

MPCC++: Model Predictive Contouring Control for Time-Optimal Flight with Safety Constraints

Maria Krinner,^{*1} Angel Romero,^{*2} Leonard Bauersfeld,² Melanie Zeilinger,¹ Andrea Carron,¹ Davide Scaramuzza²

¹ Institute for Dynamics Systems and Control, ETH Zurich, Switzerland

² Robotics and Perception Group, University of Zurich, Switzerland

*These authors contributed equally

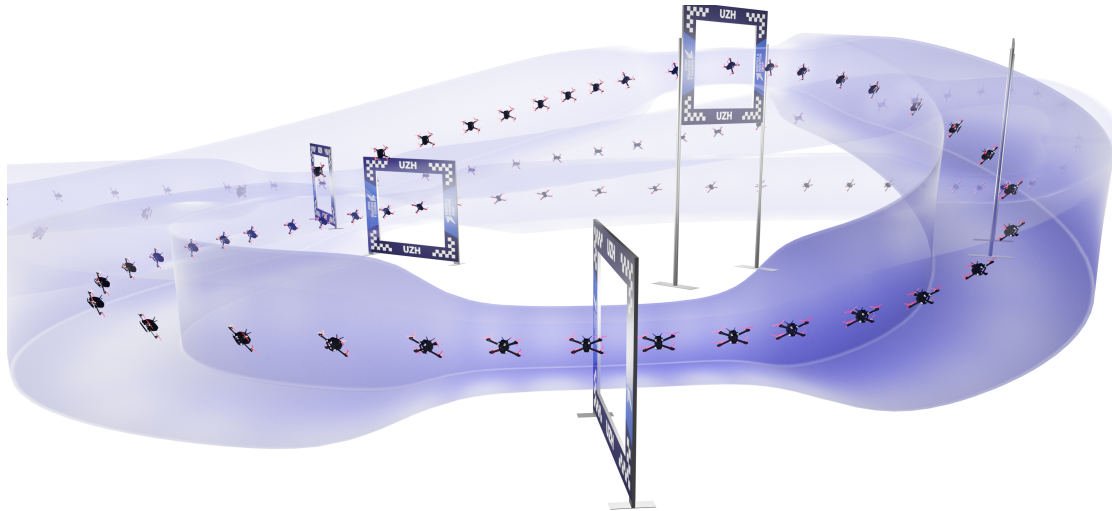


Fig. 1: The proposed method is able to fly a quadrotor at the limit of handling while ensuring safety. In the figure, the tunnel constraint is shown in translucent blue, and it expands in gate-free regions and shrinks at the gate passages, avoiding gate collisions. We are able to fly at speeds of more than 80 km/h while consistently preventing gate collisions and achieving 100% success rate in real-world experiments.

Abstract—Quadrotor flight is an extremely challenging problem due to the limited control authority encountered at the limit of handling. Model Predictive Contouring Control (MPCC) has emerged as a promising model-based approach for time optimization problems such as drone racing. However, the standard MPCC formulation used in quadrotor racing introduces the notion of the gates directly in the cost function, creating a multi-objective optimization that continuously trades off between maximizing progress and tracking the path accurately. This paper introduces three key components that enhance the state-of-the-art MPCC approach for drone racing. First and foremost, we provide safety guarantees in the form of a track constraint and terminal set. The track constraint is designed as a spatial constraint which prevents gate collisions while allowing for time optimization only in the cost function. Second, we augment the existing first principles dynamics with a residual term that captures complex aerodynamic effects and thrust forces learned directly from real-world data. Third, we use Trust Region Bayesian Optimization (TuRBO), a state-of-the-art global Bayesian Optimization algorithm, to tune the hyperparameters of the MPCC controller given a sparse reward based on lap time minimization. The proposed approach achieves similar lap times to the best-performing RL policy and outperforms the best model-based controller while satisfying constraints. In both simulation and real world, our approach consistently prevents gate crashes with 100% success rate, while pushing the quadrotor to its physical limits reaching speeds of more than 80km/h.

SUPPLEMENTARY MATERIAL

A narrated video with real-world experiments is available at: <https://youtu.be/sbKe9emghtM>

I. INTRODUCTION

The last decade has seen a remarkable growth in the number of quadrotor applications. Many missions are time-critical, such as search and rescue, aerial delivery, flying cars, or space exploration [1, 2, 3]. Among them, drone racing has emerged as a testbed for research on time-optimal flight. Quadrotors are carefully engineered to push the limits of speed and agility, making drone racing an ideal benchmark for aerial performance and safety at high speeds [4, 5]. State-of-the-art approaches to autonomous drone racing can be categorized into learning-based and optimization-based methods.

Within learning-based methods, Reinforcement Learning (RL) has arisen as an attractive alternative to conventional planning and control algorithms, outperforming world-champion pilots [6, 7]. Unlike optimal control, RL optimizes a controller using data sampled from numerous trial-and-error interactions with the environment. This approach can manage sparse objectives and unstructured observation spaces, providing substantial flexibility and versatility in the controller

design. As shown in [7], RL can directly optimize a task-level objective, eliminating the need for explicit intermediate representations such as trajectories. Furthermore, it leverages domain randomization in simulation to cope with model uncertainty, allowing the discovery of more robust control responses. These advantages have facilitated numerous breakthroughs in pushing quadrotor systems to their operational limits. However, despite its empirical success, RL policies lack theoretical guarantees. Integrating safety considerations into learning-based frameworks remains an area of ongoing research [8, 9]. The combination of optimization-based and learning-based architectures to enhance safety is emerging as a prominent topic within the robotics community [10, 11, 12].

Optimization-based architectures have tackled the problem from a different perspective. In [13], the authors propose a method to find time-optimal trajectories through a predefined set of waypoints and show how they outperform expert drone-racing pilots. However, these trajectories take several hours to calculate, rendering them impractical for replanning in the face of model mismatch and unknown disturbances (e.g., drone model, gate positions, aerodynamic effects, wind gusts). Addressing this challenge necessitates an algorithm capable of generating time-optimal trajectories in real-time. In [14, 15], a Model Predictive Contouring Control (MPCC) method is introduced to track non-feasible paths by maximizing progress along the designated path in a manner akin to time-optimal strategies. By solving the complex time-allocation problem online at every time step, MPCC selects the optimal states and inputs that enhance progress. This approach results in a controller that demonstrates great adaptability in response to model mismatches and unknown disturbances.

Several modifications to the MPCC cost function are necessary to adapt the formulation to the drone racing task. In [14], the concept of gates is introduced in the cost function by parameterising the contour weight as a multivariate Gaussian function. Specifically, this adaptation employs a Gaussian function at each gate to scale the contour weight in the proximity of the gates. This ensures that the drone closely follows the reference path when navigating through the gate. However, this approach presents several shortcomings. First, employing a multivariate Gaussian to model the contour weight significantly increases the number of hyperparameters, making manual tuning a highly challenging task [16]. Secondly, while this contour function encourages the drone’s proximity to the designated path, it does not explicitly prevent gate collisions, resulting in a sub-optimal trade-off between performance and safety. Moreover, the contour function requires specific tuning for each gate configuration, which restricts its flexibility when gate positions change and increases the dimensionality of the parameter space with the number of gates.

Although this method has resulted in control capabilities surpassing those of human pilots, the approximations in the cost function are exceedingly intricate, suggesting the possibility for more robust and scalable alternatives. Such alternatives would ideally achieve the desired outcomes by imposing constraints that consistently prevent gate collisions, avoiding

the need to handcraft complex, high-parametric cost functions.

Previous works have aimed to tackle this issue by introducing a spatial constraint around the trajectory of the quadrotor [17, 18, 19]. In particular, [19] adapts the quadrotor’s dynamics to a local Frenet-Serret frame, which naturally gives rise to tunnel-like position constraints. However, this spatial reformulation of the dynamics requires using first and second derivatives of the coordinates in the Frenet-Serret frame, which introduces singularities that require careful handling.

