

# Model Predictive Contouring Control for Near-Time-Optimal Quadrotor Flight

Angel Romero, Sihao Sun, Philipp Foehn, Davide Scaramuzza

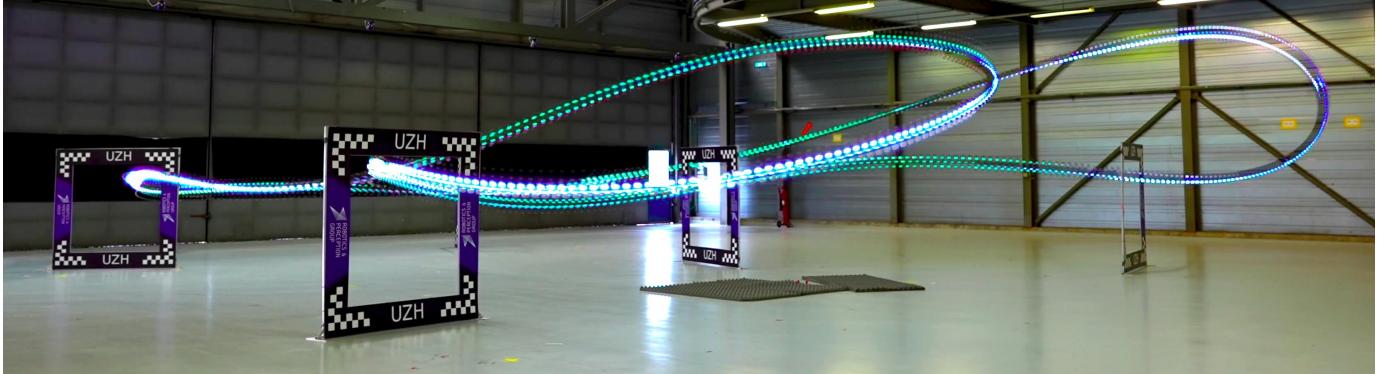


Fig. 1: Execution of a trajectory generated using a point-mass model and tracked by the proposed Model Predictive Contouring Controller. This control approach results in near-time-optimal quadrotor flight, achieving velocities of up to 60 km/h, accelerations over 5 g, and lap times that outperform those of world-class professional drone pilots.

**Abstract**—We tackle the problem of flying time-optimal trajectories through multiple waypoints with quadrotors. State-of-the-art solutions split the problem into a planning task—where a global, time-optimal trajectory is generated—and a control task—where this trajectory is accurately tracked. However, at the current state, generating a time-optimal trajectory that takes the full quadrotor model into account is computationally demanding (in the order of minutes or even hours). This is detrimental for replanning in presence of disturbances. We overcome this issue by solving the time-optimal planning and control problems concurrently via Model Predictive Contouring Control (MPCC). Our MPCC optimally selects the future states of the platform at runtime, while maximizing the progress along the reference path and minimizing the distance to it. We show that, even when tracking simplified trajectories, the proposed MPCC results in a path that approaches the true time-optimal one, and which can be generated in real-time. We validate our approach in the real-world, where we show that our method outperforms both the current state-of-the-art and a world-class human pilot in terms of lap time achieving speeds of up to 60 km/h.

Video of the experiments: <https://youtu.be/mHDQcckqdg4>

## I. INTRODUCTION

Multirotor drones are extremely agile, so agile, in fact, that they have become an essential tool for time-critical missions, such as search and rescue, aerial delivery, and even flying cars [1, 2]. For this reason, over the past decade, research on autonomous multirotor drones has continually pushed platforms to higher speeds and agility [3–12]. Several competitions have been organized, such as the Autonomous

The authors are with the Robotics and Perception Group, University of Zurich, Switzerland (<http://rpg.ifi.uzh.ch>). This work was supported by the National Centre of Competence in Research (NCCR) Robotics through the Swiss National Science Foundation (SNSF) and the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement No. 871479 (AERIAL-CORE) and the European Research Council (ERC) under grant agreement No. 864042 (AGILEFLIGHT).

Drone Racing series at the recent IROS and NeurIPS conferences [13–15] and the AlphaPilot challenge [8, 16]. Their goal: to develop autonomous systems that will eventually outperform expert human pilots. Million-dollar projects, such as AgileFlight [17] and Fast Light Autonomy (FLA) [6], have also been funded by the European Research Council and the United States government, respectively, to further advance the field.

In this paper, we address the problem of flying a quadrotor through multiple waypoints in minimum time. This is often desired for delivery and inspection tasks, and, in the context of search and rescue and drone racing, it is even the ultimate goal. Professional drone racing pilots achieve this with astonishing performance, flying their quadrotors through race tracks at speeds difficult to achieve for autonomous systems.

Time-optimal multi-waypoint flight raises fundamental challenges for robotics research in terms of planning, control, aerodynamics, and modeling. These become increasingly complex in the presence of high accelerations and aggressive attitude changes. Exploring new ways of generating and tracking minimum-time trajectories not only addresses these challenges, but also has direct implications in what is physically possible for a quadrotor given its limited actuation, ultimately pushing the frontiers of current and emerging real-world applications.

State-of-the-art approaches for time-optimal multi-waypoint flight [12, 18] split this problem into a planning task—where a global, time-optimal trajectory is generated—and a control task—where this trajectory is accurately tracked.

While the control task can be solved in real-time, at the current state of the art, generating a feasible, global, time-optimal trajectory that considers the full quadrotor dynamics, including single-rotor thrust constraints, is computationally demanding (in the order of minutes or even hours) [12]. This is detrimental for replanning in presence of disturbances.

When the platform is at the limit of actuation, the slightest deviation from the pre-planned trajectory may result in a sub-optimal flight path or even in a crash. To circumvent this, the trajectory needs to be either planned with conservative actuation limits—which sacrifices speed ([12], Section IV.E)—or be replanned online—which is not possible with the current solver times. Thereby, all computationally efficient near-time-optimal waypoint planners resort to either modeling the platform as a point-mass or approximating trajectories with polynomials. For point-mass model approaches, the problem of finding time-optimal, point-to-point trajectories has a closed form solution [19] and is, therefore, very fast to solve. This low computational burden has enabled a number of sampling-based methods for quadrotors, including time-optimal, multi-waypoint flight [8]. However, these simplified trajectories lack the notion of 3D rotation and are dynamically infeasible (since quadrotors are underactuated systems, they need to rotate to align their thrust with the desired acceleration direction). On the other hand, polynomial trajectories offer a fast way of generating feasible paths. However, polynomial control inputs are smooth and cannot fully exploit the actuator potential, rendering control policies sub-optimal.

We show that these limitations can be overcome by changing the paradigm and using a Contouring Control approach [20, 21] instead of standard trajectory tracking methods.

#### *Contribution*

In this paper, we propose an MPCC method that considers the full quadrotor dynamics and the real single-rotor thrust constraints to achieve near-time-optimal quadrotor flight. In contrast to standard trajectory tracking control methods, our MPCC approach balances the maximization of the progress along a given 3D path and the minimization of the distance to it. Instead of relying on a pre-computed, computationally expensive path, the proposed MPCC has the freedom to optimally select at runtime the states at which the reference trajectory is sampled such that the progress is maximized and the platform stays within actuator bounds.

Furthermore, we demonstrate that this control approach is particularly well suited for the problem of autonomous drone racing. To accurately pass through the waypoints, we encode their positions within our cost function by dynamically allocating the contour weight. Since the reference 3D path does not need to be feasible, we efficiently generate it by using a simpler point-mass model. The proposed controller renders speeds, accelerations, and lap times very similar to the theoretical time-optimal trajectories [12] while keeping the computational burden low enough to be solved in real-time.

We show that the proposed MPCC can make better use of the available actuator potential and results in faster real-flight lap times than both a standard MPC controller tracking a time-optimal trajectory and a world-class professional human pilot.

