AUTONOMOUS DRONE RACING THROUGH GATES: MPC VS. RL

KAITIAN CHAO, JINYUAN ZHANG, YUFEIYANG GAO, YUCHEN ZHENG

ABSTRACT. In this project, we compare Model Predictive Control (MPC) and Actor-Critic Reinforcement Learning (RL) for autonomous drone racing. MPC offers robust performance under ideal conditions but may struggle with unmodeled dynamics. RL adapts well with more flexibility but requires extensive training. We implemented both RL and MPC for drone racing task. Our results highlight their strengths and limitations, motivating Actor-Critic MPC as a hybrid approach that combines MPC's reliability with RL's flexibility for improved drone navigation in the future.

1. INTRODUCTION

Autonomous drone racing demands high-speed navigation through multiple gates with precision and agility, offering a prime testbed for advanced control and learning algorithms. Model Predictive Control (MPC), relying on accurate models and offline trajectory planning, can provide performance guarantees but struggles under high uncertainty or rapid environmental changes. By contrast, Reinforcement Learning (RL) is inherently adaptive and excels in uncertain conditions without needing an explicit model, though training can be resource-intensive and policies can be difficult to interpret. In this project, we compare MPC and Actor-Critic RL on an autonomous drone racing task. We employ time-optimal offline planning for MPC and exploit RL's adaptability to investigate potential synergies between structured reliability and end-to-end optimization. These insights set the stage for integrated methods like Actor-Critic MPC, aiming to fuse predictive capabilities with learning flexibility for faster and more robust quadrotor navigation. Example of using RL to train drone racing is shown https://drive.google.com/drive/folders/17WymVr2KXJIAYUAR2Nn9ZuCj-4IGZF2r.

2. BACKGROUND

We model the quadrotor as a rigid body with state space described as

$$\boldsymbol{x} = \left[\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}, \boldsymbol{w}\right]^T \tag{1}$$

where Let $\boldsymbol{p} \in \mathbb{R}^3$ denote the quadrotor's position in the inertial frame, $\boldsymbol{q} \in \mathbb{SO}(3)$ is orientation as a unit quaternion relative to the inertial frame, $\boldsymbol{v} \in \mathbb{R}^3$ is linear velocity in the inertial frame, and $\boldsymbol{\omega} \in \mathbb{R}^3$ is angular velocity in the body-fixed frame.

The dynamics of the quadrotor are governed by the state-space equation, as $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) = \left[\dot{\boldsymbol{p}}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{v}}, \dot{\boldsymbol{\omega}}\right]^T$, with control input $\boldsymbol{u} = [T_1, T_2, T_3, T_4]^T \in \mathbb{R}^4$ corresponding to the thrusts generated by the four rotors. The detailed dynamic equations are

$$\begin{aligned} \dot{\boldsymbol{p}} &= \boldsymbol{v} & \dot{\boldsymbol{q}} &= \frac{1}{2}\boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}, \\ \dot{\boldsymbol{v}} &= \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m}\mathbf{R}(\boldsymbol{q})\mathbf{T} & \dot{\boldsymbol{\omega}} &= \mathbf{J}^{-1}\big(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}\big). \end{aligned} \tag{2}$$

where $\otimes$ is the quaternion multiplication, $\mathbf{R}(\boldsymbol{q})$ is the quaternion rotation, transforming vectors from the body frame to the inertia frame. Here, $g = 9.81\text{m/s}^2$ is the gravity. $\mathbf{T} = \left[0, 0, \sum_{i=1}^{4} T_i\right]^T$ is the total thrust vector. $\mathbf{J}$ is the inertia matrix. $\boldsymbol{\tau}$ represents the three-dimensional torque, specifically defined according to our quadrotor's "$\times$" configuration with arm length l and rotor's torque constant c_τ as:

$$\boldsymbol{\tau} = \begin{bmatrix} l/\sqrt{2}(T_1 + T_2 - T_3 - T_4) \\ l/\sqrt{2}(-T_1 + T_2 + T_3 - T_4) \\ c_\tau(-T_1 - T_2 + T_3 + T_4) \end{bmatrix} \tag{3}$$

3. Related Work

There are two state-of-the-art optimal control methods commonly used in drone racing. The first, [1], involves planning a minimum lap time trajectory offline and then employing MPC to track it online. The second, known as contouring control [2], achieves time-optimal flights by simultaneously progressing along a predefined reference path while minimizing deviation from it during the race. Traditional MPC, [3, 1], approaches rely heavily on accurate modeling and offline computation, which can limit their responsiveness in dynamic or uncertain environments. Reinforcement Learning has emerged as a promising alternative due to its model-free nature and adaptability. [1] uses RL-based controllers and has demonstrated strong performance in complex environments, but with the cost of high training complexity. Recent work has begun to explore the possibility of combining MPC and RL [4].