Contributions

This paper introduces three key components that enhance the state-of-the-art MPCC formulation for drone racing [14]. First, we provide safety guarantees in the form of a track constraint and terminal set. The track constraint is designed as a spatial constraint that prevents gate collisions in the form of a prismatic tunnel. Unlike [19], our formulation retains the dynamics in their standard Euclidean form, avoiding the complex reformulation into the Frenet-Serret frame and its associated singularities. The terminal set consists of a periodic, feasible trajectory. This combination provides guarantees of recursive feasibility and inherent robustness. Second, we augment the existing first principles dynamics with a residual term that captures complex aerodynamic effects and thrust forces inferred directly from real-world data. Third, we show that using Trust-Region Bayesian Optimization (TuRBO) to tune the hyperparameters of the MPCC controller results in superior performance compared to previous work [16]. In both simulation and real-world experiments, we illustrate how combining these elements improves the controller’s performance and safety beyond the state-of-the-art MPCC. Moreover, the performance of our method aligns with that of the best-performing RL policy, with the added benefit of incorporating safety into our formulation.

II. RELATED WORK

A. High-speed quadrotor flight

The literature on high-speed quadrotor flight is categorized into two primary approaches: model-based and learning-based. A comprehensive survey of the literature on this subject can be found in [5].

The model-based category originates from polynomial planning and classical control techniques. Traditionally, these methods focused on harnessing the differential flatness property of quadrotors and leverage the use of polynomial for planning [20, 21, 22, 23]. More recently, optimization-based methods have achieved planning of time-optimal trajectories using the quadrotor dynamics and numerical optimization [13]. Nonetheless, the substantial computational demand of these methods typically necessitates offline trajectory computation, rendering these approaches impractical for real-time applications. Within model-based approaches, the problem is typically framed either as time minimization or as progress maximization, depending on the optimization problem’s cost function. Time minimization formulations incorporate the time variable directly into the cost function but often require

a spatial reformulation using the Frenet-Serret frame. This introduces several complexities, such as handling the inherent nonlinearities and singularities that arise from the coordinate transformations [24, 25, 26]. Conversely, contouring methods propose employing progress maximization along a path as a proxy for time minimization. This approach simplifies the control problem by allowing for more robust and efficient solutions. This distinction underlines that although progress maximization does not address the time-optimal problem directly, it offers a practical alternative with significant success. Consequently, MPCC has shown promising results in achieving minimum-time flight [14].

A significant benefit of optimization-based methods is their ability to incorporate safety considerations through state and input constraints. Notably, works like [17, 18, 19, 27] have investigated the application of positional constraints in the form of gates or tunnels to prevent collisions.

On the other hand, a collection of learning-based methodologies for autonomous racing have emerged, which aim to replace the traditional planning and control layers with a neural network [28, 29, 30]. These purely data-driven control strategies, such as model-free RL, strive to circumvent the limitations of model-based controllers by learning effective policies directly from interactions with the environment. For instance, the authors in [31] employ a neural network policy for guiding a quadrotor through waypoints and recovering from challenging initialization setups. In [32], model-free RL is employed for low-level attitude control, showing that a learned low-level controller trained with Proximal Policy Optimization (PPO) outperformed a fully tuned PID controller. In [33], model-based RL is used to train a hovering controller. The family of learning-based approaches is rapidly advancing, fueled by recent breakthroughs in quadrotor simulation environments. These advancements have resulted in test environments that facilitate the training, assessment, and zero-shot transfer of control policies to the real world. The state-of-the-art on realistic simulations for quadrotors is [34], which introduces a hybrid aerodynamic quadrotor model that combines Blade Element Momentum Theory (BEM) with learned aerodynamic representations from highly aggressive maneuvers.

The authors of [35] employed deep RL to generate near time-optimal trajectories for autonomous drone racing and tracked the trajectories by an MPC controller. More recently, [6, 7] have demonstrated that policies trained with model-free RL can achieve super-human performance. However, none of these learning-based approaches include safety considerations into their designs.

B. Tunnel MPC

The concept of employing spatial constraints to systematically prevent collisions is well-established in car racing, where tracks are clearly defined at all points. However, drone racing introduces a unique challenge: while the positions of the gates are fixed, the space in between remains open for exploration. Recent advancements in this field have introduced the idea

of using a tunnel around the trajectory as a spatial constraint [17, 18, 19]. This method involves a spatial reformulation of the drone’s dynamics into the Frenet-Serret frame, focusing on the transverse distances from the drone’s current position to the track’s centerline. Such a state definition, while providing a natural mechanism to enforce tunnel constraints, necessitates a conversion of Euclidean coordinates into terms dependent on these transverse distances. Although this approach elegantly integrates the spatial constraints within the drone’s control framework and has shown promising results in simulations [19], including high-speed navigation across various tracks, it also presents significant challenges due to the spatial reformulation required. Specifically, it introduces singularities that must be carefully addressed to maintain a smooth and reliable flight trajectory.

C. Data-driven models for MPC

Learning-based MPC [36, 37, 38, 39, 40, 41, 42, 43, 44] utilize real-world data to refine dynamic models or learn an objective function tailored for MPC applications. These strategies typically focus on learning dynamics for tasks where deriving an analytical model of the robot or their operational environments poses significant challenges. This is particularly relevant for highly dynamic tasks, such as aggressive autonomous driving around a loose-surface track, exemplified in [45].

Another promising line of research is the development of specialized solvers designed for use within a learning-based MPC framework [46]. This direction takes advantage of zero-order Sequential Quadratic Programming (SQP) techniques, potentially enhancing the efficiency of the control solutions for complex tasks. This includes adapting to unpredictable elements and optimizing performance over a wide range of operational conditions.

D. Automatic controller tuning

The classic approach [47] for controller tuning analytically finds the relationship between a performance metric, e.g. tracking error or trajectory completion, and the controller parameters. It then optimizes the parameters with gradient-based methods [48, 49, 50]. However, expressing the long term performance metric, such as the lap time, as a function of the tuning parameters is impractical and generally intractable. There exist different methods for automatically tuning controllers [51, 52, 53, 54, 55, 56]. RL methods [35] utilize trial-and-error to adapt controller settings to complex system dynamics without the need for explicit modeling. Similarly, BO [57] leverages probabilistic models to navigate and explore the parameter space of black-box functions, effectively identifying optimal settings with minimal data. Furthermore, Weighted Maximum Likelihood Estimation (WML) [16] employs policy search techniques to iteratively refine parameter estimates from observed data, aiming to pinpoint the most effective parameters for the cost function. These methods provide a range of techniques for controller tuning, each with its own set of strengths and limitations.

III. PRELIMINARIES

In this section we introduce the nominal quadrotor dynamics model and present the basics of the MPCC algorithm from [14].