## II. RELATED WORK

### A. Quadrotor Trajectory Tracking Control

Nonlinear controllers have been proposed to address quadrotors' complex attitude dynamics, such as the geometric con-

troller [22], quaternion-based controller [23], tilt-prioritized controller [24]. Model Predictive Control (MPC) is another trajectory tracking solution that simultaneously handles nonlinearities and input constraints [25, 26]. Aerodynamic effects are detrimental to tracking accuracy, especially in high-speed flights. Various methods are proposed to tackle aerodynamic effects, such as using first-principle models [27], Gaussian-process models [28], or leveraging accelerometer measurements [29]. All these control strategies rely on a previously computed dynamically feasible sequence of states and inputs to track. The problem of generating this sequence is called trajectory planning.

### B. Quadrotor Time-Optimal Trajectory Planning

Quadrotor trajectory planning has been extensively studied in the literature. Polynomial trajectories are the first category of planning algorithms, such as the now widely used minimum-snap trajectories [30, 3]. Given a 4<sup>th</sup> order smooth trajectory, the full state at each point on the trajectory can be derived using the differential flatness property of the quadrotor.

Autonomous drone racing considers the problem of traversing a path in minimum time while passing through the designated waypoints (gates). To this end, polynomial-based multi-waypoint trajectory planning algorithms have been widely used in the literature due to their simplicity. However, in the context of completing a race track in minimum time, sampling states and inputs that are inherently polynomials result in sub-optimal policies. Due to their continuity and smoothness, polynomial trajectories are not able to render control inputs that constantly maximize accelerations during a certain time segment (an issue already observed in [12]). By contrast, optimization-based trajectory planning allows to independently select the optimal sequence of states and inputs at every time step, which inherently considers time minimization while complying with quadrotor dynamics and input constraints. Optimization-based approaches have been extensively considered in the literature, ranging from exploiting point-mass models [31], simplified quadrotor models [32, 33], and full-state quadrotor models [12, 34].

Regarding sampling-based methods, several time-minimizing approaches have been suggested. In [8, 35], a point-mass model is used for the high-level time-optimal trajectory planning. In [35], an additional trajectory smoothing step is done where they connect the generated trajectory with high order polynomials by leveraging the differential flatness property of the quadrotor. Similarly, in [36, 37] the authors take advantage of the differential flatness property in order to search in the space of polynomial, smooth minimum-jerk trajectories while also including the minimization of the time in the cost function. However, these approaches need to relax the single actuator constraints and instead limit the per-axis acceleration, which results in control policies that are conservative and sub-optimal given a minimum time objective. Furthermore, since the authors use polynomials, these sampling approaches can only generate smooth control inputs, meaning that they cannot, for instance, continuously apply full thrust if required. The MPCC approach proposed

in our work circumvents these problems by separating the planning, which is done using a point-mass model, from the time-optimal objective, which is ultimately handled by the controller. This allows for non-smooth control inputs that respect the true actuator constraints to take full advantage of the input authority of the platform.

Apart from time-optimality, complying with intermediate waypoint constraints is another requirement for path planning in autonomous drone racing. A common practice of solving a trajectory optimization problem with waypoint constraints is allocating waypoints to specific time steps and minimizing the spatial distance between these waypoints and the position at the corresponding allocated time steps on the reference trajectory (e.g., [38, 39]). The time allocation of the waypoints is, however, non-trivial and difficult to determine. This is tackled in [34] which, however, uses body rates and collective thrust as control inputs and does not represent realistic actuator saturation. Recent work [12] introduces a complementary progress constraints (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 the above methods guarantee global optimality of the quadrotor trajectory passing through gates, they are computationally costly and hence intractable in real-time.

### C. Model Predictive Contouring Control

In contrast to solving trajectory planning and control separately, Model Predictive Contouring Control (MPCC) combines trajectory generation and tracking into a single optimization problem [40, 20]. This method minimizes the distance to the reference while maximizing the progress along it in a receding horizon fashion, thereby giving more freedom to the controller to determine its state trajectory. The MPCC method only needs a continuously differentiable 3D path as reference, hence dropping the feasibility requirement that standard MPC methods have. Furthermore, it allows adding additional constraints and cost functions for obstacle avoidance, such as in [41–43]. Most pieces of literature apply MPCC on biaxial systems, such as CNC machines [44], ground vehicles [21, 41], and autonomous ground robots [43]. In [21], the authors apply MPCC in the context of car racing and discuss that the latter offers control policies that are closer to human-like driving than the ones from standard MPC methods. In [42] the corridor-based MPCC is proposed for drone flights. A simplified 3<sup>rd</sup> order linear integral model is adopted where jerk is regarded as the control input. However, the nonlinear dynamics and the input constraints are neglected, limiting the method to relatively low-speed flights.

In this paper, we propose an MPCC architecture that considers the more challenging three-dimensional space for autonomous drone racing. To fully exploit the capability and realize near-optimal flight performance, we formulate the optimization problem using the full nonlinear quadrotor model. Additionally, a novel cost function is designed to guarantee the traversal of gates.

## III. METHODOLOGY

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. Let us denote the time discretized evolution of the system  $f : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$  such that

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

where the sub-index  $k$  is used to denote states and inputs at time  $t_k$ .

The general Optimal Control Problem 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:

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

### A. Model Predictive Control

For standard Model Predictive Control approaches [10, 25, 26], the objective is to minimize a quadratic penalty on the error between the predicted states and inputs, and a given dynamically feasible reference  $\mathbf{x}_{k,\text{ref}}$  and  $\mathbf{u}_{k,\text{ref}}$ . Consequently, the cost function  $J(\mathbf{x})$  in problem (1) is substituted by:

$$J_{\text{MPC}}(\mathbf{x}) = \sum_{k=0}^{N-1} \|\Delta \mathbf{x}_k\|_Q^2 + \|\Delta \mathbf{u}_k\|_R^2 + \|\Delta \mathbf{x}_N\|_P^2 \quad (2)$$

where  $\Delta \mathbf{x}_k = \mathbf{x}_k - \mathbf{x}_{k,\text{ref}}$ ,  $\Delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k,\text{ref}}$ , and where  $\mathbf{Q} \succeq 0$ ,  $\mathbf{R} \succ 0$  and  $\mathbf{P} \succeq 0$  are the state, input and final state weighting matrices. The norms of the form  $\|\cdot\|_A^2$  represent the weighted Euclidean inner product  $\|\mathbf{v}\|_A^2 = \mathbf{v}^T \mathbf{A} \mathbf{v}$ .

This formulation relies on the fact that a feasible reference  $\mathbf{x}_{k,\text{ref}}$ ,  $\mathbf{u}_{k,\text{ref}}$  is accessible for the future time horizon  $N$ . Searching for these is often referred to as *planning*, and for several applications, it becomes rather arduous and computationally expensive [12, 45].

### B. Model Predictive Contouring Control

In contrast to standard MPC methods, where a time-sampled state and input reference is tracked, MPCC methods consider the higher-level task of minimizing the Euclidean distance to a three-dimensional path while maximizing the speed at which the path is traversed.

