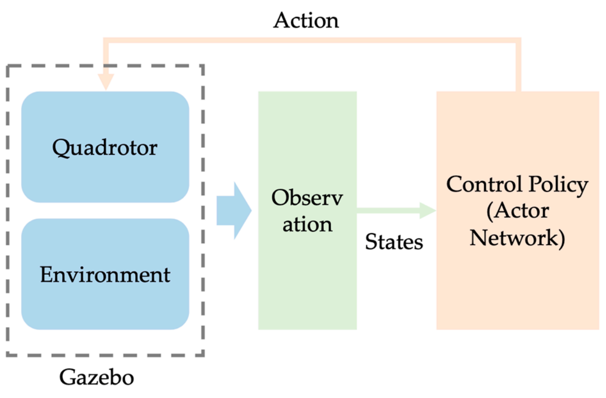


Figure 11. Test framework.

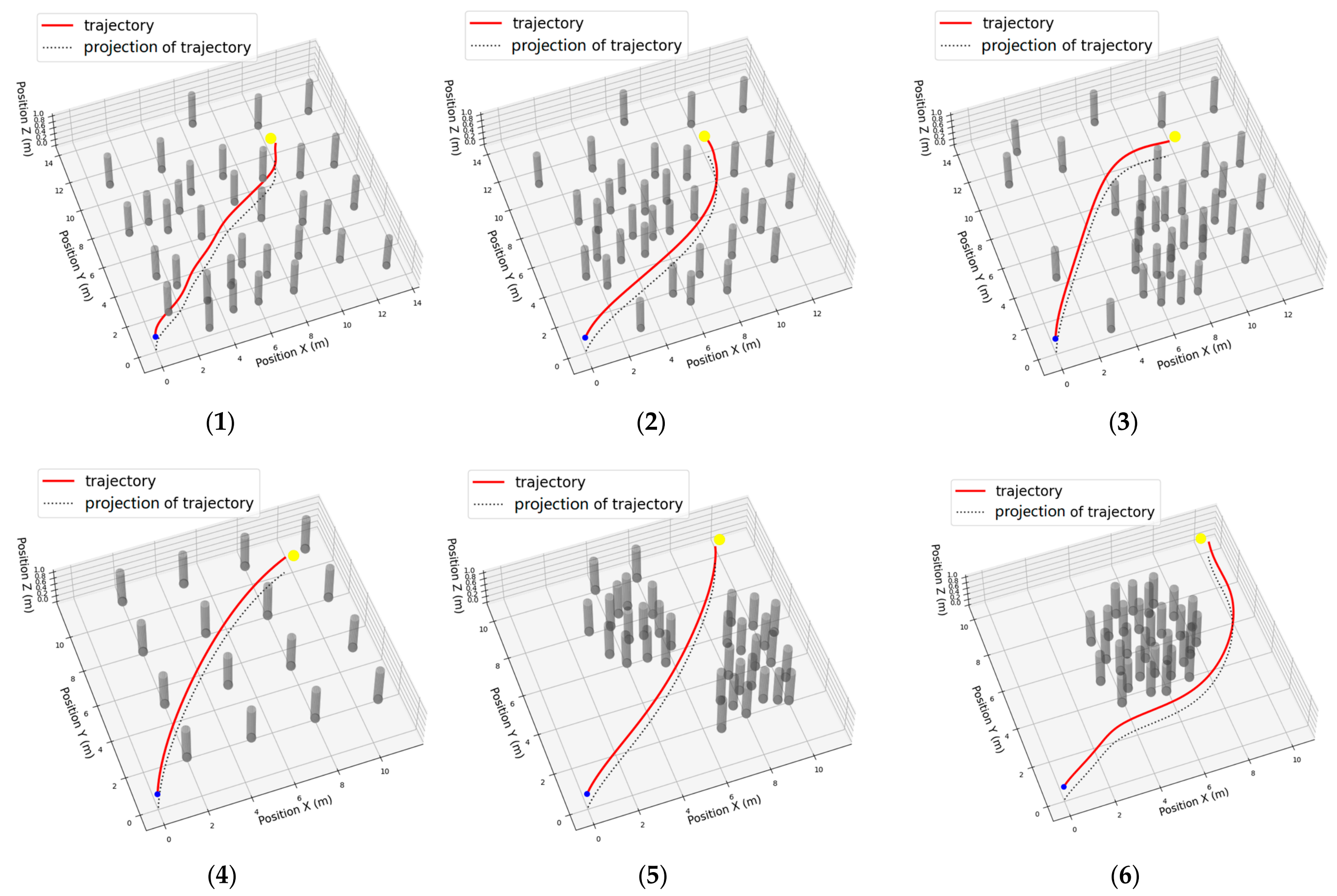
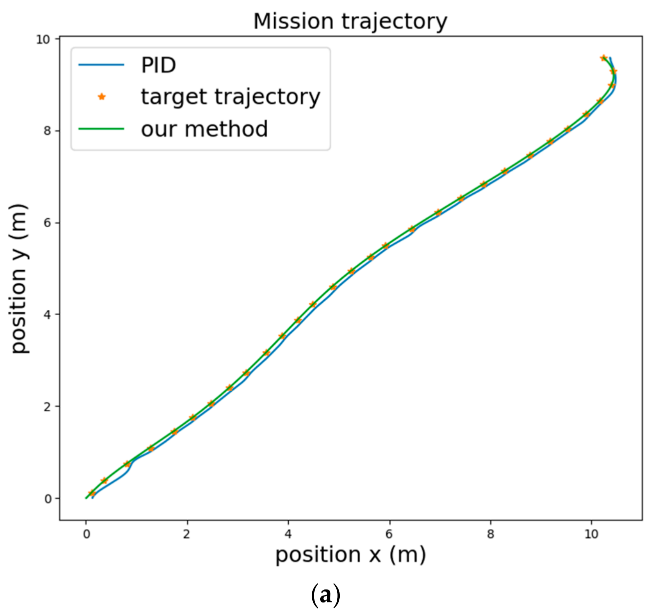


