

Supplementary Materials for
**Reaching the limit in autonomous racing: Optimal control versus
reinforcement learning**

Yunlong Song *et al.*

Corresponding author: Yunlong Song, song@ifi.uzh.ch

Sci. Robot. **8**, eadg1462 (2023)
DOI: 10.1126/scirobotics.adg1462

The PDF file includes:

Figs. S1 and S2
Tables S1 to S3
Legends for movies S1 to S4
References (40–51)

Other Supplementary Material for this manuscript includes the following:

Movies S1 to S4

Supplementary Methods

Quadrotor Dynamics

The quadrotor is modeled as a rigid body that is actuated by four motors. The quadrotor state is described using a state vector $\mathbf{s} = [\mathbf{p}_W, \mathbf{q}_W, \mathbf{v}_W, \boldsymbol{\omega}_B]$, where W and B indicate the world frame and body frame separately. Here, \mathbf{p}_W is the position, \mathbf{q}_W is a unit quaternion, \mathbf{v}_W is the linear velocity, and $\boldsymbol{\omega}_B$ is the body rates. We use the following dynamical model to describe the quadrotor's motion:

$$\begin{aligned}\dot{\mathbf{p}}_W &= \mathbf{v}_W & \dot{\mathbf{v}}_W &= \mathbf{q}_W \odot \mathbf{c} - \mathbf{g} - \mathbf{R}\mathbf{D}\mathbf{R}^T \mathbf{v}_W \\ \dot{\mathbf{q}}_W &= \frac{1}{2} \boldsymbol{\Lambda}(\boldsymbol{\omega}_B) \cdot \mathbf{q}_W & \dot{\boldsymbol{\omega}}_B &= \mathbf{J}^{-1}(\boldsymbol{\eta} - \boldsymbol{\omega}_B \times \mathbf{J}\boldsymbol{\omega}_B)\end{aligned}$$

where $\mathbf{D} = \text{diag}(d_x, d_y, d_z)$ is a constant diagonal matrix that defines the rotor-drag coefficients and \mathbf{R} is the rotation matrix. Here, $\mathbf{g} = [0, 0, -g_z]^T$ with $g_z = 9.81 \text{ m/s}^2$ is the gravity vector, and $\boldsymbol{\Lambda}(\boldsymbol{\omega}_B)$ is a skew-symmetric matrix. Moreover, $\mathbf{c} = [0, 0, c]$ is the mass-normalized thrust vector. The conversion of single rotor thrusts $[f_1, f_2, f_3, f_4]$ to the mass-normalized thrust c and the body torques $\boldsymbol{\eta}$ is formulated as

$$\boldsymbol{\eta} = \begin{bmatrix} \frac{l}{\sqrt{2}}(f_1 - f_2 - f_3 + f_4) \\ \frac{l}{\sqrt{2}}(-f_1 - f_2 + f_3 + f_4) \\ \kappa f_1 - \kappa f_2 + \kappa f_3 - \kappa f_4 \end{bmatrix} \quad (9)$$

$$c = (f_1 + f_2 + f_3 + f_4)/m \quad (10)$$

where m is the quadrotor's mass and l is the arm length. We model the dynamics of a single-rotor thrust as first-order systems $\dot{f} = (f_{\text{des}} - f)/\alpha$ where α is the time-delay constant. We use a 4th-order Runge-Kutta method for integrating the dynamic equations.

Hardware Design

Table S1 provides an overview of the hardware component configurations. We build our drone using off-the-shelf components that are easily available. Two types of physical drones have been developed for real-world deployment: the 4s drone and the 6s drone. These names derive from the number of battery cells they use (4 and 6 cells, respectively). The number of battery cells directly affects the maximum voltage applied to the motors and, consequently, the maximum force generated. The 4s drone can generate a maximum force of 34 N, whereas the 6s drone can produce up to 63 N. Compared to the 4s drone, the design of the 6s drone is more agile and compact, with reduced inertia and mass. These improvements result in a significant increase in the maximum thrust-to-weight ratio, from 4.62 to 12.45. To measure the maximum generated force, we used a load cell with a constant voltage supply. During real-world deployment, the actual voltage supply via an onboard battery usually fluctuates, resulting in lower total thrust.