Let us denote the arc length from the beginning of the path to step  $k$  as  $\theta_k$ . Let us also denote the desired arc length parameterized 3D path as  $\mathbf{p}^d(\theta_k) = [x^d(\theta_k) \quad y^d(\theta_k) \quad z^d(\theta_k)]^T$ , and the system's position as  $\mathbf{p}_k = [x_k \quad y_k \quad z_k]^T$ . The Model Predictive Contouring Control formulation (MPCC [20]) attempts to minimize the projected distance from the current position  $\mathbf{p}_k$  to the desired position  $\mathbf{p}^d(\theta_k)$  while maximizing the progress  $\theta$  along it. This is referred to in the literature as the *Contouring Control Problem* [20], and it can be written in

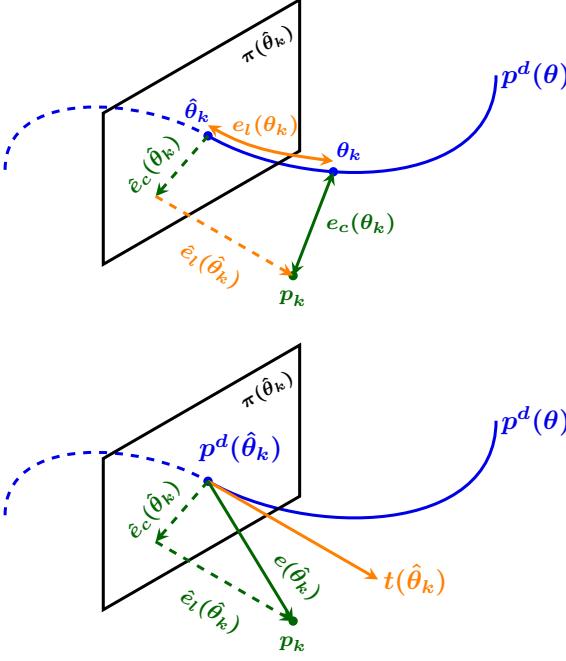


Fig. 2: Top: definition of contour error and its approximation, shown in green, and lag error and its approximation, shown in orange.  $\mathbf{p}_k$  is the current position of the platform at time  $k$  and  $\mathbf{p}^d(\theta_k)$  is the reference position at time  $k$ . Bottom: illustration of position error  $e(\theta_k)$  and tangent vector  $t(\theta_k)$ .

the same shape as problem (1) by substituting the cost function by:

$$J_{MPCC}(\mathbf{x}) = \sum_{k=0}^N q_c e_c^2(\theta_k) - \rho \theta_N \quad (3)$$

where  $e_c(\theta_k) \in \mathbb{R}^+$  is the *contour error* at step  $k$  and is defined as the projection of  $\mathbf{p}_k$  over  $\mathbf{p}^d(\theta_k)$ ,

$$e_c(\theta_k) = \min_{\theta_k} \|\mathbf{p}_k - \mathbf{p}^d(\theta_k)\| \quad (4)$$

In cost function (3), the negative sign on  $\rho \in \mathbb{R}^+$  indicates that the progress  $\theta$  at step  $N$  is maximized. Finally,  $q_c \in \mathbb{R}^+$  is the *contour weight*.

However, this formulation of the contour error is not well suited to be used inside an online optimization scheme because it is an optimization problem by itself (4). To overcome this limitation, the contour error  $e_c(\theta_k)$  is approximated by the norm of its linear projection  $\hat{e}_c(\hat{\theta}_k)$  onto the tangent plane  $\pi(\hat{\theta}_k)$ . The arc length of the true projection  $\theta_k$  and the arc length of the approximated projection  $\hat{\theta}_k$  are linked by the *lag error*  $e_l(\theta_k) = \|\theta_k - \hat{\theta}_k\|$ . However, since the true projection  $\theta_k$  is not known, the true lag error  $e_l(\theta_k)$  is also approximated by the norm of  $\hat{e}_l(\hat{\theta}_k)$ . Fig. 2 (top) depicts a graphical interpretation of  $e_c(\theta_k)$ ,  $e_l(\theta_k)$  and their approximations.

**Proposition 1.** If  $\hat{e}_l(\hat{\theta}_k) = \mathbf{0}$ , the approximations  $\hat{\theta}_k$ ,  $\hat{e}_l(\hat{\theta}_k)$ , and  $\hat{e}_c(\hat{\theta}_k)$  are equal to their respective true values.

*Proof.* If  $\hat{e}_l(\hat{\theta}_k) = \mathbf{0}$ , then  $\mathbf{p}_k \subset \pi(\hat{\theta}_k)$ , where  $\pi(\hat{\theta}_k)$  is the normal plane to the curve at the point  $\mathbf{p}^d(\hat{\theta}_k)$ . By (4),  $e_c(\theta_k)$  is the shortest distance from  $\mathbf{p}_k$  to  $\mathbf{p}^d(\theta_k)$  and

hence  $e_c(\theta_k) \subset \pi(\theta_k)$  and  $p_k \subset \pi(\theta_k)$ . It follows that  $\pi(\hat{\theta}_k) \equiv \pi(\theta_k)$ . Therefore,  $\hat{\theta}_k = \theta_k$ ,  $e_c(\theta_k) = \|\hat{e}_c(\hat{\theta}_k)\|$  and  $e_l(\theta_k) = \|\hat{e}_l(\hat{\theta}_k)\| = 0$ .  $\square$

Consequently, to ensure that these approximations are accurate it is necessary to include the minimization of  $\|\hat{e}_l(\hat{\theta}_k)\|$  in the cost function of problem (6).

Now, consider  $v_{\hat{\theta}} = \frac{d\hat{\theta}}{dt}$ , the speed at which  $\hat{\theta}_k$  evolves, discretized, as  $\hat{\theta}_{k+1} = \hat{\theta}_k + v_{\hat{\theta}} \Delta t$ . Hence, we can write

$$\hat{\theta}_N = \hat{\theta}_0 + \sum_{k=0}^N v_{\hat{\theta},k} \Delta t \quad (5)$$

Adding  $\|\hat{e}_l(\hat{\theta}_k)\|$  and (5) to (3), the optimization in problem (1) becomes:

$$\begin{aligned} \pi(\mathbf{x}) = \operatorname{argmin}_u & \sum_{k=0}^N \|\hat{e}_c(\hat{\theta}_k)\|_{q_c}^2 + \|\hat{e}_l(\hat{\theta}_k)\|_{q_l}^2 - \mu v_{\hat{\theta},k} \\ \text{subject to } & \mathbf{x}_0 = \mathbf{x} \\ & \mathbf{x}_{k+1} = f(\mathbf{x}_k, u_k) \\ & \mathbf{x} \in \mathcal{X} \quad u_k \in \mathcal{U} \end{aligned} \quad (6)$$

where  $q_l \in \mathbb{R}^+$  is the *lag weight*, the maximization of  $\theta_N$  has been rewritten in terms of  $v_{\hat{\theta},k}$  and  $\mu \in \mathbb{R}^+$  is the new progress weight variable. From this point on, we drop the hat nomenclature for the approximations of the errors for readability. Thus,  $\theta_k$ ,  $e_c(\theta_k)$ ,  $e_l(\theta_k)$  and  $v_{\hat{\theta},k}$  directly refer to their corresponding approximations.

This control strategy offers multiple advantages that are exploited in this paper. Particularly, since the maximization of the progress is directly encoded in the optimization problem being solved online, it benefits from the freedom to select the best times at which the states along the reference 3D path are sampled. This way, the MPCC formulation eliminates the need for a set of pre-sampled states and inputs. Thus the only requirement is an arc length parameterized, continuously differentiable 3D path to track,  $\mathbf{p}^d(\theta)$ .

### C. Arc Length Parameterization of the Paths

Arc length parameterization of general curves is a non-trivial problem, and finding a closed-form solution for an arc-length parameterized curve given another arbitrary parameterization is, in general, not possible. Instead, we find approximations to arc length parameterized paths. Let  $\mathbf{p}^d(t)$  be the 3D path to track, and it can be given in two forms: as a continuous time-dependent curve or as a sequence of sampled points in time. If it is given as a continuous curve, we use the *bisection method* from [46] where the curve is sampled such that the arc length between samples is constant by numerically computing the integral and doing binary search until convergence. If it's given as a sequence of sampled points, we search for these equidistant segments by assuming linearity between samples. In both cases, for each point, we store the corresponding arc length, the position, and the normalized velocity. This process results in a sequence of  $P$  of these points to which we fit 3<sup>rd</sup> order splines  $\rho_i(\theta_k)$   $i \in [0, \dots, P-1]$  as shown in (7). For further details, we refer the reader to [46].

$$\mathbf{p}^d(\theta_k) = \begin{cases} \boldsymbol{\rho}_0(\theta_k) & 0 \leq \theta_k \leq \theta_0 \\ \boldsymbol{\rho}_1(\theta_k) & \theta_1 \leq \theta_k \leq \theta_2 \\ \vdots \\ \boldsymbol{\rho}_{P-1}(\theta_k) & \theta_{P-1} \leq \theta_k \leq \theta_P \end{cases} \quad (7)$$