A. Quadrotor Dynamics

In this section, we describe the nominal dynamics $\mathbf{f}(\mathbf{x}, \mathbf{u})$ where $\mathbf{x} \in \mathbb{R}^{13}$ is the state of the quadrotor and $\mathbf{u} \in \mathbb{R}^4$ is the input to the system. The state of the quadrotor is given by $\mathbf{x} = [\mathbf{p}_I, \mathbf{q}_{IB}, \mathbf{v}_I, \boldsymbol{\omega}_B]^T$, where $\mathbf{p}_I \in \mathbb{R}^3$ is the position, $\mathbf{q}_{IB} \in \mathbb{SO}(3)$ is the unit quaternion that describes the rotation from the body to the inertial frame, $\mathbf{v}_I \in \mathbb{R}^3$ is the linear velocity vector, and $\boldsymbol{\omega}_B \in \mathbb{R}^3$ are the bodyrates in the body frame B . For ease of readability, we drop the frame indices, as they remain consistent throughout the description. The nominal dynamic equations are given by:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} & \dot{\mathbf{v}} &= \mathbf{g} + \frac{\mathbf{R}(\mathbf{q})\mathbf{f}_T}{m} \\ \dot{\mathbf{q}} &= \frac{\mathbf{q}}{2} \odot [0 \ \boldsymbol{\omega}]^T & \dot{\boldsymbol{\omega}} &= \mathbf{J}^{-1}(\boldsymbol{\tau}_T - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}) \end{aligned} \quad (1)$$

where \odot represents the Hamilton quaternion multiplication, $\mathbf{R}(\mathbf{q})$ the quaternion rotation, m the quadrotor's mass, \mathbf{J} the quadrotor's inertia, \mathbf{f}_T the collective thrust, and $\boldsymbol{\tau}_T$ the body torques. The input space, given by \mathbf{f}_T and $\boldsymbol{\tau}_T$, is further decomposed into single rotor thrusts $\mathbf{f} = [f_1, f_2, f_3, f_4]$ as follows:

$$\mathbf{f}_T = \begin{bmatrix} 0 \\ 0 \\ \sum f_i \end{bmatrix} \quad \text{and} \quad \boldsymbol{\tau}_T = \begin{bmatrix} l/\sqrt{2}(f_1 + f_2 - f_3 - f_4) \\ l/\sqrt{2}(-f_1 + f_2 + f_3 - f_4) \\ c_\tau(f_1 - f_2 + f_3 - f_4) \end{bmatrix} \quad (2)$$

where f_i is the i -th rotor's thrust, l the quadrotor's arm length, and c_τ the rotor's torque constant.

In addition to the nominal dynamics, we introduce the full dynamic model $\hat{\mathbf{f}}(\mathbf{x}, \mathbf{u}) = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathbf{g}(\mathbf{x}, \mathbf{u})$, which is augmented with a residual term $\mathbf{g}(\mathbf{x}, \mathbf{u})$ that captures unmodeled terms such as aerodynamic effects. The residual term is inferred from real-world data.

B. Model Predictive Contouring Control

We consider the discrete-time dynamic system of a quadrotor with continuous state and input spaces, $\mathbf{x}_k \in \mathcal{X}$ and $\mathbf{u}_k \in \mathcal{U}$ respectively. We denote the time discretized evolution of the system $\hat{\mathbf{f}}: \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$ such that:

$$\mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) \quad (3)$$

where the index k refers to the states and inputs at time t_k . The general Optimal Control Problem (OCP) considers the task of finding a control policy $\pi(\mathbf{x})$, a map from the current state to the optimal input, $\pi: \mathcal{X} \mapsto \mathcal{U}$, such that the cost function $J: \mathcal{X} \mapsto \mathbb{R}^+$ is minimized:

$$\begin{aligned} \pi(\mathbf{x}) &= \underset{\mathbf{u}}{\operatorname{argmin}} & J(\mathbf{x}) \\ &\text{subject to} & \mathbf{x}_0 = \mathbf{x} \\ & & \mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) \\ & & \mathbf{x}_k \in \mathcal{X}, \quad \mathbf{u}_k \in \mathcal{U} \end{aligned} \quad (4)$$

In MPCC, the goal is to compromise between maximizing progress along a predefined path, while tracking it accurately. The main ingredients of the cost function are the progress term θ , the contour error $e^c(\theta)$, and the lag error $e^l(\theta)$, which describe the perpendicular and tangential error between the current position and its projection on the reference path:

$$J_{MPCC}(\mathbf{x}) = \sum_{k=0}^N \|e^l(\theta_k)\|_{Q_l}^2 + \|e^c(\theta_k)\|_{Q_c}^2 - \mu v_{\theta_k} \quad (5)$$

The resulting OCP is formulated as follows:

$$\begin{aligned} \pi(\mathbf{x}) &= \underset{\mathbf{u}}{\operatorname{argmin}} & \sum_{k=0}^N \|e^l(\theta_k)\|_{Q_l}^2 + \|e^c(\theta_k)\|_{Q_c}^2 + \|\boldsymbol{\omega}_k\|_{Q_\omega}^2 \\ & & + \|v_{\theta_k}\|_{R_{v_\theta}}^2 + \|\Delta \mathbf{f}_k\|_{R_{\Delta f}}^2 - \mu v_{\theta_k} \\ &\text{subject to} & \mathbf{x}_0 = \mathbf{x} \\ & & \mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) \\ & & \underline{\boldsymbol{\omega}} \leq \boldsymbol{\omega} \leq \overline{\boldsymbol{\omega}} \\ & & \underline{\mathbf{f}} \leq \mathbf{f} \leq \overline{\mathbf{f}} \\ & & 0 \leq v_\theta \leq \overline{v_\theta} \\ & & \underline{\Delta \mathbf{f}} \leq \Delta \mathbf{f} \leq \overline{\Delta \mathbf{f}} \end{aligned} \quad (6)$$

where v_θ is the first derivative of θ with respect to time. We model the progress θ as a first order system to penalize the variation in progress in the cost, which provides a much smoother state. The norms of the form $\|\cdot\|_A^2$ represent the weighted Euclidean inner product $\|\mathbf{v}\|_A^2 = \mathbf{v}^T \mathbf{A} \mathbf{v}$. For more details about this formulation we refer the reader to [14].

IV. METHODOLOGY

In this section we introduce our enhancements to the standard MPCC formulation described in Section III-B.

A. Safety constraints

We first introduce the track constraint as a spatial constraint which consistently prevents gate collisions. We then define a terminal set and show that the proposed controller is recursively feasible.

We define the track constraint as a prismatic tunnel that joins the inner corners of the gates (see Figure 2). We refer to this formulation as MPCC++ to distinguish it from the baseline MPCC introduced in [14]. Two components are required to define such a set: i) a centerline which passes through the gate centers and matches the first derivative with the gate normal; and ii) a parameterization of the width and height of the cross section as a function of the progress θ , i.e. $W(\theta_k)$, $H(\theta_k)$ respectively.

We employ a simple hermetian spline as the centerline. The gate cross section is constructed around the centerline using standard Frenet-Serret formulas, similar to [19]. At every point of the curve, consider the reference frame that consists of the vectors $[\mathbf{t}(\theta_k), \mathbf{n}(\theta_k), \mathbf{b}(\theta_k)]$. Let \mathbf{p}_k be the coordinates of the platform at current time k , and $\mathbf{p}^d(\theta_k)$ the corresponding point on the centerline. For the width $W(\theta_k)$ and height $H(\theta_k)$ of the tunnel, define the bottom left corner of the tunnel

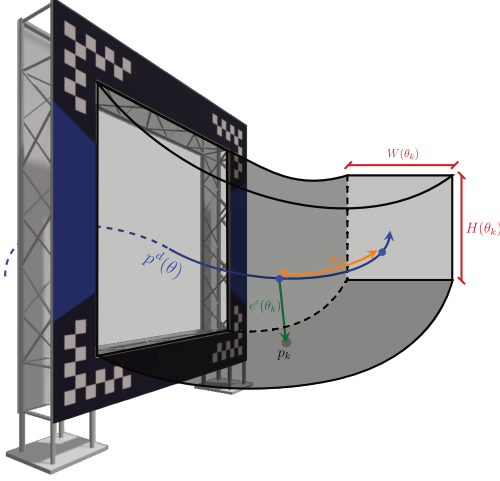


Fig. 2: Spatial constraint forming a tunnel around the centerline. The width and height are parameterized by $W(\theta_k)$ and $H(\theta_k)$, respectively.

as $\mathbf{p}_0(\theta_k) = \mathbf{p}^d(\theta_k) - W(\theta_k) \cdot \mathbf{n}(\theta_k) - H(\theta_k) \cdot \mathbf{b}(\theta_k)$. The boundary of the tunnel is then determined by four halfspace constraints as follows:

$$\begin{aligned} (\mathbf{p}_k - \mathbf{p}_0(\theta_k)) \cdot \mathbf{n}(\theta_k) &\geq 0 \\ 2H(\theta_k) - (\mathbf{p}_k - \mathbf{p}_0(\theta_k)) \cdot \mathbf{n}(\theta_k) &\geq 0 \\ (\mathbf{p}_k - \mathbf{p}_0(\theta_k)) \cdot \mathbf{b}(\theta_k) &\geq 0 \\ 2W(\theta_k) - (\mathbf{p}_k - \mathbf{p}_0(\theta_k)) \cdot \mathbf{b}(\theta_k) &\geq 0 \end{aligned} \quad (7)$$

Furthermore, we set $W(\theta_k) = H(\theta_k)$ and parameterize $W(\theta_k)$ by two distinct values: a nominal value W_n for the broader sections of the tunnel, and an inner gate value W_{gate} for the narrower sections at the gates. A sigmoid function is employed to smoothly transition between these two levels, ensuring a gradual narrowing of the tunnel as it approaches a gate.