Figure 12. Test flight trajectory in env (**1**–**6**). The mission start from blue dots to yellow boxes.

It can be observed that our algorithm performs well in these scenes and can choose different paths to avoid collisions with obstacles and arrive at the destination as soon as possible. In an environment with randomly distributed obstacles (env 0), neatly arranged obstacles (env 3), and two clusters obstacles (env 4), it selects the paths that are closest to a straight line, perfectly avoiding the obstacles and minimizing the path length. In an environment with a manually designed optimal path, the shortest smooth curve is chosen to minimize energy and time consumption, which proves that the energy evaluation item in our reward function works well. In env 5, with one cluster of obstacles, it first tries to move closer to the direction of the destination and then finds the shortest path to bypass the obstacles.

We compared the performance of the proposed method with that of a PID controller, as shown in Figure 13. It was observed that the proposed method achieves a maximum speed of 7 m/s, whereas the PID controller can only reach 3 m/s and maintain it for a short duration. The proposed method completes the task in 3.4 s, while the PID algorithm requires 18 s. This clearly demonstrates that the proposed method has a significant advantage in terms of mission efficiency in complex environments.



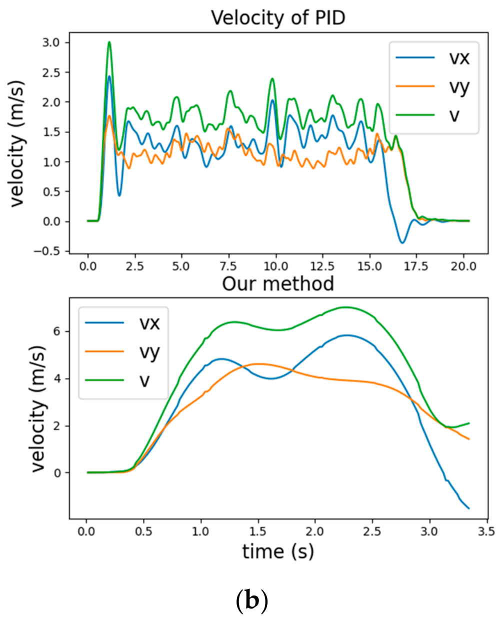
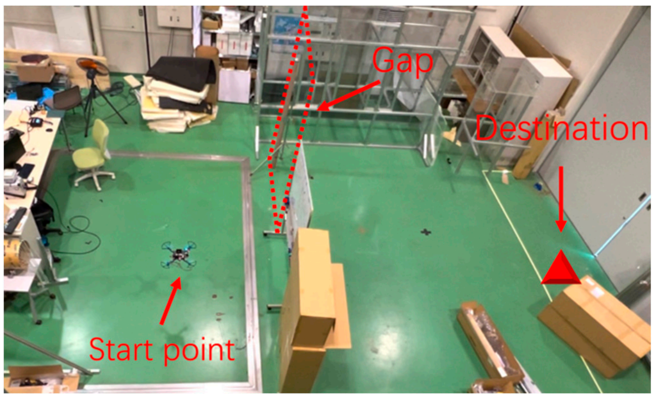


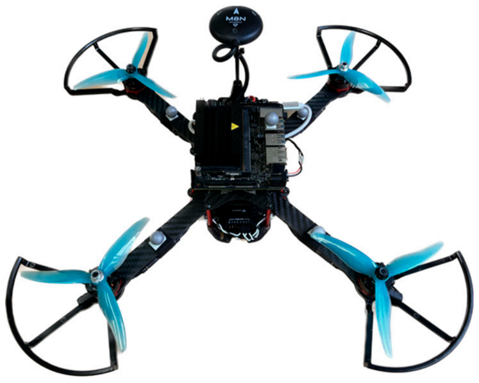
Figure 13. Comparison of proposed method with PID. (**a**) Mission trajectory. (**b**) Velocity during flight.

4.4. Real-World Test

To validate the performance of the proposed algorithm in the real world, we designed a task of navigating through the gap between two walls, as shown in Figure 14a. There is a wall with a gap between the starting point and the destination. The drone must pass through this gap, reach its destination, and hover. The testing methodology is essentially the same as that shown in Figure 11, and the only difference is the substitution of the simulated environment and drone in Gazebo with a real drone (Figure 14b). The low-level controller runs on Pixhawk hardware, whereas our high-level control policy runs on the Jetson Nano. The motion capture system data nodes run in the ROS and communicate through network interfaces.



(**a**)



(**b**)

Figure 14. (**a**) Real-world environment with walls and gap; (**b**) drone.

Figure 15a,b show the position and velocity, respectively. For safety purposes, we have implemented a backup system using traditional methods, which can be switched back and forth with the RL method. After 0.2 s from the start of the mission, the control commands generated by the policy begin to take effect, as shown in Figure 15b. Due to the limited area of the site and considering safety, the quadcopter needs to hover after reaching the destination. Therefore, the maximum total speed is only around 4.39 m/s, and in the simulation, it can reach a maximum speed of 10 m/s. This experiment demonstrates that our method can still control the drone to complete the task of navigating through complex environments in the real world, despite the changes in drone hardware and low-level controllers.