#### D. Derivation of Contour and Lag Errors in 3D

Let the position error at time  $k$  be  $\mathbf{e}(\theta_k) = \mathbf{p}_k - \mathbf{p}^d(\theta_k)$ . Consider the tangent line of the desired path,  $\mathbf{t}(\theta_k) \in \mathbb{R}^3$ , as shown in Fig. 2 (bottom),

$$\mathbf{t}(\theta_k) = \frac{\partial \mathbf{p}^d(\theta_k)}{\partial \theta_k} = \left[ \frac{\partial x^d(\theta_k)}{\partial \theta_k} \quad \frac{\partial y^d(\theta_k)}{\partial \theta_k} \quad \frac{\partial z^d(\theta_k)}{\partial \theta_k} \right]^T \quad (8)$$

$\mathbf{e}(\theta_k)$  can be decomposed in a component that is projected onto  $\mathbf{t}(\theta_k)$ ,  $\mathbf{e}_l(\theta_k)$ , and a component contained in  $\pi(\theta_k)$ , which is  $\mathbf{e}_c(\theta_k)$ . Note that because  $\mathbf{p}^d(\theta_k)$  is arc length parameterized,  $\|\mathbf{t}(\theta_k)\|$  is always 1.

Thus, the expression for the projection of  $\mathbf{e}(\theta_k)$  onto  $\mathbf{t}(\theta_k)$  is:

$$\mathbf{e}_l(\theta_k) = (\mathbf{t}(\theta_k)^T \cdot \mathbf{e}(\theta_k)) \mathbf{t}(\theta_k) = e_l(\theta_k) \mathbf{t}(\theta_k)$$

We are interested in minimizing the  $q_l$ -weighted Euclidean norm of this error,

$$\begin{aligned} \|\mathbf{e}_l(\theta_k)\|_{q_l}^2 &= \|e_l(\theta_k)\|_{q_l}^2 \cdot \|\mathbf{t}(\theta_k)\|_{q_l}^2 \\ &= q_l e_l^2(\theta_k) \end{aligned} \quad (9)$$

Similarly, we can write the contour error as:

$$\begin{aligned} \mathbf{e}_c(\theta_k) &= \mathbf{e}(\theta_k) - \mathbf{e}_l(\theta_k) = \\ &\begin{bmatrix} 1 - t_x^2(\theta_k) & -t_x(\theta_k)t_y(\theta_k) & -t_x(\theta_k)t_z(\theta_k) \\ -t_x(\theta_k)t_y(\theta_k) & 1 - t_y^2(\theta_k) & -t_y(\theta_k)t_z(\theta_k) \\ -t_x(\theta_k)t_z(\theta_k) & -t_y(\theta_k)t_z(\theta_k) & 1 - t_z^2(\theta_k) \end{bmatrix} \mathbf{e}(\theta_k) \end{aligned}$$

And its  $q_c$ -weighted Euclidean norm,

$$\|\mathbf{e}_c(\theta_k)\|_{q_c}^2 = \mathbf{e}_c(\theta_k)^T \mathbf{Q}_c \mathbf{e}_c(\theta_k) \quad (10)$$

where  $\mathbf{Q}_c = q_c \cdot \mathbf{I}^{3 \times 3}$ .

## IV. APPLICATION TO QUADROTOR

To apply our MPCC method to a quadrotor, in this section we define its state and input space, and its dynamics.

#### A. Quadrotor Dynamics

The quadrotor's state space is described from the inertial frame  $I$  to the body frame  $B$ , as  $\mathbf{x} = [\mathbf{p}_{IB}, \mathbf{q}_{IB}, \mathbf{v}_{IB}, \mathbf{w}_B]^T$  where  $\mathbf{p}_{IB} \in \mathbb{R}^3$  is the position,  $\mathbf{q}_{IB} \in \mathbb{SO}(3)$  is the unit quaternion that describes the rotation of the platform,  $\mathbf{v}_{IB} \in \mathbb{R}^3$  is the linear velocity vector, and  $\mathbf{w}_B \in \mathbb{R}^3$  are the bodyrates in the body frame. The input of the system is given as the collective thrust  $\mathbf{f}_B = [0 \ 0 \ f_{Bz}]^T$  and body torques  $\tau_B$ . For readability, we drop the frame indices as they are

consistent throughout the description. The dynamic equations are

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} & \dot{\mathbf{q}} &= \frac{1}{2} \mathbf{q} \odot \begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix} \\ \dot{\mathbf{v}} &= \mathbf{g} + \frac{1}{m} \mathbf{R}(\mathbf{q}) \mathbf{f}_T & \dot{\boldsymbol{\omega}} &= \mathbf{J}^{-1} (\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{J} \boldsymbol{\omega}) \end{aligned} \quad (11)$$

where  $\odot$  represents the Hamilton quaternion multiplication,  $\mathbf{R}(\mathbf{q})$  the quaternion rotation,  $m$  the quadrotor's mass, and  $\mathbf{J}$  the quadrotor's inertia.

Additionally, the input space given by  $\mathbf{f}$  and  $\boldsymbol{\tau}$  is decomposed into single rotor thrusts  $\mathbf{f} = [f_1, f_2, f_3, f_4]$  where  $f_i$  is the thrust at rotor  $i \in \{1, 2, 3, 4\}$ .

$$\mathbf{f}_T = \begin{bmatrix} 0 \\ 0 \\ \sum f_i \end{bmatrix} \quad \text{and} \quad \boldsymbol{\tau} = \begin{bmatrix} l/\sqrt{2}(f_1 + f_2 - f_3 - f_4) \\ l/\sqrt{2}(-f_1 + f_2 + f_3 - f_4) \\ c_\tau(f_1 - f_2 + f_3 - f_4) \end{bmatrix} \quad (12)$$

with the quadrotor's arm length  $l$  and the rotor's torque constant  $c_\tau$ .

Furthermore, in order to approximate the most prominent aerodynamic effects, we extend the quadrotor's dynamics to include a linear drag model [47]. Let  $\mathbf{D}$  be the diagonal matrix with drag coefficients  $d$  such that  $\mathbf{D} = \text{diag}(d_x, d_y, d_z)$ , we expand the dynamic model from (11) by:

$$\dot{\mathbf{v}} = \mathbf{g} + \frac{1}{m} \mathbf{R}(\mathbf{q}) \mathbf{f}_T - \mathbf{R}(\mathbf{q}) \mathbf{D} \mathbf{R}^\top(\mathbf{q}) \cdot \mathbf{v} \quad (13)$$

#### B. Optimal Control Problem Formulation

In this section, we propose to include the dynamics introduced in section IV-A within the MPCC formulation outlined in problem (6). To do this, the state space described in (11) needs to be augmented to include progress dynamics. A direct way to achieve this is to add  $\theta$  as a state and  $v_\theta$  as a virtual input. However, this way there would be no limit in how fast  $v_\theta$  changes and infinite changes would be allowed leading to very noisy control inputs. To alleviate this, and following typical implementations of contouring controllers [21, 41], we use the *progress acceleration*,  $\Delta v_\theta = \frac{dv_\theta}{dt}$  as a virtual input instead. The resulting augmented state and input spaces are shown in (14).

$$\mathbf{x} = [\mathbf{p} \ \mathbf{q} \ \mathbf{v} \ \mathbf{w} \ \mathbf{f} \ \theta \ v_\theta]^T, \quad \mathbf{u} = [\Delta v_\theta \ \Delta \mathbf{f}]^T \quad (14)$$

where the dynamics of the augmented states are:

$$\begin{aligned} \mathbf{f}_{k+1} &= \mathbf{f}_k + \Delta \mathbf{f}_k \Delta t \\ \theta_{k+1} &= \theta_k + v_{\theta,k} \Delta t \\ v_{\theta,k+1} &= v_{\theta,k} + \Delta v_{\theta,k} \Delta t \end{aligned} \quad (15)$$