4. Approach

4.1. **Model Predictive Control.** In general, optimal control methods, such as MPC, typically approach drone racing by dividing the task into two primary stages: planning and control. During the planning stage, a reference trajectory $p_{\text{ref}} = (\boldsymbol{x}_{\text{ref}}, \boldsymbol{u}_{\text{ref}})_k$ is generated. This trajectory is designed to be both dynamically feasible and time-optimal, which takes into account the predetermined configuration of the race gates. Once the reference trajectory is established, the control stage involves driving the drone to follow this trajectory accurately.

Our implementation utilizes the method in [1], which plans a minimum lap time **trajectory** offline and then employs MPC to follow it online.

4.1.1. *Trajectory Planning.* Planning truly time-optimal trajectories through multiple waypoints poses a significant challenge, particularly when the exact time allocation for each waypoint is unknown.

Inspired by the direct multiple shooting approach, we discretize the system's states and inputs over a time horizon divided into N intervals. Let $\{\boldsymbol{x}_k, \boldsymbol{u}_k\}_{k=0}^{N}$ be the state/input pairs at discrete nodes t_k. The system dynamics are imposed via nonlinear equality constraints (e.g., through Runge–Kutta integration), while the single-rotor thrust and torque limits are encoded as inequality constraints. The cost function is purely the end-to-end flight time, t_N. This ensures that *maximizing* progress while *minimizing* the overall trajectory time is prioritized. However, to pass through multiple race gates (or waypoints) in a time-optimal manner, each waypoint constraint must be formulated to avoid presupposing fixed time allocation.

Following the approach in [3], we address the time allocation issue by introducing a dedicated set of *progress variables* μ_k that indicate if a waypoint has been "completed." These progress variables can only switch from "incomplete", represented by 0, to "completed", represented by 1, when the quadrotor is within a specified spatial tolerance, δ, of that waypoint. The distance from current position, p_k to the waypoint location p_{wj} is measured with:

$$(p_k - p_{wj})^T (p_k - p_{wj}) \leq \delta$$

The progress variable and the distance functions are formulated together as the complementaruty constraint:

$$\mu_k \cdot ||p_k - p_{wj}||_2^2 = 0$$

This ensures that one of the progress variable and the distance has to be 0. For example, when p_{wj} has not been reached, $||p_k - p_{wj}||_2^2 > 0$, so $\mu_k = 0$, which is in the incomplete state.

4.1.2. *MPC Tracking.* Since the quadrotor system is under-actuated, and its dynamics in (2) are highly nonlinear and coupled. To reduce computational demands, we linearize the system along the planned time-optimal trajectory at each reference point $(\mathbf{x}_{\text{ref}}, \mathbf{u}_{\text{ref}})_k$. This process yields time-varying linearized matrices $A(t)$ and $B(t)$, resulting in the following linearized system:

$$\begin{aligned} \dot{\bar{\boldsymbol{x}}} &= \frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\Big|_{x(t)_{\text{r}}, u(t)_{\text{r}}} \bar{\boldsymbol{x}}(t) + \frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\Big|_{x(t)_{\text{r}}, u(t)_{\text{r}}} \bar{\boldsymbol{u}}(t) \\ &= A(t)\bar{\boldsymbol{x}}(t) + B(t)\bar{\boldsymbol{u}}(t) \end{aligned} \tag{4}$$

Where $\bar{\boldsymbol{x}}(t) = \boldsymbol{x}(t) - \boldsymbol{x}_r(t)$ and $\bar{\boldsymbol{u}}(t) = \boldsymbol{u}(t) - \boldsymbol{u}_r(t)$ are defined in error coordinates. To perform simulations, we discretized the linearized system matrices $A(t)$ and $B(t)$. This discretization results in a discrete-time linearized dynamic system with continuous-valued state and input variables. The system dynamics are expressed as:

$$\bar{\boldsymbol{x}}_{k+1} = A_{d,k}\bar{\boldsymbol{x}}_k + B_{d,k}\bar{\boldsymbol{u}}_k \tag{5}$$