We define the terminal set \mathcal{X}_f as a periodic feasible trajectory which passes through the center of the gates, similar to [58]:

$$\mathbf{p}_N \in \mathcal{X}_f \quad (8)$$

where \mathbf{p}_N is the position of the last shooting node \mathbf{x}_N . By making the first and last elements of the trajectory coincide, we establish its periodicity, which ensures that the resulting trajectory is a positive invariant set. We compute this set offline by solving the following optimization problem:

$$\begin{aligned} \underset{\mathbf{x}, \mathbf{u}}{\operatorname{argmin}} \quad & \sum_{k=0}^M \|\mathbf{p}^d(\theta_k) - \mathbf{p}_k\|_2^2 \\ \text{subject to} \quad & \mathbf{x}_0 = \mathbf{x}_M \\ & \mathbf{x}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) \\ & \mathbf{x}_k \in \mathcal{X}, \quad \mathbf{u}_k \in \mathcal{U} \end{aligned} \quad (9)$$

where M is the number of points that parameterize the terminal set, and the first constraint ensures periodicity of the trajectory. The resulting discrete terminal set is obtained from

the solution of Problem 9:

$$\mathcal{X}_f = \{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_M\} \quad (10)$$

where \mathbf{p}_i is the position of the i -th shooting node.

Proposition 1: The MPC formulation in Problem 6 subject to constraints (7) and (8) is recursively feasible and satisfies constraints at all times.

The proof follows from standard MPC recursive feasibility arguments [59]. It is also possible to show that Problem 6 is inherently robust to disturbances by following the same arguments used in [58].

B. Dynamics augmentation

In this section we augment the nominal dynamics $\mathbf{f}(\mathbf{x}, \mathbf{u})$ introduced in Section III-A, with a residual term $\mathbf{g}(\mathbf{x}, \mathbf{u})$ that captures unmodeled effects. We first provide a general overview of the forces and torques that act on the system, then identify the sources of model mismatch, and propose a set of polynomial features to approximate these terms from real-world data.

We differentiate between the lift force \mathbf{f}_{prop} produced by the propellers, and the collective aerodynamic forces \mathbf{f}_{aero} which encompasses effects such as drag, induced lift, and blade flapping. The total torque acting on the system is composed of four elements: i) the torque generated by the individual propellers τ_{prop} ; ii) the yaw torque τ_{mot} arising from changes in motor speeds; iii) the aerodynamic torque τ_{aero} that captures a variety of aerodynamic influences; and iv) the inertial term τ_{iner} .

While \mathbf{f}_{prop} and τ_{prop} can be precisely estimated from first principles, modelling the aerodynamic terms is significantly more challenging. In [34], \mathbf{f}_{prop} and τ_{prop} were modelled analytically using Blade Element Momentum Theory (BEM), which due to its precision, notably raises computational demands. Instead, we follow the structure proposed by [6], in which a simple polynomial model is fitted to real-world data through conventional regression techniques. The polynomial model describes the aerodynamic forces and torques as a linear combination of polynomial features involving the drone's linear velocity (in body-frame) and the mean-squared rotor speed, Ω^2 .

$$\begin{aligned} f_x &= C_{f_x} [v_x \quad v_x^3 \quad \Omega^2 \quad v_x \Omega^2]^T \\ f_y &= C_{f_y} [v_y \quad v_y^3 \quad \Omega^2 \quad v_y \Omega^2]^T \\ f_z &= C_{f_z} [v_z \quad v_z^3 \quad v_{xy} \quad v_{xy}^2 \quad v_{xy} \Omega^2 \quad v_z \Omega^2 \quad v_{xy} v_z \Omega^2]^T \\ \tau_x &= C_{\tau_x} [v_y \quad \Omega^2 \quad v_y \Omega^2]^T \\ \tau_y &= C_{\tau_y} [v_x \quad \Omega^2 \quad v_x \Omega^2]^T \\ \tau_z &= C_{\tau_z} [v_x \quad v_y]^T \end{aligned} \quad (11)$$

where v_{xy} represents the horizontal velocity component. The selection of these terms is grounded in their relevance to the Blade Element Momentum theory (BEM) solution [34, 6], reflecting their significant impact on the system dynamics while maintaining computational efficiency. The ground truth force and torque values are derived from IMU measurements,

as well as from a VICON¹ system. The respective coefficients C_f , C_τ are then identified using ordinary linear least-squares regression, and remain constant at runtime. The data is gathered from the same race track to ensure that it captures the unique aerodynamic effects that arise from the racing maneuvers.

C. TuRBO tuning

We consider the MPCC++ controller as a black-box function in the context of BO. For a given set of controller parameters ϕ , we run an episode, collect a trajectory $\tau(\phi)$, and compute the reward $R(\tau(\phi))$. For the sake of clarity, we simply refer to the reward as $R(\phi)$. The controller tuning task can be framed as an optimization problem:

$$\phi_{n+1} = \arg \max_{\phi \in \Theta} R(\phi) \quad (12)$$

Problem 12 presents two caveats: i) no analytical form of $R(\phi)$ exists, allowing only for pointwise evaluation; and ii) each evaluation corresponds to completing an entire episode within the simulator, which means the number of evaluations is capped by our interaction budget with the simulated environment. BO works in an iterative manner, relying on two key ingredients: i) a probabilistic surrogate model approximating the objective function; and ii) an acquisition function $\alpha(\phi)$ that determines new evaluation points, balancing exploration and exploitation. A common choice for the surrogate model are Gaussian Processes (GPs), which allow for closed-form inference of the posterior mean $\mu(\phi)$ and variance $\sigma^2(\phi)$. The next evaluation point ϕ_{n+1} is then chosen by maximizing the acquisition function $\alpha(\phi)$. We utilize the Upper Confidence Bound (UCB) acquisition function, defined as $\alpha_{UCB}(\phi) = \mu(\phi) + \beta\sigma(\phi)$, where β is the exploration parameter. This process is iterated until the evaluation budget is exhausted. Among the various BO methodologies, we opt for TuRBO [60], a global BO strategy that runs multiple independent local BO instances concurrently. Each local surrogate model enjoys the benefits of local BO such as diverse modeling of the objective function across different regions. Each local run explores within a Trust Region (TR) - a polytope centered around the optimal solution of the local instance. The base side length of the TR, L , is adjusted based on the success rate of evaluations to guide exploration towards promising areas. In this work we use the TuRBO-1 implementation from the BOTorch library [61] and modify it for an eightfold setup, TuRBO-8.

In contrast to the baseline MPCC [14], which requires $N_\phi = 2n_{gates} + 4$ parameters to be tuned due to the complexity of the contour function, MPCC++ reduces the number of tunable parameters to $N_\phi = 8$. From Problem 6, the parameters in question within the MPCC++ formulation include Q_l , Q_c , Q_ω , R_{v_θ} , $R_{\Delta f}$ and μ , with Q_c , Q_ω further divided into horizontal and vertical components.