Thus, the full optimal control problem is as follows:

$$\begin{aligned}
\pi(\mathbf{x}) = \operatorname{argmin}_u & \sum_{k=0}^N \|\mathbf{e}_l(\theta_k)\|_{q_l}^2 + \|\mathbf{e}_c(\theta_k)\|_{q_c}^2 + \|\boldsymbol{\omega}_k\|_{Q_\omega}^2 \\
& + \|\Delta v_{\theta_k}\|_{r_{\Delta v}}^2 + \|\Delta \mathbf{f}_k\|_{R_{\Delta f}}^2 - \mu v_{\theta,k} \\
\text{subject to } & \mathbf{x}_0 = \mathbf{x} \\
& \mathbf{x}_{k+1} = f(\mathbf{x}_k, u_k) \\
& \underline{\boldsymbol{\omega}} \leq \boldsymbol{\omega} \leq \bar{\boldsymbol{\omega}} \\
& \underline{\mathbf{f}} \leq \mathbf{f} \leq \bar{\mathbf{f}} \\
& 0 \leq v_\theta \leq \bar{v}_\theta \\
& \underline{\Delta v_\theta} \leq \Delta v_\theta \leq \bar{\Delta v_\theta} \\
& \underline{\Delta \mathbf{f}} \leq \Delta \mathbf{f} \leq \bar{\Delta \mathbf{f}}
\end{aligned} \tag{16}$$

where the lag and countour norms were defined in section III-D, in (9) and (10), respectively, and the dynamics  $f(\mathbf{x}_k, u_k)$  come from the discretization of (11) and (12), and from (15).

In problem (16), constraints and costs on  $\Delta v_\theta$  and  $\Delta \mathbf{f}$  have been added such that the solver does not have complete freedom of choosing them arbitrarily. These extra constraints and costs are needed in contouring control formulations to provide stability to the optimization problem [41, 21]. Without them, the outputs of the solver are very noisy, rendering the inputs unusable for real-world applications. The cost on  $\boldsymbol{\omega}$  has also proven to be necessary for the stability of the solution since it forces the solver to keep low body rates whenever possible. Furthermore, this cost also allows tracking a reference in  $\omega_z$  to control the platform's orientation externally. Lastly, the limits on  $v_\theta$  ensure non-reversal of the reference path and set a maximum speed at which the path is traversed. The complete MPCC pipeline is described in Algorithm 1.

### C. Dynamic Allocation of Contouring Weight

Given a set of  $M$  waypoints located at  $\{\mathbf{p}_{g,j}(\theta_k)\}_{j=0}^M$ , we consider the problem of passing through them in the shortest time possible. When relating this to the optimization problem described in (16), one can notice two competing objectives in the controller cost function: the contouring weight  $q_c$ , which pushes the platform towards the center of the 3D path and eventually through the waypoints, and the progress weight  $\mu$ , which makes it traverse the path as fast as possible.

Because of these competing terms in the cost function, there is a need to find a balance between going fast and reducing the contour error. For waypoint tracking, however, we are only concerned about the tracking performance to the 3D path in the points where the waypoints are. For this, the position of the waypoints  $\{\mathbf{p}_g\}_{j=0}^M$  need to be included in our optimal control formulation (16). This can be done in different ways.

On the one hand, the position constraints could be directly included in the optimization problem, i.e., adding

$$\|\mathbf{p}_k - \mathbf{p}_g(\theta_j)\|_2^2 \leq d_{tol,j} \quad \forall j \in [0, M]$$

to problem (16). However, this approach has been found to be unsuccessful. If the constraints are added as hard constraints, the problem quickly becomes unfeasible. If they are soft constraints, the controller behavior becomes extremely conservative. The optimization problem becomes very complex

---

**Algorithm 1:** MPCC

---

```

Input:  $p^d(t)$  the 3D path to track,  $x_0$  initial state
 $\mathcal{P} \leftarrow []$  vector of corresponding positions, velocities
and arc lengths

if  $p^d(t)$  is a sequence of sampled points then
|  $\mathcal{P}(i) \leftarrow$  assume linearity to compute arc length
else if  $p^d(t)$  is continuous then
|  $\mathcal{P}(i) \leftarrow$  use bisection method from [46]

for  $i \in [1, P]$  do
|  $\rho_i(\theta_k) \leftarrow \mathcal{P}(i), \mathcal{P}(i-1)$  compute splines from
| points
end
 $p^d(\theta_k) \leftarrow \{\rho_i(\theta_k)\}_{i=0}^P$  as in (7)

 $\{\mathbf{x}_k\}_{k=0}^N \leftarrow x_0, \{\theta_k\}_{k=0}^N \leftarrow 0$ 
loop
|  $\mathcal{C}(\theta_k) \leftarrow []$  vector of spline coefficients
| for  $k \in [0, N]$  do
| |  $\theta_{pred,k} \leftarrow \theta_k$  from previous  $X_{pred,k}^*$ 
| |  $\mathcal{C}(\theta_k) \leftarrow p^d(\theta_{pred,k})$ 
| end
| compute (9) and (10) given  $\mathcal{C}(\theta_k)$ 
|  $U_{pred}^*, X_{pred}^* \leftarrow$  solve problem (16)
| apply  $U_{pred,0}^*$  to the platform
end

```

---

because we need to add a slack variable, a constraint for the slack variable, a cost for the slack variable, and the soft constraint itself.

On the other hand, the contour weight  $q_c$  could be dynamically allocated online. Instead of keeping it constant, we make it depend on the desired position of the platform  $q_c(\theta_k) = q_c(\mathbf{p}^d(\theta_k))$  such that when we are close to a waypoint, the contour error is the largest one in the cost function, and everywhere else we give more importance to the progress. The mapping that has been chosen is a collection of 3D Gaussians placed at the waypoint positions  $\mathbf{p}_{g,j}(\theta_k)$ , yielding

$$q_c(\mathbf{p}^d(\theta_k)) = \sum_{j=0}^M \frac{e^{-\frac{1}{2}(\mathbf{p}^d(\theta_k) - \mathbf{p}_{g,j}(\theta_k))^T \Sigma^{-1} (\mathbf{p}^d(\theta_k) - \mathbf{p}_{g,j}(\theta_k))}}{\sqrt{(2\pi)^3 |\Sigma|}}$$

where  $\Sigma$  is the diagonal covariance matrix that indicates how wide the Gaussians are in  $x$ ,  $y$ , and  $z$  axes. The elements of  $\Sigma$  are chosen to be equal in the three axes and relatively small such that (i) they only contain information around the waypoint position and (ii) the Gaussians do not overlap with each other.

Using Gaussians as the waypoint weighting function has been traditionally used in the literature for similar purposes, such as [48]. For this work, the main reasons behind the decision of using Gaussians are:

- As they are smooth curves, arbitrarily small changes in  $\theta_k$  lead to arbitrarily small changes in  $q_c(\mathbf{p}^d(\theta_k))$ , preventing discontinuities in the cost function and therefore helping with controller stability.

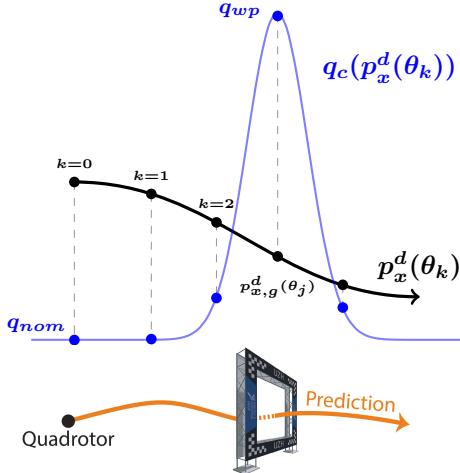


Fig. 3: Top: dynamic allocation of the weight in the  $x$  axis where the reference path  $p_x^d(\theta_k)$  (shown in black) is the  $x$  component of  $\mathbf{p}^d(\theta_k)$ . When closer to the waypoint, the contouring weight  $q_c(p_x^d(\theta_k))$  (shown in blue) grows to  $q_{wp}$ , decreasing the tracking error around the waypoints. Everywhere else  $q_c$  is kept constant and equal to  $q_{nom}$  such that the MPCC has more freedom to optimize for progress. Bottom: Illustration of how the contouring weight allocation would translate to for the drone racing task, where the gate position corresponds to the position of the waypoint.