Where k is discrete time index. The linearized system dynamics are incorporated as dynamic constraints within the MPC tracking framework to form a linear MPC problem. The primary objective of the MPC tracking controller is to

minimize the deviation between the current state and the reference state. This is achieved through a simplified quadratic cost function, which sums the squared differences between the predicted states/inputs and the reference states/inputs over the N prediction horizons:

$$\begin{aligned} \underset{\{\bar{\boldsymbol{u}}_k\}_{k=0}^{N-1},\{\bar{\boldsymbol{x}}_k\}_{k=0}^{N}}{\text{minimize}} \quad & J = \sum_{k=0}^{N-1} \left(\bar{\boldsymbol{x}}_k^\top Q \bar{\boldsymbol{x}}_k + \bar{\boldsymbol{u}}_k^\top R \bar{\boldsymbol{u}}_k\right) + \bar{\boldsymbol{x}}_N^\top Q_f \bar{\boldsymbol{x}}_N \\ \text{subject to} \quad & \bar{\boldsymbol{x}}_0 = \boldsymbol{x}_{\text{int}} - \boldsymbol{x}_{\text{ref,0}} \\ & \bar{\boldsymbol{x}}_{k+1} = A_{d,k}(t)\bar{\boldsymbol{x}}_k + B_{d,k}(t)\bar{\boldsymbol{u}}_k \\ & \boldsymbol{u}_{\text{min}} - \boldsymbol{u}_{\text{ref,k}} \leq \bar{\boldsymbol{u}}_k \leq \boldsymbol{u}_{\text{max}} - \boldsymbol{u}_{\text{ref,k}} \\ & \boldsymbol{x}_{\text{min}} - \boldsymbol{x}_{\text{ref,k}} \leq \bar{\boldsymbol{x}}_k \leq \boldsymbol{x}_{\text{max}} - \boldsymbol{x}_{\text{ref,k}} \end{aligned} \tag{6}$$

Where $Q \succeq 0$, $R \succ 0$, and $Q_f \succeq 0$ are the weighting matrices. It should be noted that the decision variables in this optimization problem are the errors in the state and control inputs.

Our goal is to determine the optimal control input that drives the quadrotor along a time-optimal trajectory by solving the real-time quadratic programming (QP) problem presented in equation (6) at each control time step. This optimization process yields an optimal control sequence $\bar{\boldsymbol{u}}^* = \{\bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_1, \ldots, \bar{\boldsymbol{u}}_{N-1}\}$ from which only the first control input, $\bar{\boldsymbol{u}}_0$, is applied to the quadrotor. Hence, our final output control command is defined as:

$$\boldsymbol{u}_k^* = \boldsymbol{u}_{\text{ref},k} + \bar{\boldsymbol{u}}_0 \tag{7}$$

By executing $\boldsymbol{u}_k^*$, the quadrotor transitions to the next state, and the optimization process is subsequently repeated based on this updated state.

4.2. **Actor Critic Reinforcement Learning.** We implement the Actor-Critic RL method, which leverages both policy gradients and value function estimation to achieve stable and efficient learning.

In the Actor-Critic framework, two principal components work in tandem to optimize the agent's behavior. The **Actor**, represented by a parameterized policy $\pi_\theta(a|s)$, selects actions a based on the current state s while the **Critic**, represented by a parameterized value function $V_\phi(s)$, evaluates how favorable a state is under the current policy. This dual approach allows the Actor to improve its policy using value estimates provided by the Critic, thereby reducing the variance in the gradient updates and improving learning stability.

The goal of the Actor is to maximize the expected return $J(\theta)$, defined as:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] \tag{8}$$

where τ denotes a trajectory, $\gamma \in [0, 1]$ is the discount factor, and $r(s_t, a_t)$ is the immediate reward. To achieve this, the Actor updates its parameters θ using the policy gradient:

$$\nabla_\theta J(\theta) \approx \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) A_t\right], \tag{9}$$

where A_t is the advantage function that measures how much better or worse an action is compared to the baseline value provided by the Critic. Specifically, the advantage function is given by:

$$A_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t). \tag{10}$$

The Critic, in turn, learns to approximate the true value function by minimizing the Mean Squared Error (MSE) between the predicted value $V_\phi(s_t)$ and the observed returns. The Critic's objective function is (11).

$$\min_\phi \mathbb{E}\left[\left(r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)\right)^2\right]. \tag{11}$$

This update ensures that the Critic provides accurate estimates of the expected returns, which the Actor uses to inform its policy updates.