Our quadrotor’s frame consists of two types of carbon fiber, making the vehicle both durable and lightweight. The structural parts, excluding the frame, are custom-designed plastic components (made of PLA and TPU material) created using a 3D printer. PLA is used for most parts because of its stiffness, while TPU is used for impact protection and predetermined breaking points. To drive the quadrotor, we use fast-spinning brushless DC motors in conjunction with two types of 5.1-inch three-bladed propellers. Lithium-polymer batteries are employed to meet the high power demands of the motors. The motors are powered by an electronic speed controller that comes in a compact form and supports the DShot protocol for motor speed feedback. We use BetaFlight, an open-source software, as our low-level flight controller, as it offers real-time, low-latency control.

Related Work

Given perfect state estimation, the conventional paradigm for autonomous racing is to decompose the solution into two stages: planning and control. For the planning stage, existing approaches in the context of quadrotors can be mostly categorized as polynomial approaches and optimization methods. Polynomial trajectories (41–43) are efficient to generate and can leverage the differential flatness property of quadcopters. Polynomial trajectories are limited to be smooth and are thus not always able to represent optimal control sequences, which may feature sharp mode switches, akin to bang-bang control (32).

Optimization-based trajectory planning allows for independently selecting the optimal sequence of states and control inputs at every time step, considering time minimization while complying with quadrotor dynamics and input constraints (32, 44–46). Foehn et al. (32) introduced a complementary progress constraint (CPC) approach, which considers true actuator saturation, uses single rotor thrusts as control inputs, and exploits quaternions to allow full, singularity-free representation of the orientation space with consistent linearization characteristics. While this method yields the optimal trajectory for a given quadrotor model, it is computationally expensive and is not tractable in real-time.

Previous studies (23,29,30,32,47) have showcased remarkable advancements in autonomous drone racing using model-based control methods. Notably, Foehn et al. (32) demonstrated that it is possible to surpass two professional human pilots by proposing a time-optimal trajectory planning method, and then tracking it with a standard model predictive controller. However, i) the inherent nature of time-optimal trajectories and ii) the separation of the high-level task in planning and control hinder the capabilities of model-based control. The time-optimal trajectory given a dynamic system consists of control commands that lie mostly at the limit of actuation, leaving no extra control authority for the controller to counteract disturbances or model mismatches. As a result, in order to track a time-optimal trajectory, Foehn et al. (32) needed to lower the effective thrust limit to 70% of the maximum available thrust when deploying their system in the real world. For a fair comparison against professional human pilots, the quadrotor platform used for human flights was reduced to the same thrust-to-weight ratio as the autonomous platform (32).

Learning-based approaches to autonomous racing replace the planning and control stack with a neural network (40, 48). Purely data-driven control strategies, such as model-free reinforcement learning, aim to circumvent the limitations of model-based controller design by learning effective controllers directly from experience. For example, control of a physical quadrotor using reinforcement learning was demonstrated by Hwangbo et al. (48), who used a neural network policy to track waypoints and recover from challenging initialization. Koch et al. (49) applied model-free RL to low-level attitude control and showed that a learned low-level controller trained with PPO outperformed a fully tuned PID controller on almost every metric. Song et al. (50) used deep RL to generate near-time-optimal trajectories for autonomous drone racing and tracked the trajectories by an MPC controller. Lambert et al. (51) used model-based RL to train a hovering controller.

Supplementary Figures

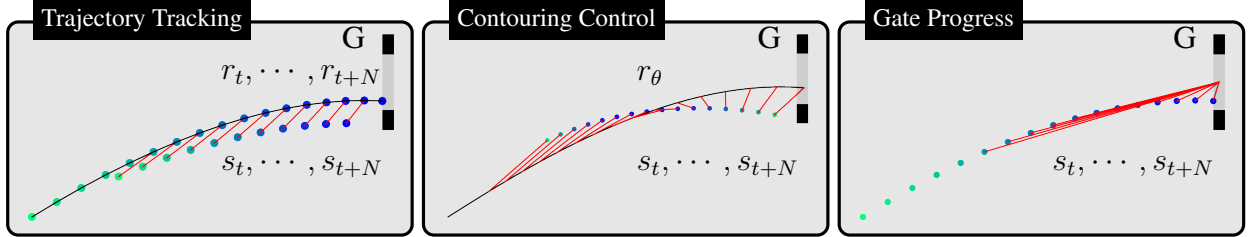


Figure S1: A visualization of three optimization objectives for autonomous drone racing. All objectives aim to achieve minimum-time flight through the track.