For our experiments, we define one episode of the tuning task as the completion of $M = 3$ consecutive laps around

the track. The reward attained at the end of the episode is calculated as the mean lap time, adjusted by a penalty factor for any solver failures:

$$R(\phi) = -\frac{1}{M} \sum_{i=0}^M t_i - \gamma r_{fail} \quad (13)$$

where t_i denotes the lap time of the i -th lap, while r_{fail} represents the solver failure rate, i.e. the proportion of steps that fail relative to the total steps. We apply a scaling factor of $\gamma = 100$ to the penalty term. Such failures arise mainly due to i) the complexity of the optimization problem at hand; and ii) the need to solve the optimization problem in real-time (one SQP iteration). Incorporating this penalty term, while not strictly necessary, has proven effective in practice, as it smoothens the objective function in favor of parameters that minimize solver failures. This reward structure is consistently applied across all experiments.

V. EXPERIMENTS

In this section, we compare our proposed method, MPCC++, against two baselines: MPCC and RL, and conduct a series of variations in the formulation. The methodology is consistent across all experiments, and utilizes the same quadrotor configuration. We choose the Split-S race track, featuring 7 gates, for all our experiments due to its prevalent use in previous works [7, 13, 14, 15]. We test our approach across three distinct simulation environments: i) a simple simulator which uses the nominal dynamics; ii) a high-fidelity simulator that calculates the propeller forces via Blade Element Momentum Theory (BEM) [34]; and iii) a data-driven simulator that predicts aerodynamic forces from real-world data. We then validate our method in the real world. All experiments were conducted on the same hardware under uniform conditions.

A. Simulation

We train the policies across the three simulators previously described using TuRBO as the default tuning method. Each policy is allocated a total budget of 600 episodes for training, which translates to a maximum of 1800 interactions with the environment, given that each episode involves completing 3 laps. For the environment setup, simulation and training, we use a combination of *Flightmare* [62] and *Agilicious* [63] software stacks.

At test time, we select the best policy from the training phase as the one with the highest reward, and run 10 episodes (equivalent to 30 laps). The results reported in Table I are based on the average lap time over the 10 episodes evaluated at test time. Additionally, we introduce the Training Success Rate (TSR) to quantify the percentage of training episodes completed successfully without gate collisions (out of 600). We also employ the Success Rate (SR) metric, as introduced in [7], to quantify the percentage of successful episodes during testing (out of 10).

¹<https://www.vicon.com>

Category	Methods	Tuning	Environments							
			Simple		BEM		Residual		Real World	
			Lap Time [s]	SR[%]	Lap Time [s]	SR[%]	Lap Time [s]	SR[%]	Lap Time [s]	SR[%]
MPCC [14]	Nominal	WML [16]	5.38 ± 0.1	100	5.51 ± 0.06	100	5.51 ± 0.13	83.3	-	-
		TuRBO	5.65 ± 1.07	89.7	5.37 ± 0.06	100	5.62 ± 0.23	96.7	5.67 ± 1.06	59.3
MPCC++ (ours)	Nominal w/ augment. w/ random.	TuRBO	5.16 ± 0.02	100	5.30 ± 0.02	100	5.37 ± 0.09	100	5.41 ± 0.14	100
		TuRBO	5.09 ± 0.10	100	5.15 ± 0.03	100	5.19 ± 0.03	100	5.38 ± 0.26	100
		TuRBO	5.20 ± 0.13	100	5.37 ± 0.08	100	5.26 ± 0.27	100	-	-
RL [7]	-	-	5.14 ± 0.09	100	-	-	5.26 ± 0.32	100	5.35 ± 0.15	85.0

TABLE I: Results for MPCC, MPCC++, and RL across three simulation environments - Simple, BEM, and Residual - as well as for real-world experiments. MPCC++ achieves a 100% success rate across all simulation and real-world environments.

1) *TuRBO vs WML*: We first investigate the difference between TuRBO [60] and WML [16], and discuss several advantages of TuRBO. We train the baseline MPCC [14] using both variants. Despite both methods achieving comparable maximum rewards, in this section we delve into understanding how each of these explore the parameter space. Figure 3 illustrates two of the tuned parameters - the minimum contour weight Q_c and the progress weight μ - over the number of episodes.

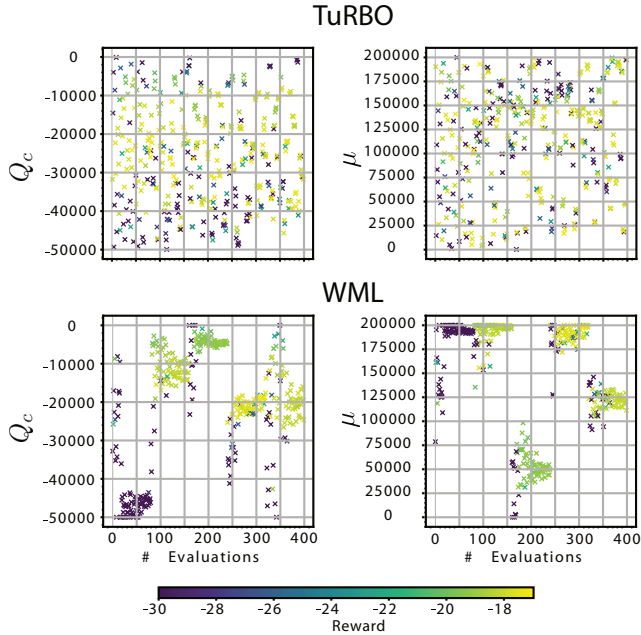


Fig. 3: Parameter exploration using TuRBO and WML. We plot two of the tuning parameters - Q_c and μ - over the number of episodes. TuRBO’s trust regions enhance exploration of the entire parameter space, while WML randomly restarts the exploration every 100 episodes.

To encourage exploration, WML initiates a new trial every 100 iterations with random initialization, aiming to discover new parameter regions. In contrast, TuRBO employs a more strategic approach by selecting trust regions in a similar manner to how it selects BO points, using informed reasoning. An additional advantage of TuRBO is that the trust regions are independent, allowing parallel execution without incurring extra computational cost. Consequently, we advocate for TuRBO as our preferred tuning algorithm for two reasons: i) comprehensive exploration of the entire parameter space; and

ii) computational efficiency and scalability, which are crucial for exploring large parameter spaces.

2) *MPCC++ vs MPCC*: We train both MPCC++ and the baseline MPCC using TuRBO. We make some practical adjustments to the MPCC++ formulation introduced in Section IV-A. Specifically, we relax the hard constraints of the tunnel and impose soft constraints instead. While our MPCC++ optimization problem provides numerical stability in settings that do not require real-time computations, employing real-time solvers does not inherently ensure numerically stable solutions. Even if a solution is found, real-world implementation presents challenges in ensuring the subsequent state respects the predefined constraints. This is particularly true when planning near boundaries, such as tunnel edges, where minor discrepancies due to model mismatches, disturbances, or state-estimation drift can lead to violations of these constraints in the subsequent states. Soft constraints are commonly employed to address such scenarios and can be performed in a systematic way [64]. This has several benefits: i) improves the numerical stability of the solver; ii) handles infeasibilities which arise from model mismatch by allowing minor violations; and iii) maintains the low computational complexity of MPCC. This adaptation does not have any practical implications in terms of performance or safety. The soft constraint is implemented using a barrier function $p(h(x))$ to embed the constraint $h(x) \geq 0$ into the cost function:

$$p(h(x)) = \log(1 + \exp(-\alpha h(x))) \quad (14)$$

where the penalty slope is set to $\alpha = 100$. Our results, as detailed in Table I, demonstrate MPCC++’s superiority over MPCC in terms of lap times and success rates. Due to the track constraint, MPCC++ prevents gate crashes consistently without compromising the lap time. Allowing the controller to independently navigate within the tunnel, instead of following a predefined path, gives it greater flexibility to identify the best trajectory based on its current state. The tunnel dimensions, W_n and W_{gate} , provide a mechanism to intuitively trade off between performance and safety. This balance was adjustable in test runs without necessitating retuning, indicating the policy’s robustness across varying tunnel widths.