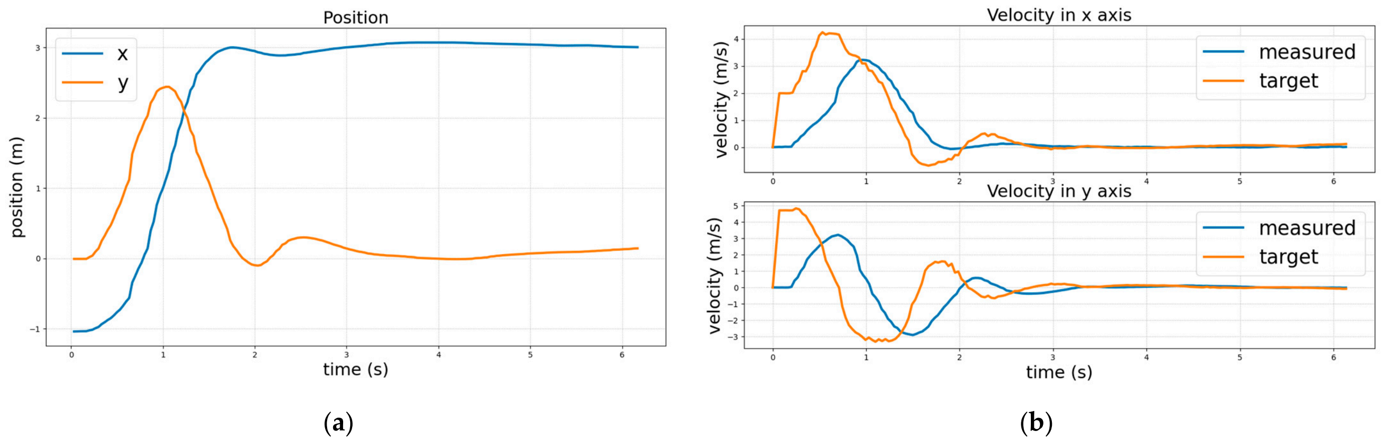


Figure 15. (**a**) Position during mission, start from position (−1, 0), destination position (3, 0); (**b**) measured velocity and target velocity outputted by well-trained policy during mission.

Furthermore, we observed an overshoot that was not present in the simulation tests, as shown in Figure 16a. We analyzed two reasons for this observation. First, the low-level controller of the real-world drone performed poorly than that in the simulator. Second, the policy must wait for all the node data to be updated before it can start the computation. The ROS operates in a soft, real-time manner, and asynchrony between data can cause delays in the control commands. To mitigate the overshoot issue, we adjusted the speed controller parameters and increased the target-tracking speed, which partially improved the situation, as shown in Figure 16b. The test video and code can be found in https://www.youtube.com/watch?v=hRQc1lmJNkY (accessed on 8 July 2023) and https://github.com/pilotliuhx/Drone-navigation-RL.git (accessed on 8 July 2023).

