REINFORCEMENT LEARNING FOR AUTONOMOUS DRONE RACING

PEI AN HSIEH [PAHSIEH@SEAS], THANH LY [THANHLY@SEAS], XUAN HAO ONG [OXUANHAO@SEAS],

Abstract. This project delves into the application of Reinforcement Learning (RL) in the challenging domain of autonomous drone racing, highlighting the intricacies of policy training and the critical role of design choices in achieving effective training outcomes. By utilizing model-free methods, particularly the Proximal Policy Optimization (PPO) algorithm, this study demonstrates how different control inputs, reward structures, and training strategies influence the performance of RL policies. Extensive simulations, incorporating customized drone dynamics and a variety of racecourse configurations, provide a platform for assessing the generalization capabilities of learned policies across different environments. The results underline the significant impact of action space and reward configurations on the robustness and adaptability of RL policies, offering insights into optimal strategies for deploying RL in real-world applications like drone racing.

1. Introduction

Reinforcement learning (RL) has shown promise in solving time-optimal trajectory planning for drones. Model-free methods learn a parametric policy through direct interaction with the environment, making them suitable for problems that are challenging to model. Notably, Song et al. demonstrated significant advancements in autonomous drone racing, where RL-learned policies consistently outperformed human champions and surpassed state-of-the-art trajectory planners and controllers [1]. Their earlier work [2] explored different reward structures and control inputs, prompting us to investigate how these design choices impact policy performance.

While existing research showcases successful outcomes of trained policies for demanding drone racing tasks, the underlying decisions in developing environments and training strategies remain less transparent. In this study, we conducted experiments with various setups and strategies, assessing their effectiveness in training RL policies for drone racing.

1.1. Contributions.

This paper aims to explore decision-making nuances in implementing RL for drone racing. Emphasis is placed on optimal configurations of action and observation spaces, training methods, and reward frameworks. We developed a customized simulator tailored for model-free RL training, including drone dynamics, collision detection, and visualization. Extensive experimentation with diverse racecourses and training strategies seeks insights into guiding design choices.

2. Approach

2.1. Reinforcement Learning Algorithm.

We used Proximal Policy Optimization (PPO) due to its ease of implementation, stability, and empirical performance, as discussed in [1]. PPO was implemented using Stable Baselines3 [3].

2.2. Drone Dynamics and Action Space.

This study experimented with two control inputs for the action space: rotor speed and body rate control, following the precedent set by previous papers such as [1] and [2]. The six degree-of-freedom nominal drone dynamics model aligns with models proposed by Mellinger et al. [4] and Kongyao et al. [5], serving as the basis for the simulation environment used in RL training.

2.2.1. *Drone Dynamics.* Consider a drone with mass m and moments of inertia $I = \text{diag}(I_{xx}, I_{yy}, I_{zz})$. The dynamics are given by:

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \ddot{r} = \frac{1}{m}\left(\begin{bmatrix}0\\0\\-mg\end{bmatrix} + R_{WB}\begin{bmatrix}0\\0\\u_1\end{bmatrix}\right), \quad \begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \dot{\omega} = I^{-1}\left(u_2 - \left[\begin{bmatrix}p\\q\\r\end{bmatrix} \times I \begin{bmatrix}p\\q\\r\end{bmatrix}\right]\right)$$

where u_1 is thrust, $u_2 = [u_x, u_y, u_z]$ are body rates, and R_{WB} is the rotation matrix between the drone body and world frames.

2.2.2. *Rotor Speed Control.* The relation between rotor speeds, thrust, and angular rates can be described by the following mapping: $u = [u_1, u_2]^T$, where ω_i denotes the rotor speed of rotor i, k_F and k_M are physical parameters, and L is the drone's arm length.

$$u = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

2.2.3. *Body Rate Control.* Controlling thrust and body rate is simply controlling with inputs u_1 and u_2.