- As explained in section III, there is always a lag error. Therefore the projection from our current position  $p_k$  to the desired path  $\mathbf{p}^d(\theta_k)$  might not be completely accurate at all times, and this approximate projection might both be lagging or leading. The symmetry properties of Gaussians ensure that the costs are attained equally no matter in which direction these projection inaccuracies lie.

A 1D illustrative example in Fig. 3 shows how this looks like for the  $x$  axis. Note that the contouring cost is a function of the desired path  $\mathbf{p}^d(\theta_k)$  and not of the current position  $p_k$ . This way, we ensure that these costs will always be attained since the reference path will always pass through the waypoints.

Summarizing, the proposed controller takes as inputs a 3D path  $\mathbf{p}^d(\theta)$  and a set of waypoints  $\{\mathbf{p}_g\}_{j=0}^M$  and outputs a policy that optimally balances going as fast as possible and being close to the tracked path where the waypoints are. It is therefore interesting to see how its performance compares with a true time-optimal trajectory, even if we expect to encounter differences due to (i) the horizon of the MPCC is limited, (ii) several approximations have been made in the proposed formulation, and (iii) there are additional terms in the cost function of problem (16) apart from purely maximizing the progress.

## V. SIMULATION EXPERIMENTS

In this section, we design a series of simulation experiments aimed to demonstrate the capabilities of our method. First, in section V-A we show a comparison between our method and a true time-optimal trajectory for the simple task of hover to hover flight. Then, in section V-B we evaluate the performance of our method in the more challenging task of multi-waypoint flight applied to drone racing. In this context, we design

TABLE I: Quadrotor Parameters

Property	RPG Quad
$m$ [kg]	0.85
$l$ [m]	0.15
$\text{diag}(J)$ [gm <sup>2</sup> ]	[2.5, 2.1, 4.3]
$[T_{min}, T_{max}]$ [N]	[0.0, 7.0]
$c_T$ [1]	0.022
$\omega_{max}$ [rad s <sup>-1</sup> ]	10

different ways of generating the 3D path  $\mathbf{p}^d(\theta)$  that passes through all the waypoints and compare them in terms of speed and lap times with respect to the main baseline, [12]. Finally, in section V-C we evaluate the robustness of the proposed MPCC controller against the presence of time delay in the loop and compare its performance to that of a standard MPC controller. Note that, while in this section we focus only on simulation experiments, in Section VI we present the real-world results.

The quadrotor configuration used throughout this paper, denoted *RPG Quad* in Table I, represents the model parameters of our in-house designed drone.

### A. Time-Optimal Hover to Hover Comparison

To compare up to which extent the maximization of the progress in problem (16) is equivalent to the minimization of the trajectory time, we have set up a simple simulation experiment: to execute a 15 m hover to hover trajectory in the  $x$  axis. This task has been used as a benchmark in previous work [12, 32, 33]. First, we compute the time-optimal trajectory using the same quadrotor model described in section IV-A. We compute it using the CPC approach [12], which takes between 20 and 30 minutes to converge on a desktop computer. This trajectory is the theoretical lower bound on how fast this task can be performed. On the other hand, we simulate the platform controlled by the proposed MPCC to follow a 3D straight line reference that can be run in real-time. It is important to clarify that no information is shared between the full model time-optimal trajectory and the MPCC controller: the 3D reference path  $\mathbf{p}^d(\theta)$  is plainly an arc-length parametrized straight line. In both cases, the maximum acceleration of the platform has been limited to 20 m/s<sup>2</sup>, as in [12, 32, 33].

In Fig. 4 we show a comparison of position, velocity, and acceleration between the CPC time-optimal reference and the simulated MPCC sequence of states. The results of this simulation show that the evolution of the MPCC controlled system is very close to the time-optimal CPC reference. The main differences between both approaches can be attributed to various reasons:

- Due to its limited time horizon, the proposed MPCC strategy finds locally optimal solutions. In contrast, the CPC algorithm has access to information about the evolution of all the past and future states of the platform and therefore finds the time-optimal trajectory.
- While the MPCC already shows the simulated results of an executed trajectory, the CPC is a planned reference

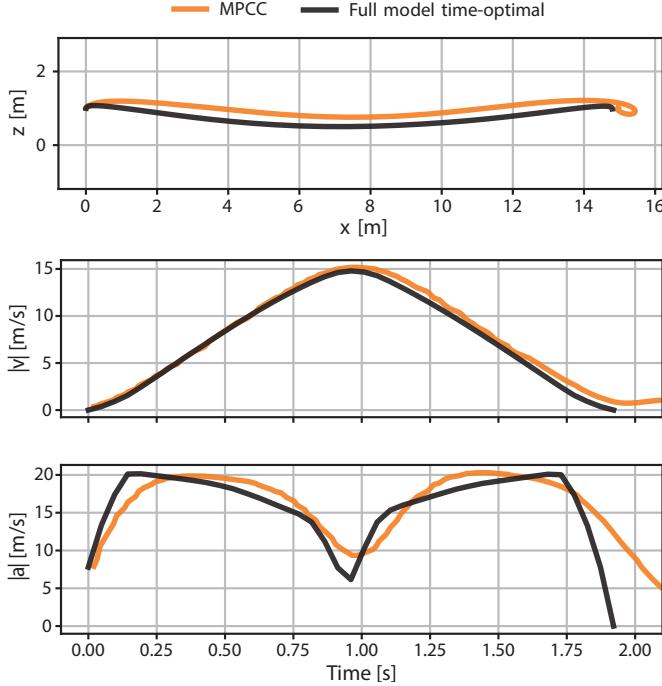


Fig. 4: Simulated position, velocity, and acceleration profiles of the MPCC compared to the time-optimal trajectory when executing a 15 meters hover to hover task. Note that the MPCC is solely tracking a straight line between the two hover states. The similarities in the speed and acceleration profiles show how the proposed MPCC controller attempts to find an approximation to the pre-computed time-optimal trajectory.

trajectory. We have chosen to compare directly with the reference because it does not depend on the tuning parameters of an external controller.

#### B. Ablation Study on the Choice of Reference Trajectory

In this section, we perform an ablation study that aims to show and evaluate up to which extent a better choice of the reference path helps the MPCC finding a good approximation of the time-optimal policy. First, we introduce different approaches to generate a 3D path that passes through a pre-defined sequence of waypoints, and then we benchmark them in a simulation experiment. In the end, we compare in terms of achieved lap times the presented approaches.

The locations of these waypoints are the same as in [12] and their spatial distribution is shown in Fig. 8. This track has been designed by professional drone pilots and will be used as the main benchmark throughout this paper, under the name *CPC-track*. It is important to note that the same controller tuning and contouring weights have been used for all our simulation experiments.