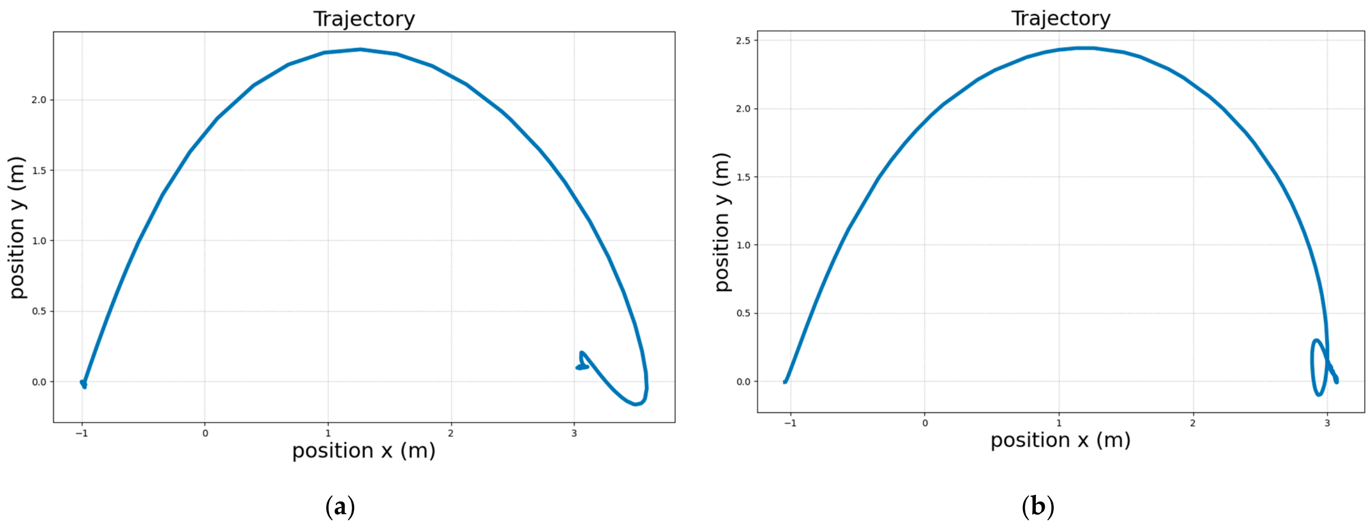


Figure 16. (**a**) Trajectory before parameter tuning and (**b**) trajectory after parameter tuning.

5. Conclusions

This paper proposes a novel end-to-end drone control method based on deep reinforcement learning. It directly takes the drone's state, environmental information, and task objectives as inputs, and outputs the desired velocity to a low-level controller. Compared with traditional model-based pipeline and existing teacher–student learning framework, we faced two main difficulties. The first one was how to lead the policy optimization; we designed a hybrid reward function to evaluate the policy, with a consideration of task progress, energy cost, collision with obstacles, and arrival at the destination. And the other was how to maintain the same level of effectiveness in the simulation within the real world; we used the Gazebo simulator to ensure the reality of physics, and a model-based low-level angular rate, attitude angle, and velocity controller were used to provide consistent performance in velocity command tracking.

To evaluate the performance of the proposed method, obstacle environments with diverse characteristics were designed in a simulator, and the control policy was trained from scratch. Subsequently, the trained control policy was separated and tested in both the Gazebo simulator and a real-world environment. The results show that the proposed method successfully completed flying tasks in complex environments in both simulated and real-world experiments.

It was demonstrated that the proposed method performs well in complex environments with static obstacles. However, certain challenges remain unaddressed. First, the policy's training efficiency was not high. With our current desktop configuration, a single training session takes approximately 7 h. In future works, we will aim to improve the training efficiency by using parallel training technology. Second, the direct application of the training policy to real-world drones resulted in an overshoot. This discrepancy between the simulation and reality suggests that further adjustments or enhancements may be required to improve the control performance in real-world scenarios.

In addition, there is a potential issue when dealing with moving obstacles. The navigation algorithm must consider the presence of dynamic obstacles and adjust its trajectories accordingly. Further investigation is required to ensure the safe and efficient navigation of drones in the presence of moving obstacles. These challenges provide valuable insights for future research and development, and addressing them will further enhance the performance and applicability of the proposed method.

Author Contributions: Conceptualization, H.L.; methodology, H.L.; software, H.L.; validation, H.L. and S.S.; formal analysis, H.L.; investigation, H.L. and S.S.; resources, S.S.; data curation, H.L.; writing—original draft preparation, H.L.; writing—review and editing, S.S. and H.L.; visualization, H.L. and S.S.; supervision, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The video of the experiment can be found at: https://www.youtube.com/watch?v=hRQc1lmJNkY (accessed on 8 July 2023). The code can be found at: https://github.com/pilotliuhx/Drone-navigation-RL.git (accessed on 8 July 2023).