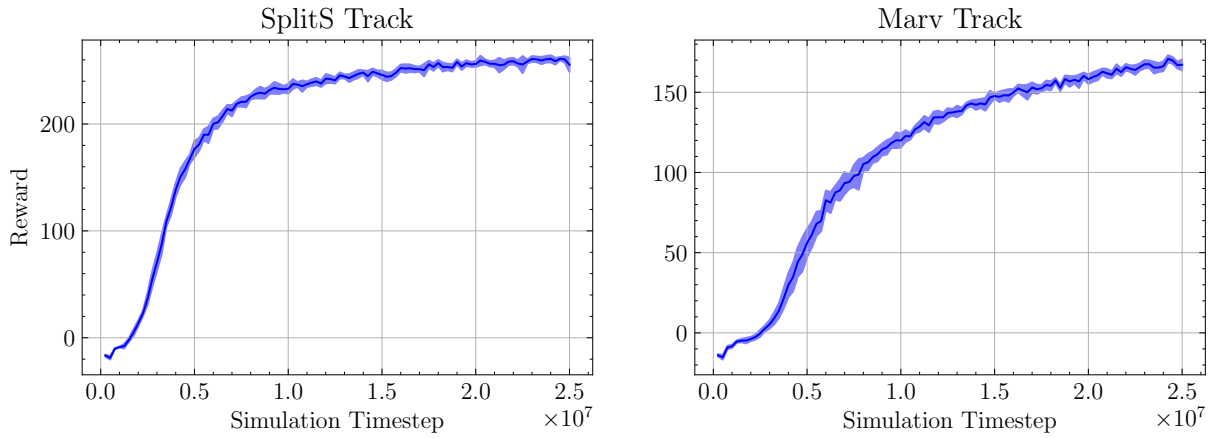


Figure S2: **Learning curves for gate progress maximization, averaged across 10 runs with different seeds.** We compute the mean (solid line) and standard deviation (shadow region). Each policy is trained for a total of 25 million simulation timesteps. The total training time for every single policy takes around 10 minutes. We use a desktop computer equipped with an Intel i9-9900K CPU and an Nvidia Titan RTX graphics card.

Supplementary Tables



<div> <div>4s Drone</div>  </div>		<div> <div>6s Drone</div>  </div>	
Parameters	4s Drone	6s Drone	
Thrust-to-weight Ratio	4.62	12.45	
Mass [kg]	0.75	0.52	
Maximum Thrust [N]	34.00	63.76	
Arm Length [m]	0.15	0.11	
Inertia [g m^2]	[2.50, 2.51, 4.32]	[1.37, 1.34, 2.45]	
Motor Time Constant [s]	0.033	0.020	
Components	4s Drone	6s Drone	
Frame	Chameleon 6 inch	BMSRacing JS-1	
Motor	Xrotor 2306	BMS Raptor V2	
Propeller	Azure Power SFP 5148 5	HQ Prop R42	
Flight Controller	BrainFPV radix	Foxeer Mini F722	
Battery	Tattu 4 cells 1800 mAh	Tattu 6 cells 1300 mAh	
Electronic Speed Controller (ESC)	Hobbywing XRotor	Foxeer Reaper Mini 128K	

Table S1: **Overview of the racing drone configurations for real-world experiments.** The 4s Drone is used for the baseline comparison between our RL policy and optimal control methods. The 6s Drone is used for pushing the limit of our RL controller in the physical world and for the competition against human pilots.

Parameter	Value
learning rate	0.0003
discount factor	0.99
GAE- λ	0.95
learning epoch	10
policy network	MLP [512, 512]
value network	MLP [512, 512]
clip range	0.2
entropy coefficient	0.005
batch size	25000

Table S2: **PPO hyperparameters.** We train the policy using Proximal Policy Optimization (PPO). We simulate 100 environments in parallel to collect rollouts. We use the running mean and standard deviation to normalize the observation. MLP represents a multilayer perceptron network, and GAE represents generalized advantage estimate