1) *Multi-Waypoint Polynomial Trajectories*: Polynomial, and in particular minimum snap trajectories, have been used for full state trajectory planning for quadrotors in different applications [30], [49]. One drawback of polynomial trajectories is that states and inputs are sampled from their derivatives and, therefore, can only offer polynomial control inputs. Since polynomials are smooth and can only attain their maximum value at single points, it is not possible to get arbitrarily

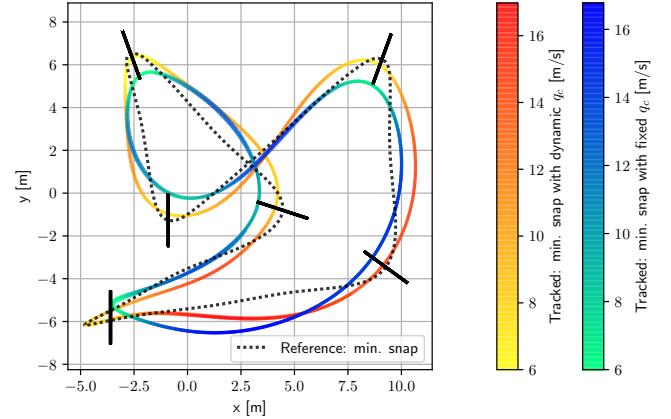


Fig. 5: *CPC-track* consisting of 7 gates (black segments). A 3D view of this track can be found in Fig. 8. The polynomial reference is shown in dotted black and is tracked using the proposed MPCC, with dynamic and fixed contouring weights  $q_c$  respectively. Both tracked trajectories are colored by speed profile (dynamic contouring weight: orange-yellow color, fixed contouring weight: blue-green color). Notice that the tracked trajectory when  $q_c$  is fixed does not successfully pass through all the gates. Instead, when  $q_c$  changes dynamically, the platform successfully goes through all the gates without any notable change in speed.

fast-changing control inputs. This undermines the agility and aggressiveness that can be achieved by the platform, rendering them suboptimal for minimum-time applications. In this section, we use polynomial trajectory planning to obtain a continuously differentiable 3D path  $p^d(t)$  that passes through the gates. The control inputs and the states of the drone (except position) will be generated by the MPCC controller at runtime.

These polynomial trajectories have been generated in a receding horizon fashion. Starting from the current position of the platform, we generate a polynomial that (i) passes through the  $N$  next waypoints and (ii) minimizes the second derivative of the acceleration (or snap, as in [30]). From this result, we select only the part from the current waypoint to the next waypoint and re-run the planning for the next waypoint with the same horizon length. This approach does not require an initial selection of speeds at each waypoint, as it is required for polynomial trajectories. Additionally, it keeps the order of the polynomial segments lower as compared to generating one single polynomial trajectory that passes through all the gates and hence avoids numerical issues. From this point on, this polynomial trajectory will be referred to as *minimum snap trajectory*.

In Fig. 5, we show an XY view of the MPCC tracking when following a minimum snap trajectory. To show the benefit of using the dynamic contouring weight method introduced in Section IV-C, we compare the trajectory execution of the proposed MPCC when  $q_c$  is dynamically allocated and when  $q_c$  is fixed to a constant value. If  $q_c$  remains fixed, one can notice how the platform does not pass through most of the gates or passes too close to the edge (which would result in a collision with the gate frame in a real flight). Since we use the same value for  $q_c$  in the entire track, the approach lacks information about the location of the waypoints and

treats the entire path equally. The balance between maximizing progress and minimizing contour error is constant. However, when using the dynamic contouring weight formulation, the platform successfully passes through all the gates. Furthermore, even if with the dynamic formulation we deprioritize the maximization of the progress at the location of the waypoints, it does not have any notable impact in the speed profile, as seen in the colored bars of Fig. 5. Therefore, in the remainder of the paper, a dynamic contouring weight formulation is used.

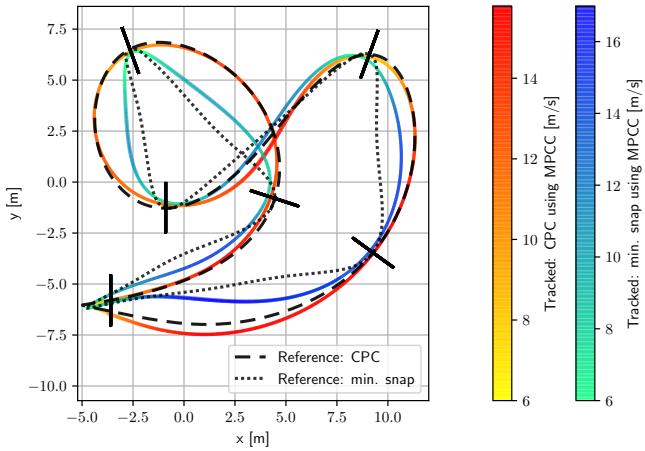


Fig. 6: Simulated tracking performance comparison on the CPC track when using minimum snap and CPC as a reference. The minimum snap reference is shown in dotted black, and the CPC reference is shown in dashed black, while the corresponding MPCC tracked trajectories are colored by their speed profile (low contouring weight: blue-green color, high contouring weight: orange-yellow color). The MPCC controller is using the dynamic contour weights shown in Fig. 3.

2) *Time-Optimal Full Model (CPC)*: In Fig. 6 we show the executed 3D path and the velocity profile for both the minimum snap reference and the CPC reference. Because the minimum snap trajectory not minimizing the total trajectory time, it has no notion of the long-term global behavior of the platform, leading to sharper turns and long straight lines. Because of its finite time horizon, when tracking this minimum snap reference, the MPCC is not able to find a path that goes through the gates without having to include these sharp turns and long straights. Even if this allows the MPCC approach to achieve higher instantaneous peak speeds (as shown in the blue color bar in Fig. 6), the overall velocity is lower, thereby leading to suboptimal, non-competitive lap times as illustrated in Fig. 10 (labeled as *Min. snap-MPCC*). In contrast, when the CPC path is provided as a reference, as shown in Fig. 10, lap times (labeled as *CPC-MPCC*) are very similar to the theoretical lower bound (labeled as *Reference CPC*). This shows that the MPCC is able to benefit from the long-term optimality information encoded in the CPC 3D path. Therefore the choice of reference path is indeed highly relevant to achieve competitive lap times. The lap times in Fig. 10 were computed using the same method as in [12]. For the rest of the paper, lap times are computed in this fashion unless stated otherwise.

One big caveat of using a time-optimal CPC trajectory

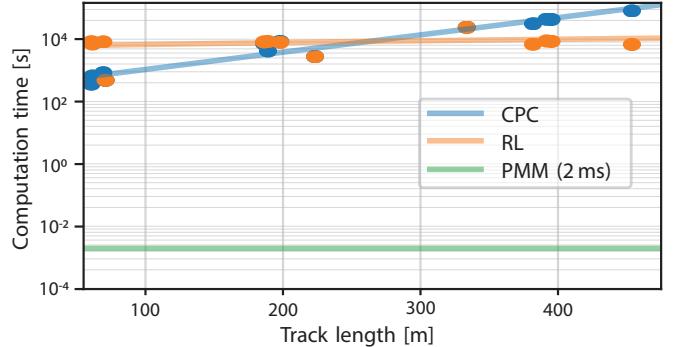


Fig. 7: Evolution of computation times with respect to track length for optimal time trajectory generation. Both available approaches so far, model-based (CPC) and learning-based (RL), are not usable in applications that require online re-planning, whereas the proposed contouring controller allows us to generate optimal trajectories using a point-mass model (PMM), rendering computation times suitable for real-time re-planning (2 ms).

as the reference is that they take a significant amount of time to compute. The generation time of these full state time-optimal trajectories ranges from 30 minutes to several hours, making them intractable for real-time applications. One attempt to reduce this computation time has been made [45] using reinforcement learning (RL), but although this approach scales better with track length, training times are still of the order of hours. Instead, the proposed MPCC approach does not need most of the information generated by these methods, and it suffices to provide the 3D path  $p^d(t)$ . In section V-B3 we leverage this property and generate time-optimal trajectories using a point-mass model (PMM) that are then tracked with MPCC. Fig. 7 depicts how the computation times to generate these trajectories using different approaches evolve with the track length.

3) *Time-Optimal Point-Mass Model*: As mentioned, one of the benefits of MPCC is that its only requirement is a continuously differentiable 3D path  $p^d(t)$ . In this section, we exploit this property and focus on generating minimum time trajectories with a point-mass model (PMM), and then track the generated 3D path with MPCC. This significantly reduces the computational burden since the generation of point-to-point trajectories using a point mass model has a closed-form solution.