The Actor-Critic training process proceeds iteratively. At each step, a trajectory is sampled by executing the current policy π_θ in the environment. The advantage A_t is then computed based on the rewards and the Critic's estimates. The Critic's parameters ϕ are updated by minimizing the MSE loss, while the Actor's parameters θ are updated in the direction suggested by the policy gradient. This iterative process continues until the policy converges to an optimal solution.

5. EXPERIMENT RESULTS

In this section, we present a racing map featuring eight gates designed to evaluate the performance of our two methods. The quadrotor used in our experiments is lightweight, with a mass of 0.03 kg and an arm length of 0.0397 meters. Its moment of inertia is represented by a diagonal matrix with components $(1.4 \times 10^{-5},\ 1.4 \times 10^{-5},\ 2.17 \times 10^{-5})\ \mathrm{kg} \cdot \mathrm{m}^2$, indicating symmetric inertia in the roll and pitch axes and slightly larger inertia in the yaw axis. The thrust-to-weight ratio (TWR) is set to 2.5, ensuring sufficient control authority during aggressive maneuvers. The torque coefficient c_τ, which relates motor thrust to the resulting torque, is 2.51×10^{-2}.

To generate the reference trajectory, we established the gate positions as illustrated in Fig. 1 to serve as waypoint constraints. To ensure the shortest possible lap time, we positioned the start and end points outside the loop. We then applied the planning algorithm from [3], generating a 400-node full-state reference trajectory in about 20 minutes, as illustrated by the purple line in Fig. 1.

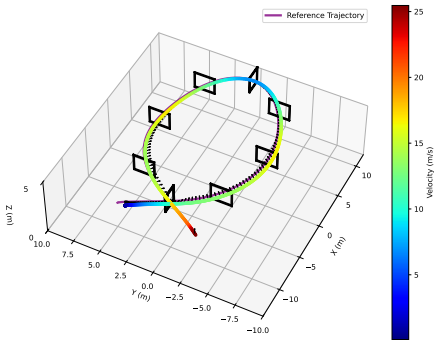

FIGURE 1. Circle track for the linear MPC tracking controller. The precomputed reference trajectory is shown in purple, and black arrows indicate orientation for quadrotor.

We implement the tracking algorithm using the CVXPY toolbox [5] together with the OSQP [6] solver. In our Python simulation environment, we integrate the system dynamics described in (2) using the RK4 method to obtain the quadrotor's next state. The simulation runs at around 300 Hz, while the control commands are updated at around 55 Hz. At each reference point, we numerically compute the time-varying Jacobian matrices $A(t)$ and $B(t)$, and then discretize them for subsequent control steps. The MPC prediction horizon is set to $N = 10$. Table 1 presents the controller's performance. To determine the terminal cost matrix Q_f, we solve a continuous-time algebraic Riccati equation.

	Lap Time (s)	Position RMSE (m)
Linear MPC Tracking	4.812	0.366

TABLE 1. Performance Metrics for Linear MPC Tracking

5.1. **Actor-Critic RL.** We implemented the Actor-Critic RL method in a simulated drone racing environment using PyBullet. The observation space for the RL agent includes two main components: the **drone-state observation** and the **track observation**. The **drone-state observation** is a 6-dimensional vector $[v_x, v_y, v_z, \text{roll}, \text{pitch}, \text{yaw}]$, where $\mathbf{v}$ represents the drone's velocity in the local frame. The **track observation** comprises $[\delta\mathbf{p}_1, \delta\mathbf{p}_2, \ldots, \delta\mathbf{p}_N]$, where $\delta\mathbf{p}_i$ is a 12-dimensional vector containing the coordinates of the four corners of the i-th gate relative to the drone's local frame. In our experiments, we set $N = 2$, meaning the agent considers information about the next two gates.

The action space consists of the RPM commands for the drone's four propellers $[a_1, a_2, a_3, a_4]$. The reward function incentivizes progress toward the current gate while penalizing deviations and excessive control efforts. It is defined as:

$$r(k) = \|\mathbf{g}_k - \mathbf{p}_{k-1}\| - \|\mathbf{g}_k - \mathbf{p}_k\| - b\|\boldsymbol{\omega}_k\|, \tag{12}$$

where $\mathbf{g}_k$ is the position of the current gate, $\mathbf{p}_k$ and $\mathbf{p}_{k-1}$ are the drone's position at the current and previous step respectively. We try to stabilize the drone by penalizing large body rate $b\|\boldsymbol{\omega}_k\|$ with a coefficient $b = 0.01$. Whenever

the drone flies through the current gate, an additional $[r(k) = +10.0]$ is rewarded, and the current gate will be iterated to the next gate. The episode will be terminated upon any collision and a negative reward [r(k) = -10.0] will be imposed.