Acknowledgments: The authors would like to thank the Graduate School of Science and Engineering, Chiba University, Chiba, Japan, for the resources provided by them.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Schedl, D.C.; Kurmi, I.; Bimber, O. An autonomous drone for search and rescue in forests using airborne optical sectioning. *Sci. Robot.* **2021**, *6*, eabg1188. [CrossRef]
2. Hayat, S.; Yanmaz, E.; Bettstetter, C.; Brown, T.X. Multi-objective drone path planning for search and rescue with quality-of-service requirements. *Auton. Robot.* **2020**, *44*, 1183–1198. [CrossRef]
3. Li, Z.; Wang, Q.; Zhang, T.; Ju, C.; Suzuki, S.; Namiki, A. UAV High-Voltage Power Transmission Line Autonomous Correction Inspection System Based on Object Detection. *IEEE Sens. J.* **2023**, *23*, 10215–10230. [CrossRef]
4. Seo, J.; Duque, L.; Wacker, J. Drone-enabled bridge inspection methodology and application. *Autom. Constr.* **2018**, *94*, 112–126. [CrossRef]
5. Ayele, Y.Z.; Aliyari, M.; Griffiths, D.; Droguett, E.L. Automatic Crack Segmentation for UAV-Assisted Bridge Inspection. *Energies* **2020**, *13*, 6250. [CrossRef]
6. Song, Y.; Scaramuzza, D. Policy Search for Model Predictive Control with Application to Agile Drone Flight. *IEEE Trans. Robot.* **2022**, *38*, 2114–2130. [CrossRef]
7. Li, S.; Ozo, M.M.; De Wagter, C.; de Croon, G.C. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *Robot. Auton. Syst.* **2020**, *133*, 103621. [CrossRef]
8. Nonami, K. Present state and future prospect of autonomous control technology for industrial drones. *IEEJ Trans. Electr. Electron. Eng.* **2020**, *15*, 6–11. [CrossRef]