In Figure 4, we show the trajectory from 10 episodes (30 laps) for both MPCC and MPCC++, noting MPCC++’s paths are notably more consistent. We attribute the inconsistency of MPCC to its representation of the contour weight as a

multivariate Gaussian, since the controller forcefully tries to follow the reference path, and acts poorly in the presence of disturbances.

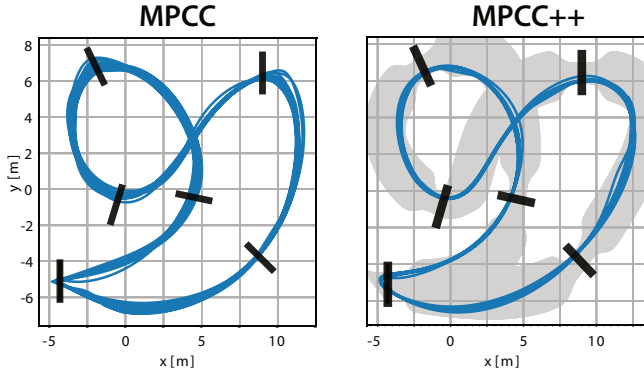


Fig. 4: Simulation experiments of MPCC with the proposed MPCC++, both tuned using TuRBO.

As shown in Table II, MPCC achieves a Training Success Rate (TSR) of roughly 70% across all simulations, indicating that 30% of the episode evaluations are not completed successfully due to gate collisions, while MPCC++ reaches a TSR of nearly 100%. Failed episodes are assigned with the lowest attainable reward and do not provide valuable insights for the BO’s surrogate model updates. As such, we consider these as fruitless interactions with the environment.

Category	Methods	Tuning	Environments		
			Simple	BEM	Residual
MPCC [14]	Nominal	WML [16]	62.9	75.7	64.4
		TuRBO	69.2	70.3	53.0
MPCC++ (ours)	w/ augment.	TuRBO	99.5	100	99.8
		TuRBO	100	100	100
		TuRBO	91.3	93.9	89.0

TABLE II: BO Training Success Rate (TSR): percentage of episodes during training completed successfully without gate collisions.

3) *MPCC++ with Model Augmentation*: Following the approach described in Section IV-B, we augment the nominal dynamics of the MPCC $f(x, u)$ with a residual component $g(x, u)$. Incorporating polynomial features to augment the dynamics requires knowledge of the motor speeds, which were not present in the original MPCC framework. To address this, we introduce several adaptations.

First, we include a first-order motor model into the dynamics to estimate the motor speeds in real-time:

$$\dot{\Omega} = \frac{1}{\tau_{mot}}(\Omega_{des} - \Omega) \quad (15)$$

where Ω_{des} and Ω denote the desired and the actual motor speeds, respectively. Second, we require an accurate estimate of these speeds to initialize the controller at each timestep. Additionally, we reformulate the optimal control problem to use motor speeds as inputs instead of single rotor thrusts. These adaptations are essential for leveraging data-driven

dynamic features within a real-time optimization framework, setting our approach apart from prior works like [6], which used augmented dynamics for environment simulation in RL.

As shown in Table I, we are able to further reduce the lap time on all of the environments by approximately 0.1s. We attribute this improvement to a combination of the above: i) respecting the motor dynamics; and ii) the actual augmentation.

4) *MPCC++ with Domain Randomization*: We address the robustness of the control policy against different noise realizations. The goal is to find policies which perform well amid model discrepancies. Such mismatches arise from using e.g. different hardware, leading to slight changes in mass, inertia, or thrust. We adapt the existing training pipeline to account for such noise realizations. For each BO iteration, we execute 10 different episodes each subjected to a distinct noise realization. We then collect the average reward over the 10 episodes and update the BO surrogate as usual. For fairness in comparison, we maintain the same budget of episode evaluations, which implies that the number of BO iterations is reduced by a factor 10 (from 600 to 60). In practice, reducing the number of BO iterations did not show any major limitations. We compare the obtained policies against the nominal MPCC++ and observe: i) slight increase in the lap time; ii) the consistency of the trajectory is preserved; and iii) 10 noise realizations are not sufficient for a truly robust policy. We conclude that the control formulation at hand is able to adapt to both changes in the platform, as well as sudden changes in the dynamics. We owe this property to the controller’s replanning capability within the constraint boundaries.

5) *MPCC++ vs RL*: In this section, we compare our MPCC++ method against the RL policy from [7]. We refer to the cited work for a detailed comparison between optimal control and RL. As shown in Table I, both MPCC++ and RL achieve similar lap times. However, MPCC++ incorporates explicit safety considerations into its design, enabling it to optimize trajectories by strategically taking shortcuts through gate corners. In practice, MPCC++ employs a low contour weight, Q_c , which, due to the explicit safety constraints in its design, allows the solver to focus on planning the optimal trajectory without excessively penalizing the contour error. In contrast, RL penalizes the gate passing distances in its reward function, which encourages the drone to navigate through the gate center. A detailed comparison of the trajectories is provided in Section V-B.

While MPCC++ incorporates specific safety measures into its design, it requires considerable engineering effort compared to an end-to-end learning framework like RL. This includes managing real-time computational constraints of the solver, adapting MPC outputs to low-level controller commands, addressing solver infeasibilities, and explicitly compensating for delays.

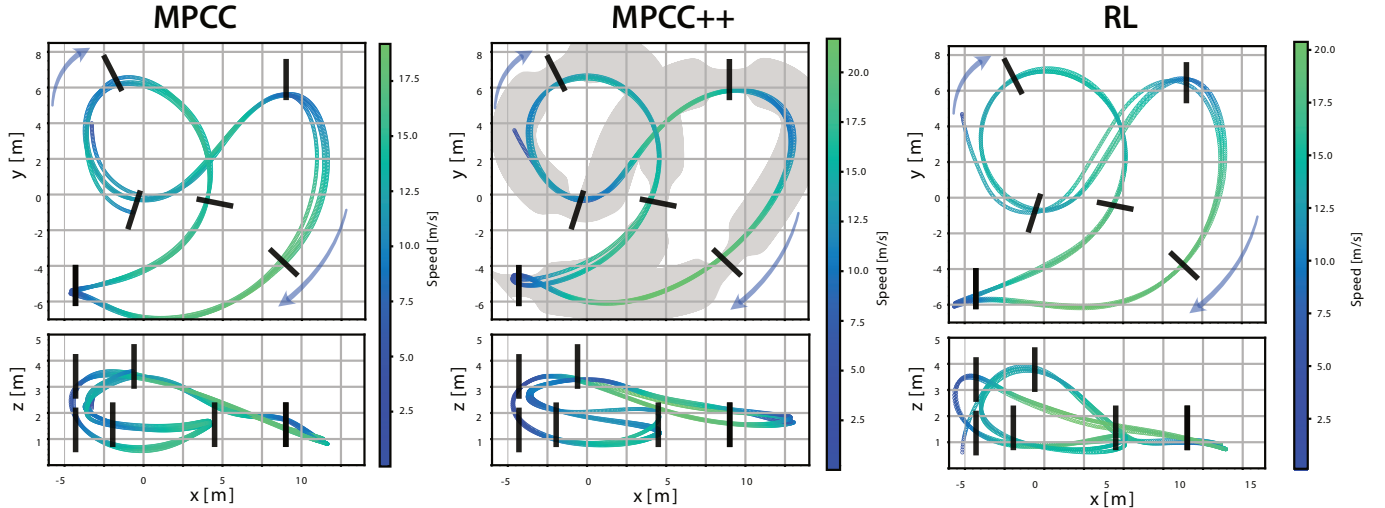


Fig. 5: Real world flight trajectories on the Split-S track for the baseline MPCC [14], MPCC++ and the best-performing RL policy [7]. Both MPCC and MPCC++ were tuned using TuRBO.