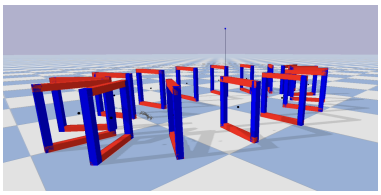

FIGURE 2. The drone racing training gym with 16 gates in an ellipse in Pybullet

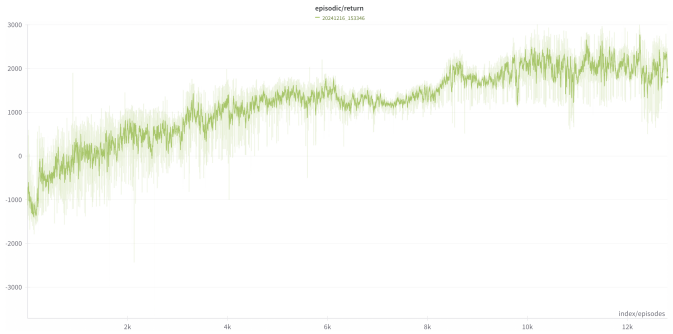

FIGURE 3. The episodic training return

The RL training environment in simulation is in Fig. 2. The episodic return gradually increases over time as shown in Fig.3, demonstrating the agent's ability to learn effective gate navigation. After approximately 12k training episodes, the drone successfully passes through nine gates but collides near the tenth, indicating the need for further training. Fig.4 illustrates some of the drone's flying data in the evaluation episode: The drone can progressively fly through nine gates, about two third of the whole ellipse. This demonstrates the effectiveness of RL learning.

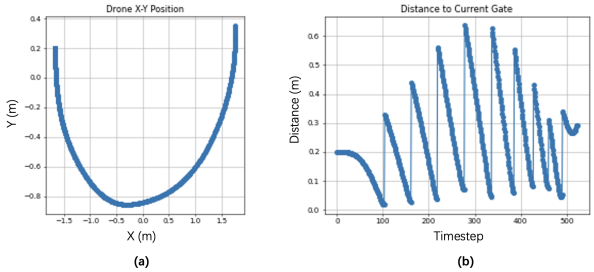

FIGURE 4. **(a)** The X-Y trace of the drone in evaluation episode. **(b)** The distance of the drone to the target gate in evaluation episode.

6. DISCUSSION

Traditional optimization-based methods, such as MPC, typically decompose the task into separate planning and control layers(trajectory serves as an intermediate representation). This modular structure approach improves interpretability and simplifies real-time decision-making. Our results show that optimization-based methods achieve satisfactory performance. However, the controller's objectives do not always directly align with the task goal, and treating MPC purely as a trajectory optimization leads to nonlinear problems that are computationally challenging to solve in real time. In our implementation of linear MPC, which relies on initial conditions and small-angle assumptions, this simplifies calculations but fails to capture system nonlinearities, which introduces challenges for stability and accuracy due to the inherent non-convexity.

In contrast, RL learns control strategies directly from interaction, bypassing the need for a reference trajectory or an explicit dynamic model. This approach offers flexibility and reduces the risk of suboptimal solutions. Our preliminary results show that RL-trained drones quickly overfit to a single motion primitive within a few hundred thousand steps. In contrast, before it can execute a reliable turning manoeuvre, it will require 5–6 million steps due to hardware constraints. In the future, we will adopt the parallel sampling and distributed initialization strategies from [7] to accelerate training, improve task completion, and enhance generalization to unseen tracks.

To address these challenges, the state-of-the-art Actor-Critic MPC algorithm [4] integrates RL with MPC. It treats MPC as an optimization problem with a learnable, time-varying cost function determined by the RL policy. This approach "softens" the MPC constraints, improving the likelihood of achieving optimal solutions while maintaining interpretability and accelerating training convergence.

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.

[2] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3340–3356, 2022.

[3] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, 2021.

[4] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," *IEEE International Conference on Robotics and Automation*, 2024.

[5] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[6] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020. [Online]. Available: https://doi.org/10.1007/s12532-020-00179-2

[7] 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)*. IEEE, 2021, pp. 1205–1212.