9. Falanga, D.; Kim, S.; Scaramuzza, D. How Fast Is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid. *IEEE Robot. Autom. Lett.* **2019**, *4*, 1884–1891. [CrossRef]
10. Florea, H.; Petrovai, A.; Giosan, I.; Oniga, F.; Varga, R.; Nedevschi, S. Enhanced Perception for Autonomous Driving Using Semantic and Geometric Data Fusion. *Sensors* **2022**, *22*, 5061. [CrossRef] [PubMed]
11. Foehn, P.; Romero, A.; Scaramuzza, D. Time-optimal planning for quadrotor waypoint flight. *Sci. Robot.* **2021**, *6*, eabh1221. [CrossRef] [PubMed]
12. Pfeiffer, C.; Scaramuzza, D. Human-Piloted Drone Racing: Visual Processing and Control. *IEEE Robot. Autom. Lett.* **2021**, *6*, 3467–3474. [CrossRef]
13. Torrente, G.; Kaufmann, E.; Foehn, P.; Scaramuzza, D. Data-Driven MPC for Quadrotors. *IEEE Robot. Autom. Lett.* **2021**, *6*, 3769–3776. [CrossRef]
14. Han, Z.; Wang, Z.; Pan, N.; Lin, Y.; Xu, C.; Gao, F. Fast-Racing: An Open-Source Strong Baseline for $\mathrm{SE}(3)$ Planning in Autonomous Drone Racing. *IEEE Robot. Autom. Lett.* **2021**, *6*, 8631–8638. [CrossRef]
15. Wu, Y.; Ding, Z.; Xu, C.; Gao, F. External Forces Resilient Safe Motion Planning for Quadrotor. *IEEE Robot. Autom. Lett.* **2021**, *6*, 8506–8513. [CrossRef]
16. Ye, H.; Zhou, X.; Wang, Z.; Xu, C.; Chu, J.; Gao, F. TGK-Planner: An Efficient Topology Guided Kinodynamic Planner for Autonomous Quadrotors. *IEEE Robot. Autom. Lett.* **2020**, *6*, 494–501. [CrossRef]
17. Araujo, P.; Miranda, R.; Carmo, D.; Alves, R.; Oliveira, L. Air-SSLAM: A Visual Stereo Indoor SLAM for Aerial Quadrotors. *IEEE Geosci. Remote. Sens. Lett.* **2017**, *14*, 1643–1647. [CrossRef]
18. Saeedi, S.; Thibault, C.; Trentini, M.; Li, H. 3D Mapping for Autonomous Quadrotor Aircraft. *Unmanned Syst.* **2017**, *5*, 181–196. [CrossRef]
19. Faessler, M.; Fontana, F.; Forster, C.; Mueggler, E.; Pizzoli, M.; Scaramuzza, D. Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle. *J. Field Robot.* **2016**, *33*, 431–450. [CrossRef]
20. Shi, L.; Li, W.; Shi, M.; Shi, K.; Cheng, Y. Opinion Polarization Over Signed Social Networks with Quasi Structural Balance. *IEEE Trans. Autom. Control.* **2023**, *99*, 1–8. [CrossRef]
21. Bailey, J.P.; Nash, A.; Tovey, C.A.; Koenig, S. Path-length analysis for grid-based path planning. *Artif. Intell.* **2021**, *301*, 103560. [CrossRef]
22. Liu, S.; Mohta, K.; Atanasov, N.; Kumar, V. Search-Based Motion Planning for Aggressive Flight in SE(3). *IEEE Robot. Autom. Lett.* **2018**, *3*, 2439–2446. [CrossRef]
23. Oleynikova, H.; Taylor, Z.; Fehr, M.; Siegwart, R.; Nieto, J. Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 1366–1373. [CrossRef]