B. Real World

We test our approach in the real world using a high-performance racing drone with a high thrust-to-weight ratio (TWR). We use the Agilicious platform [63], as introduced in [7] under the designation *4s drone*. The control framework was deployed on ACADOS [65] using SQP_RTI for real-time computation, with the control loop running at 100Hz. We use a horizon length of $N = 20$ at a prediction rate of 25Hz, resulting in a prediction span of $T = 0.8s$.

Our method runs on an offboard desktop computer equipped with an Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz. A Radix FC board equipped with the Betaflight² firmware is used as the low-level controller, which takes as inputs desired body rates and collective thrusts. An RF bridge is employed to transmit commands to the drone. For state estimation, we use a VICON system with 36 cameras that provide the platform with millimeter accuracy measurements of position and orientation at a rate of 400 Hz.

We select the best policy obtained in the residual simulator and deploy it in the real world. We compute the lap times and success rates as done for the simulation experiments, i.e. averaging over 10 episodes. In Figure 5, we show the comparison between the baseline MPCC, MPCC++ and the best-performing RL policy. Both MPCC and MPCC++ are tuned with TuRBO. MPCC++ consistently satisfies the tunnel bounds, while achieving higher speeds than MPCC. Our policy did not crash a single time, being the first approach to achieve a 100% success rate in real-world experiments. The improvement in safety comes without a compromise in performance, as our approach achieves similar lap times to the best-performing RL policy. MPCC++ plans a trajectory that takes shortcuts through the gate corners, while in RL the deviation from the gate center is penalized in the reward, hence encouraging the drone to fly closer to the gate centers. The Split-S maneuver

at $x = -4.3m$ and $y = -5.6m$, stands out as a critical test of each approach’s characteristics, as it’s performed differently in all three cases. This complex maneuver requires the drone to fly through a higher gate and then immediately descend through a second gate located directly below the first, with both gates sharing the same x, y coordinates. This is the most challenging maneuver of the track, significantly influencing the overall lap time.

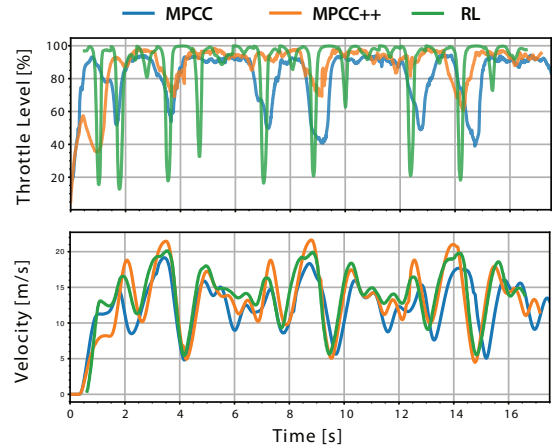


Fig. 6: Real world thrust and velocity profiles for MPCC, MPCC++ and RL.

Figure 6 illustrates the differences in throttle level, representing normalized commanded collective thrust, and velocity magnitude among the approaches. While velocity profiles were similar, the commanded throttle values differ significantly. This is because for MPCC and MPCC++, the solver output needs to be translated to equivalent desired body rates and collective thrust values to be commanded to the low-level controller. This mapping is non trivial, and needs to account for delay effects. We believe that it has a significant influence on the overall performance and is a major drawback of MPC-based approaches, while RL directly outputs a suitable

²<https://www.betaflight.com>

command.

VI. DISCUSSION

This paper introduces MPCC++, a safe MPCC controller for drone racing. We first introduce safety constraints, in the form of a track constraint and a terminal set, which systematically prevent gate collisions and ensure recursive feasibility. We then augment the nominal dynamics of the controller with a residual term that captures unmodeled behaviors such as aerodynamic effects. Finally, we tune the controller parameters using TuRBO, a state-of-the-art BO algorithm that efficiently compromises between local and global exploration. Our approach addresses several of the limitations encountered in the baseline MPCC: i) no systematic prevention against gate collisions; ii) sub-optimality from following a predefined optimal trajectory; and iii) tuning effort incurred from a high dimensional parameter space. We benchmark MPCC++ against the baseline MPCC and the best-performing RL policy in both simulation and real-world experiments and show that we are on par in terms of lap times while attaining safety against gate collisions. Our approach is the first to achieve 100% success rate in real-world experiments.

Nonetheless, MPCC++ also presents several limitations. First, it requires a centerline that passes through the gate centers and is used as the basis to construct the track constraint. While constructing the centerline incurs no computational effort, this approach does introduce certain limitations. Specifically, our approach requires mapping the drone’s current position to a point on the centerline, which is then linked to a progress value. Additionally, constructing the centerline as a spline requires defining the task as a sequence of waypoints in a known order. In contrast, RL architectures often use reward functions (sparse, binary, etc) to specify tasks, allowing for a simpler and more flexible task specification.

Second, our approach takes significantly longer to train than RL, as the environment cannot be parallelized on the GPU. This is because the solver calls cannot be batched into tensor operations, as done with learning-based policies.

Third, while our approach assumes that gate positions are known with high accuracy, future work should explore incorporating gate position uncertainty into our method.

Finally, our approach requires solving an optimization problem online at each step, which may limit its applicability on onboard computers with limited processing capabilities. In contrast, RL methods only require a simple forward pass of the neural network at runtime.

We conclude that the main advantage of MPCC++ is that it provides an intuitive lever to trade off between performance and safety. By shrinking the width of the tunnel, the quadrotor flies safely through the track. We can then gradually adjust the width to increase performance. Adjustments in the tunnel width can be made directly without the need of further re-tuning.

VII. ACKNOWLEDGMENTS

This work was supported by the European Union’s Horizon Europe Research and Innovation Programme under grant

agreement No. 101120732 (AUTOASSESS) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT). The authors especially thank Amon Lahr for his helpful acados tips, and Jiaxu Xing and Ismail Geles for their help with the experiments.

REFERENCES

- [1] G. Loianno and D. Scaramuzza, “Special issue on future challenges and opportunities in vision-based drone navigation,” 2020.
- [2] S. Rajendran and S. Srinivas, “Air taxi service for urban mobility: a critical review of recent developments, future challenges, and opportunities,” *Transportation Research Part E-logistics and Transportation Review*, Nov 2020.
- [3] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, “Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges,” *Ieee Access*, vol. 7, pp. 48572–48634, 2019.
- [4] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, “Autonomous vehicles on the edge: A survey on autonomous vehicle racing,” *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.
- [5] D. Hanover, A. Loquercio, L. Bauersfeld, A. Romero, R. Penicka, Y. Song, G. Cioffi, E. Kaufmann, and D. Scaramuzza, “Autonomous drone racing: A survey,” *arXiv e-prints*, pp. arXiv–2301, 2023.
- [6] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, “Champion-level drone racing using deep reinforcement learning,” *Nature*, vol. 620, pp. 982–987, Aug 2023.
- [7] Y. Song, A. Romero, M. Mueller, V. Koltun, and D. Scaramuzza, “Reaching the limit in autonomous racing: Optimal control versus reinforcement learning,” *Science Robotics*, p. adg1462, 2023.
- [8] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. Knoll, “A review of safe reinforcement learning: Methods, theory and applications,” *arXiv preprint arXiv:2205.10330*, 2022.
- [9] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, “Safe learning in robotics: From learning-based control to safe reinforcement learning,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 411–444, 2022.
- [10] K. P. Wabersich and M. N. Zeilinger, “A predictive safety filter for learning-based control of constrained nonlinear dynamical systems,” *Automatica*, vol. 129, p. 109597, 2021.
- [11] H. Dai, B. Landry, L. Yang, M. Pavone, and R. Tedrake, “Lyapunov-stable neural-network control,” in *Proceedings of Robotics: Science and Systems*, July 2021.
- [12] A. Romero, Y. Song, and D. Scaramuzza, “Actor-critic model predictive control,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*.