2.3. **State and Observation Space.**

2.3.1. *State Space.* In our simulator, we used: $\hat{\mathbf{x}} = [\mathbf{x}, \mathbf{q}, v, \omega] \in R^{13}$ to capture the state of the drone. $\mathbf{x}$ is the position, $\mathbf{q} = [q_x, q_y, q_z, q_w]$ is the quaternion, v is the velocity and ω is the angular velocity.

2.3.2. *Observation Space.* The observation space of the RL agent follows [1]: $\hat{\mathbf{o}} = [v, R_{WB}, \delta p, \delta c] \in R^{36}$. $\delta p \in R^{12}$ is the relative position of the drone and the four gate corners of the next gate. $\delta c \in R^{12}$ is the relative position of the gate corners of the next gate and the following gate.

2.4. **Reward Structure.** Drone rewards are contingent on collision status and racecourse progress, utilizing proxy rewards for continuous feedback, enhancing training efficiency over single-end rewards. Our implementation draws from [1], [2], and [6] for reward values and algorithm programming.

2.4.1. *Collision and Out-of-Bound Reward.* Each instance verifies drone collisions with gates during its motion from previous to current states. A collision or flying outside the map incurs a -20 negative reward, terminating the training episode and resetting the environment.

2.4.2. *Gate Progress Reward.* Positive reward is granted when the drone approaches the gate center, calculated as:

$$r_{gp} = \|g_k - p_{t-1}\| - \|g_k - p_t\| - b\|\omega_k\| = r_d - b\|\omega_k\| \tag{1}$$

where g_k is the target gate center, p_{t-1} is the previous drone position, p_t is the current drone position, b is the coefficient of penalty on body rate, and ω_k is the body rate.

2.4.3. *Path Following Reward.* Path following reward enhances gate progress reward. The drone aims to move forward along a line segment connecting consecutive gate centers. The reward is calculated as:

$$r_p = s\left(p_{t,proj}\right) - s\left(p_{t-1,proj}\right); s\left(p\right) = \frac{(p - g_1) \cdot (g_2 - g1)}{\|g_2 - g_1\|} \tag{2}$$

where $p_{t,proj}$ and $p_{t-1,proj}$ are projections of the current and previous drone positions onto the segment, and $s\left(p\right)$ indicates progress from the previous gate's center g_1 to the next gate's center g_2.

2.4.4. *Racecourse Completion.* Upon visiting all gates, the drone receives a final reward of 20..

2.4.5. *Reward Modes.* We implement two reward functions optimizing different objectives. The primary function, Gate Progress Reward, encourages shorter trajectory lengths to improve race times:

$$r = gain \cdot r_d - b\|\omega_k\| \tag{3}$$

Additionally, to promote safer trajectories and reduce collision rates during training, we implement an additional safety reward from [2] (evaluated in Section 3.4). This alternative function, with α as a hyperparameter, balances progress maximization and risk minimization:

$$r = gain \cdot (r_p + r_d) + \alpha r_s - b\|\omega_k\| \tag{4}$$

2.5. **Collision Detection.** We modeled the drone as a sphere with radius ρ. To detect collisions, we set up two criteria: (1) the drone collides with the gates; (2) the drone ventures out of the designated map. For the first criterion, we refer to [6] to program the collision detector. For the second criterion, we apply out-of-range conditionals to determine whether the drone is inside the map.

2.6. **Policy Initialization Strategy.** Early in training, most rollouts terminate due to gate or ground collisions. If the drone is initialized at the starting position of the racecourse, data collection would require many update steps until the policy learns to explore the entire state space. To improve training efficiency, we randomly initialized the drone in a hovering state at the centers of each path segment between consecutive gates, effectively exposing the drone to all gate observations to cover the state space more efficiently. An additional measure to potentially improve training efficiency is discussed in Section 3.5.