24. Sun, S.; Romero, A.; Foehn, P.; Kaufmann, E.; Scaramuzza, D. A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight. *IEEE Trans. Robot.* **2022**, *38*, 3357–3373. [CrossRef]
25. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. 2013. Available online: https://arxiv.org/abs/1312.5602 (accessed on 2 February 2022).
26. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous Control with Deep Reinforcement Learning. 2015. Available online: https://arxiv.org/abs/1509.02971 (accessed on 10 August 2022).
27. Li, H.; Zhang, Q.; Zhao, D. Deep Reinforcement Learning-Based Automatic Exploration for Navigation in Unknown Environment. *IEEE Trans. Neural Networks Learn. Syst.* **2019**, *31*, 2064–2076. [CrossRef] [PubMed]
28. Wang, W.; Hu, Y.; Scherer, S. Tartanvo: A generalizable learning-based vo. In Proceedings of the Conference on Robot Learning. PMLR, London, UK, 8–11 November 2021.
29. Kaufmann, E.; Bauersfeld, L.; Scaramuzza, D. A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight. In Proceedings of the 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, USA, 23–27 May 2022. [CrossRef]
30. Hanover, D.; Loquercio, A.; Bauersfeld, L.; Romero, A.; Penicka, R.; Song, Y.; Cioffi, G.; Kaufmann, E.; Scaramuzza, D. Autonomous Drone Racing: A Survey. *arXiv* **2023**, arXiv:2301.01755.
31. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 6–11 July 2015; pp. 1889–1897.
32. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
33. Liu, Y.-C.; Huang, C.-Y. DDPG-Based Adaptive Robust Tracking Control for Aerial Manipulators with Decoupling Approach. *IEEE Trans. Cybern.* **2022**, *52*, 8258–8271. [CrossRef] [PubMed]
34. Dong, Y.; Zou, X. Mobile robot path planning based on improved ddpg reinforcement learning algorithm. In Proceedings of the 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 16–18 October 2020; pp. 52–56.
35. He, R.; Lv, H.; Zhang, S.; Zhang, D.; Zhang, H. Lane Following Method Based on Improved DDPG Algorithm. *Sensors* **2021**, *21*, 4827. [CrossRef]
36. Zhang, Z.; Chen, J.; Chen, Z.; Li, W. Asynchronous Episodic Deep Deterministic Policy Gradient: Toward Continuous Control in Computationally Complex Environments. *IEEE Trans. Cybern.* **2021**, *51*, 604–613. [CrossRef]

37. Li, W.; Qin, K.; Li, G.; Shi, M.; Zhang, X. Robust bipartite tracking consensus of multi-agent systems via neural network combined with extended high-gain observer. *ISA Trans.* **2023**, *136*, 31–45. [CrossRef]
38. Furrer, F.; Burri, M.; Achtelik, M.; Siegwart, R. *Robot Operating System (ROS): The Complete Reference*; Springer International Publishing: Cham, Switzerland, 2016; Volume 1, pp. 595–625. [CrossRef]