- [13] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, 2021.
- [14] A. Romero, S. Sun, P. Foehn, and D. Scaramuzza, "Model predictive contouring control for time-optimal quadrotor flight," *IEEE Transactions on Robotics*, pp. 1–17, 2022.
- [15] A. Romero, R. Penicka, and D. Scaramuzza, "Time-optimal online replanning for agile quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7730–7737, 2022.
- [16] A. Romero, S. Govil, G. Yilmaz, Y. Song, and D. Scaramuzza, "Weighted maximum likelihood for controller tuning," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1334–1341, IEEE, 2023.
- [17] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.
- [18] J. Ji, X. Zhou, C. Xu, and F. Gao, "Cmpcc: Corridor-based model predictive contouring control for aggressive drone flight," in *Experimental Robotics: The 17th International Symposium*, pp. 37–46, Springer, 2021.
- [19] J. Arrizabalaga and M. Ryll, "Towards time-optimal tunnel-following for quadrotors," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 4044–4050, IEEE, 2022.
- [20] M. Mueller, S. Lupashin, and R. D'Andrea, "Quadrotor ball juggling," pp. 4972–4978, 2011.
- [21] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," vol. 19, no. 3, pp. 20–32, 2012.
- [22] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," vol. 31, no. 5, pp. 664–674, 2012.
- [23] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient algorithm for state-to-state quadcopter trajectory generation and feasibility verification," 2013.
- [24] S. Spedicato, G. Notarstefano, J. Swevers, J. De Schutter, and M. Diehl, "Minimum-time trajectory generation for quadrotors in constrained environments," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, 2018.
- [25] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, 2009.
- [26] N. Hung, F. Rego, J. Quintas, J. Cruz, M. Jacinto, D. Souto, A. Potes, L. Sebastiao, and A. Pascoal, "A review of path following control strategies for autonomous robotic vehicles: Theory, simulations, and experiments," *Journal of Field Robotics*, vol. 40, no. 3, pp. 747–779, 2023.
- [27] C. Qin, M. S. Michet, J. Chen, and H. H.-T. Liu, "Time-optimal gate-traversing planner for autonomous drone racing," *arXiv preprint arXiv:2309.06837*, 2023.
- [28] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: From simulation to reality with domain randomization," *IEEE Trans. Robotics*, vol. 36, no. 1, pp. 1–14, 2019.
- [29] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A benchmark comparison of learned control policies for agile quadrotor flight," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.
- [30] L. O. Rojas-Perez and J. Martinez-Carranza, "DeepPilot: A CNN for autonomous drone racing," *Sensors*, vol. 20, no. 16, p. 4524, 2020.
- [31] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.
- [32] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement learning for uav attitude control," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 2, pp. 1–21, 2019.
- [33] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.
- [34] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," in *Proceedings of Robotics: Science and Systems*, 2021.
- [35] 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)*, pp. 1205–1212, IEEE, 2021.
- [36] N. A. Spielberg, M. Brown, and J. C. Gerdes, "Neural network model predictive motion control applied to automated driving with unknown friction," *IEEE Transactions on Control Systems Technology*, vol. 30, no. 5, pp. 1934–1945, 2021.
- [37] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh, "Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2819–2826, 2022.
- [38] A. Saviolo, G. Li, and G. Loianno, "Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10256–10263, 2022.
- [39] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information-theoretic model predictive control: Theory and applications to autonomous driving," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.
- [40] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Robust constrained learning-based nmPC enabling reliable mobile robot path tracking," *The International Journal*

- of *Robotics Research*, vol. 35, no. 13, pp. 1547–1563, 2016.
- [41] U. Rosolia and F. Borrelli, “Learning how to autonomously race a car: a predictive control approach,” *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2713–2719, 2019.
- [42] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, “Data-driven mpc for quadrotors,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.
- [43] M. Mehndiratta, E. Camci, and E. Kayacan, “Automated tuning of nonlinear model predictive controller by reinforcement learning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3016–3021, IEEE, 2018.
- [44] A. Carron, E. Arcari, M. Wermelinger, L. Hewing, M. Hutter, and M. N. Zeilinger, “Data-driven model predictive control for trajectory tracking with a robotic arm,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3758–3765, 2019.
- [45] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, “Information theoretic mpc for model-based reinforcement learning,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1714–1721, IEEE, 2017.
- [46] A. Lahr, A. Zanelli, A. Carron, and M. N. Zeilinger, “Zero-order optimization for gaussian process-based model predictive control,” *European Journal of Control*, vol. 74, p. 100862, 2023.
- [47] K. J. Åström, “Theory and applications of adaptive control—a survey,” *automatica*, vol. 19, no. 5, pp. 471–486, 1983.
- [48] M. J. Grimble, “Implicit and explicit lqg self-tuning controllers,” *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 941–947, 1984.
- [49] K. J. Åström, T. Hägglund, C. C. Hang, and W. K. Ho, “Automatic tuning and adaptation for pid controllers—a survey,” *Control Engineering Practice*, vol. 1, no. 4, pp. 699–714, 1993.
- [50] M. A. Mohd Basri, A. R. Husain, and K. A. Danapalasingam, “Intelligent adaptive backstepping control for mimo uncertain non-linear quadrotor helicopter systems,” *Transactions of the Institute of Measurement and Control*, vol. 37, no. 3, pp. 345–361, 2015.
- [51] A. Schperberg, S. Di Cairano, and M. Menner, “Auto-tuning of controller and online trajectory planner for legged robots,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 7802–7809, 2022.
- [52] M. Zanon and S. Gros, “Safe reinforcement learning using robust mpc,” *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3638–3652, 2020.
- [53] S. Cheng, M. Kim, L. Song, C. Yang, Y. Jin, S. Wang, and N. Hovakimyan, “DiffTune: Auto-tuning through auto-differentiation,” *arXiv preprint arXiv:2209.10021*, 2022.
- [54] J. De Schutter, M. Zanon, and M. Diehl, “Tunempc—a tool for economic tuning of tracking (n)mpc problems,” *IEEE Control Systems Letters*, vol. 4, no. 4, pp. 910–915, 2020.
- [55] A. Loquercio, A. Saviolo, and D. Scaramuzza, “Autotune: Controller tuning for high-speed flight,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4432–4439, 2022.
- [56] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, “Automatic tuning for data-driven model predictive control,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7379–7385, IEEE, 2021.
- [57] L. P. Fröhlich, C. Küttel, E. Arcari, L. Hewing, M. N. Zeilinger, and A. Carron, “Model learning and contextual controller tuning for autonomous racing,” *arXiv preprint arXiv:2110.02710*, 2021.
- [58] L. Numerow, A. Zanelli, A. Carron, and M. N. Zeilinger, “Inherently robust suboptimal mpc for autonomous racing with anytime feasible sqp,” 2024.
- [59] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing.
- [60] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, “Scalable global optimization via local Bayesian optimization,” in *Advances in Neural Information Processing Systems*, pp. 5496–5507, 2019.
- [61] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy, “BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization,” in *Advances in Neural Information Processing Systems 33*, 2020.
- [62] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, “Flightmare: A flexible quadrotor simulator,” in *Conference on Robot Learning*, 2020.
- [63] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, and D. Scaramuzza, “Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight,” *Science Robotics*, vol. 7, no. 67, 2022.
- [64] K. P. Wabersich and M. N. Zeilinger, “Predictive control barrier functions: Enhanced safety mechanisms for learning-based control,” *IEEE Transactions on Automatic Control*, vol. 68, no. 5, pp. 2638–2651, 2023.
- [65] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, “acados – a modular open-source framework for fast embedded optimal control,” *Mathematical Programming Computation*, Oct 2021.