3. Experimental Results

3.1. **Basic Experimental Design.** We conducted experiments adjusting training structures to analyze policy performance. Experiments compared action spaces, reward structures, and initialization strategies, assessing drone learning across various racecourses and generalization to unseen ones. Other structures and parameters were kept constant as controls for each experiment. Racecourses and gates were designed to suit Crazyflie 2.1 drone dynamics (e.g., 10 m/s max speed) to ensure efficient exploration without prolonged mapping. Control variables include drone dynamics (Rate control), reward structure (Gate Progress Reward), and training strategy (Fixed initial states).

3.2. **Performance Measures.** To evaluate the trained policies for each of the experiments in Section 3, we rolled out each policy for 1000 episodes and tabulated the following measures for comparison and analysis:

- Timesteps trained: Total number of training timesteps to arrive at the evaluated policy.
- Completion rate (%): Percentage of rollouts completing the entire racecourse.
- Average gates passed: Mean number of gates that the drone passed.
- Average total reward: Mean rewards per episode. Note: higher reward doesn't necessarily imply better policy.
- Average trajectory length: Mean trajectory length of the drone to complete the entire racecourse.
- Average race time (s): Average time taken by the drone to complete the entire racecourse. This is calculated by taking the average of only the successful trajectory length and multiplying it by 0.01s per timestep.

3.3. **Action Space Comparison.**

3.3.1. *Study description.* We trained on the baseline circular (8-gate) race course using different control inputs mentioned in 2.2 for 10 million timesteps.

3.3.2. *Results and discussion.* Analysis of outcomes (Table 1) shows significant disparities between body rate and rotor speed control. Body rate control exhibits superior success rates and rewards, attributed to its direct mapping with drone dynamics. In contrast, rotor speed control adds complexity by requiring an additional mapping, leading to less stable trajectories (Figure 1). This underscores the importance of simplicity and coherence in action-observation mapping for neural network efficacy.

Table 1. Evaluation of Action Spaces on Circular (8 gate) Course

Controls	Body Rate	Rotor Speed
Completion	32.3%	0.0%
Gates passed	6.3	4.4
Total reward	367 ± 127	232 ± 45
Traj length	238 ± 63	130 ± 21

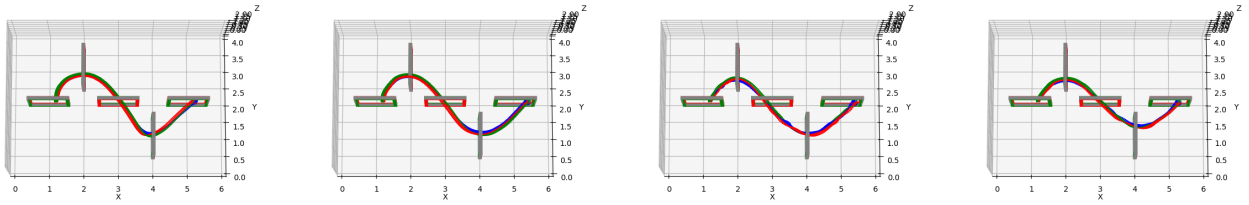
Table 2. Evaluation of Reward Modes on S-Course

Controls	Body Rate		Rotor Speed	
Reward Mode	0	1	0	1
Completion	50.7%	9.8%	1.9%	29.1%
Gates passed	3.95	3.88	3.87	3.93
Total reward	439 ± 58	370 ± 73	308 ± 47	340 ± 65
Traj length	240 ± 13	219 ± 28	157 ± 18	163 ± 17

3.4. **Reward Structure Comparison.**

3.4.1. *Study description.* As detailed in 2.4.5, this section evaluates various reward modes for complex racecourses and objectives. We selected the S-course as a baseline and trained 12 million steps for two reward functions, considering its moderately complex shape, each utilizing two distinct action spaces.