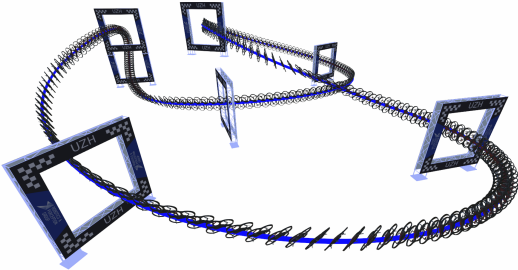
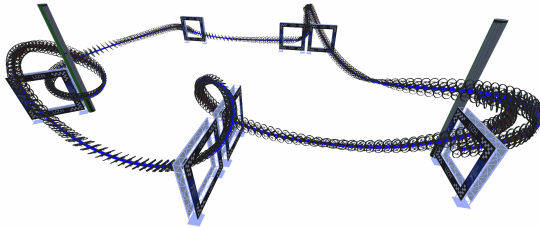
Split-S Track		Marv Track	
			
Split-S Track	Gate Center (x, y, z)	Orientation (q_w, q_x, q_y, q_z)	Depth \times Width \times Height (m)
Gate 1	$[-0.70, -0.89, 3.53]$	$[0.96, 0.0, 0.0, -0.27]$	$[0.2 \times 1.5 \times 1.5]$
Gate 2	$[9.17, 6.25, 1.12]$	$[1.0, 0.0, 0.0, 0.0]$	$[0.2 \times 1.5 \times 1.5]$
Gate 3	$[8.90, -3.81, 1.12]$	$[-0.44, 0.0, 0.0, 0.90]$	$[0.2 \times 1.5 \times 1.5]$
Gate 4	$[-4.40, -5.77, 3.20]$	$[0.0, 0.0, 0.0, 1.0]$	$[0.2 \times 1.5 \times 1.5]$
Gate 5	$[-4.40, -5.79, 1.12]$	$[1.0, 0.0, 0.0, 0.0]$	$[0.2 \times 1.5 \times 1.5]$
Gate 6	$[4.48, -0.46, 1.12]$	$[0.67, 0.0, 0.0, 0.74]$	$[0.2 \times 1.5 \times 1.5]$
Gate 7	$[-1.94, 6.82, 1.12]$	$[-0.26, 0.0, 0.0, 0.97]$	$[0.2 \times 1.5 \times 1.5]$
Marv Track	Gate Center (x, y, z)	Orientation (q_w, q_x, q_y, q_z)	Depth \times Width \times Height (m)
Gate 1	$[7.70, 11.46, 1.12]$	$[0.72, 0.0, 0.0, -0.70]$	$[0.2 \times 1.5 \times 1.5]$
Gate 2	$[7.57, 3.97, 1.12]$	$[0.71, 0.0, 0.0, -0.70]$	$[0.2 \times 1.5 \times 1.5]$
Gate 3	$[9.88, 3.97, 1.12]$	$[0.72, 0.0, 0.0, -0.70]$	$[0.2 \times 1.5 \times 1.5]$
Gate 4	$[7.15, -4.78, 1.12]$	$[-0.33, 0.0, 0.0, 0.94]$	$[0.2 \times 1.5 \times 1.5]$
Gate 5	$[-5.51, -4.32, 1.12]$	$[1.0, 0.0, 0.0, 0.0]$	$[0.2 \times 1.5 \times 1.5]$
Gate 6	$[-4.60, 4.53, 1.12]$	$[1.0, 0.0, 0.0, 0.0]$	$[0.2 \times 1.5 \times 1.5]$
Gate 7	$[-4.60, 6.79, 1.12]$	$[1.0, 0.0, 0.0, 0.0]$	$[0.2 \times 1.5 \times 1.5]$

Table S3: Race track configurations. Both tracks have in total of 7 gates, but the gates are arranged in different configurations. We consider all gates to have a rectangular shape with the same dimension. The Split-S track contains two vertically stacked gates, which require a so-called Split-S manoeuvre. The Marv track features more challenging maneuvers commonly used in FPV drone racing, including one Split-S manoeuvre, two power-loop manoeuvres, one ladder manoeuvre, and one hairpin manoeuvre.

Supplementary Movies

Movie S1. An overview of the methodology and results.

Movie S2. An animation of policy training in simulation.

Movie S3. Deployment of the reinforcement learning policy in the SplitS track.

Movie S4. Deployment of the reinforcement learning policy in the Marv track.