3.4.2. *Results and discussion.* Results comparing various rewards and dynamics are shown in Table 2. Without safety rewards, trajectories exhibit riskier behavior, evident in sharp V-turns (Figure 1(a)). Conversely, with safety rewards, the drone navigates obstacles, resulting in smoother U-turns (Figure 1(b)). However, this distinction is less pronounced with rotor speed control. `Mode 1` better suits rotor speed control, indicating the necessity of safety rewards for unstable controls.



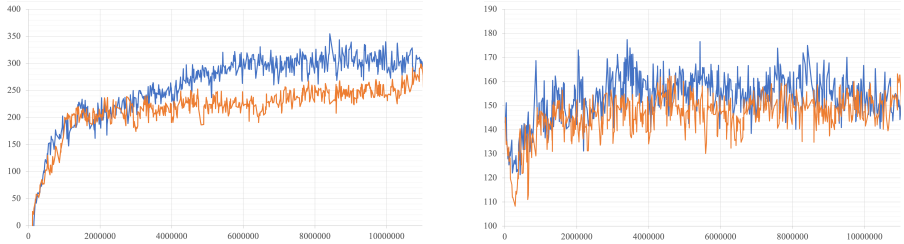
(a) Mode 0, Body Rate (b) Mode 1, Body Rate (c) Mode 0, Rotor Speed (d) Mode 1, Rotor Speed

FIGURE 1. S-Course with different reward functions and action spaces

3.5. **Initialization Strategy Comparison.**

3.5.1. *Study description.* To enhance training efficiency, we implemented an initial state buffer inspired by [1], collecting states along trajectories passing through multiple gates. We compared the efficiency of completing the S-course map between fixed and updated initial states.

3.5.2. *Results and discussion.* Figure 2 indicates that while rollout rewards and lengths converge similarly by training end, using an initial state buffer shows poorer training performance compared to fixed initial states. Despite the theoretical efficiency improvement, inconsistent buffer quality during training leads to inconsistent performance boosts. Further exploration, like extracting entire successful trajectory lengths rather than just initial gate states, could yield better approaches with more time.



(a) Average Rollout Rewards (b) Average Rollout Length

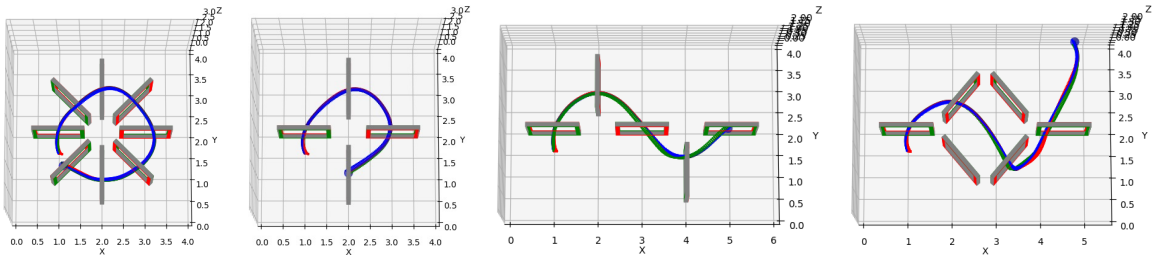
FIGURE 2. Comparison of rollouts by (i) fixing the initial states (blue) and (ii) updating an initial states buffer throughout the training (orange) of 10 million timesteps.

3.6. **Racecourse Evaluation.**

3.6.1. *Study description.* We trained and evaluated the drone policy performances on racecourses with varying complexity: (1) circular courses (8 & 4 gates), (2) S-course, and (3) figure-of-8 course.

3.6.2. *Results and discussion.* Table 3 presents trained policy results, while Figure 3 displays the best-performing policy rollouts across evaluated racecourses. The drone mastered simpler circular (8 and 4 gates) and S-course tracks within 10 million timesteps, albeit with a lower completion rate for the more complex S-course. On average, the drone passed 6.3/8 gates (78.8%) for the circular course and 4.0/5 gates (80%) for the S-course, indicating relatively strong performance on both.

The drone was unable to complete the Figure-of-8 course. Even with a total of over 100 million timesteps being trained over multiple training strategies, the drone's performance deteriorated after around 20 million



(a) Circular (8 gates) (b) Circular (4 gates) (c) S-course (5 gates) (d) Figure-of-8 (6 gates)

FIGURE 3. Best performing RL policy rollouts for the 3 evaluated racecourses.

timesteps as it experienced some form of forgetting. Figure 3(d) shows a trajectory rolled out by the best-performing policy, where the drone achieved an average of 3.8/6 gates (63%), completing half of the Figure-of-8 most of the time. Given more time, better policies could be obtained with other approaches or strategies, such as tuning the reward structure (modifying the gain for Gate Progress Reward) and increasing discount factor γ to reduce the impact of forgetting for these longer courses.

TABLE 3. Performance Evaluation on Various Racecourses

Racecourse	Circle (8)	Circle (4)	S (5)	Fig-of-8 (6)
Completion	32.3%	94.0%	8.7%	0%
Gates passed	6.3	3.94	4.0	3.8
Total reward	367 ± 127	458 ± 22	463 ± 38	361 ± 36
Traj length	238 ± 63	197 ± 4	218 ± 7	247 ± 22

TABLE 4. Generalization Evaluation of Learned Policies on 4-gate Racecourse

Policy	Circle (8)	S (5)
Completion	62.8%	0.0%
Gates passed	2.535	3.0
Total reward	294 ± 115	22 ± 75
Race time	1.84 ± 0.05	DNF

3.7. **Generalization Evaluation.**

3.7.1. *Study description.* We examined how RL policies generalize across various racecourses by testing policies trained on circular (8-gate) and S-course tracks on a simpler 4-gate course. These were compared to a policy initially trained on the 4-gate course, Table 3, column 3.

3.7.2. *Results and discussion.* From Table 4, the 8-gate policy completes a circular (4-gate) map, whereas the S-course policy fails to do so. Remarkably, the 4-gate policy achieves a higher success rate but takes longer, on average, to finish the race course. This illustrates the RL-trained policy's ability to generalize to similar racecourses, but a more intricate policy like the S-course policy, designed for handling left and right turns, may not perform optimally on simpler racecourses.

4. DISCUSSION

Implementing RL for complex tasks such as drone racing is challenging due to the influence of numerous interrelated factors. Prior to conducting the experiments, refinement of several factors are necessary. Control inputs that have direct relations with the observations are preferred. The reward structures and environment setups such as map sizes and gate structures needs to be adjusted to better suit drone dynamics to facilitate convergence to successful policies. In addition, hyperparameters of the PPO algorithm needs iteratively tuning (e.g. batch size, network architectures), initialization buffer and the Soft Actor-Critic (SAC) algorithm was tested but deemed less effective. Despite these efforts, the implementation falls short in completing more complex racecourses (Figure 3(d) in Section 3.6) and exhibiting poor performance on unseen ones (Section 3.7). Improvements could involve rewriting the code in a more efficient language like C++ to allow for more efficient exploration of additional strategies and parameters. The initialization buffer strategy (3.5) and domain randomization are also potential areas for further investigation.

References

[1] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza, "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning," *Science Robotics*, vol. 8, no. 82, p. eadg1462, 2023. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.adg1462

[2] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous Drone Racing with Deep Reinforcement Learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1205–1212.

[3] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *IEEE International Conference on Robotics and Automation*, pp. 2520–2525, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/5980409

[5] T. Z. J. K. Y. Chee and M. A. Hsieh, "Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots," *IEEE Robotics and Automation Letters*, vol. 7, pp. 2819–2826, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9691797

[6] Y. Shen, Q. Jia, G. Chen, Y. Wang, and H. Sun, "Study of rapid collision detection algorithm for manipulator," in *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*, 2015, pp. 934–938.

Acknowledgment

The quadDynamics.py is modified from a research project with Kongyao Chee [ckongyao@seas].