To generate these PMM time-optimal 3D paths that can be tracked using our MPCC approach, the same sampling approach that was used in [8] is employed here. Given an initial state consisting of position and velocity, and given the reduced dynamics of a point-mass model,  $\ddot{p} = u$ , with the input acceleration being constrained  $\underline{u} \leq u \leq \bar{u}$ , it can be shown by using Pontryagin's maximum principle [50] that the time-optimal control input results in a bang-bang policy. Furthermore, if we also add constraints on the velocity,  $\underline{v} \leq v \leq \bar{v}$ , the time optimal control policy has a bang-

singular-bang solution [19] of the form:

$$u_x^*(t) = \begin{cases} \underline{u}_x, & 0 \leq t \leq t_1^* \\ 0, & t_1^* \leq t \leq t_2^* \\ \bar{u}_x, & t_2^* \leq t \leq T_x^* \end{cases} \quad (17)$$

or vice versa, starting with  $\bar{u}_x$ . It is straightforward to verify that there exists a closed-form solution for the switching times. In (17) we depict the solution for the  $x$  axis, but this can be extended for  $y$  and  $z$  axes without loss of generality.

Once we have the per-axis minimum time policy, we choose the maximum of these minimum times ( $T^* = \max(T_x^*, T_y^*, T_z^*)$ ) and slow down the other two axes' policies to make all three axes' durations equal. To do this, a new parameter  $\alpha \in [0, 1]$  is introduced that scales the acceleration bounds. For example, if we need to increase the duration of the  $x$  axis, the applied control inputs are scaled to  $\alpha \underline{u}_x$  and  $\alpha \bar{u}_x$ , respectively. For more details on how these trajectories are sampled, the reader is referred to [8].

By using the described method, it is possible to generate a time-optimal point-to-point motion primitive given start and end positions and velocities. In order to find the time-optimal, multiwaypoint trajectory efficiently, the path planning problem is interpreted as a shortest path problem. At each gate,  $M$  different velocities are sampled at random such that they lie in a cone pointing towards the exit direction of the gate. The cost from each sampled state at the previous gate is set to be equal to the duration,  $T^*$ , of the time-optimal motion primitive that guides the drone from one state to the other. Due to the existence of a closed-form expression for the minimum time,  $T^*$ , setting up and solving the shortest path problem can be done very efficiently using, e.g., Dijkstra's algorithm [50]. In order to further reduce the computational cost, the path is planned in a receding horizon fashion, i.e., the path is only planned through the next  $H_g$  gates.

The resulting trajectory for our race track is shown in Fig. 8 and from now on it will be referred to as point-mass model (PMM) reference trajectory.

TABLE II: Total times (averaged over 10 runs) for different gate horizons ( $H_g$ ) for the task of completing 3 laps in the CPC-track using the described Time-Optimal Point-Mass model (PMM) algorithm. Notice that from  $H_g = 3$ , the improvement in total time is negligible.

$H_g$	Avg. Time [s]	Min. Time [s]	Max. Time [s]
1	16.0	15.75	16.19
2	15.8779	15.62	16.27
3	14.8244	14.80	14.85
4	14.8147	14.80	14.83
5	14.818	14.80	14.83

In our particular application, the gate horizon  $H_g = 3$  gates. To justify this choice, we have generated PMM trajectories for the task of completing 3 laps on the CPC-track, for different values of  $H_g$ . We then compare the resulting total lap times (averaged over 10 runs) and show them in Table II. One can notice how for  $H_g$  larger than 3, the improvement in total time is negligible. The number of samples has been chosen as in

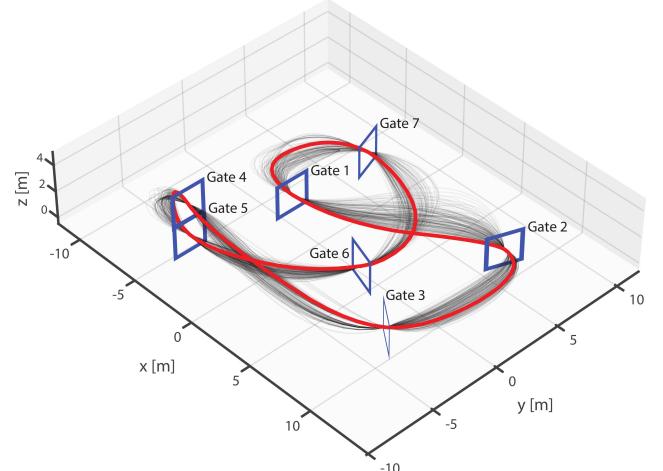


Fig. 8: Time-optimal trajectory generated using a point-mass model (PMM) by sampling the velocities at the gates. The time-optimal trajectory is colored in red, while the different samples are in gray. This PMM trajectory is unfeasible for our platform, so traditional MPC methods fail to track it. However, the proposed MPCC solely needs a continuously differentiable 3D path. The computed PMM path has a shape sufficiently close to the time-optimal one that it renders close to time-optimal results when executed by our MPCC, while it can be computed in real-time.

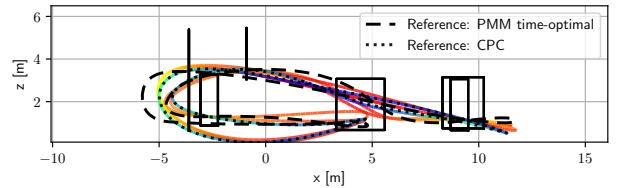
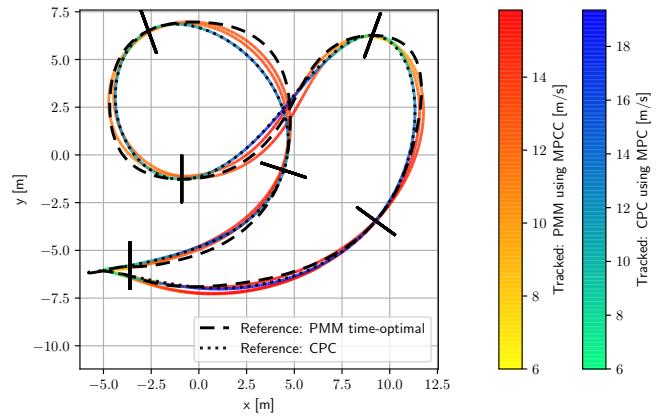


Fig. 9: Simulated tracking performance of our MPCC approach when tracking a PMM reference (black dashed line) shown from the top view (top panel) and the side (bottom panel). The CPC time-optimal reference has been added for comparison. Notice how the position of the tracked MPCC trajectory approaches the time-optimal reference trajectory.

[8],  $M = 150$  for each gate in the prediction horizon, resulting in computation times of around 2 ms.

Fig. 9 shows the results of tracking the time-optimal point mass model trajectory with the proposed MPCC, and the time-optimal CPC reference tracked with standard MPC for comparison. It can be noticed that the behavior of the platform

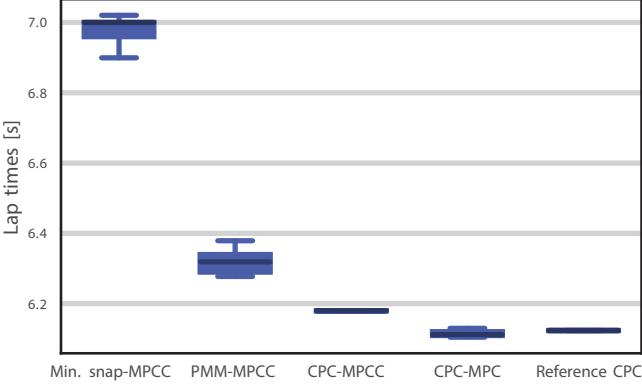


Fig. 10: Sampled lap times of the different trajectories tracked using our MPCC approach in simulation and of the time-optimal reference (*Reference CPC*). Notice how the point mass model tracked trajectory (*PMM-MPCC*) is very close to the tracked time-optimal trajectory (*CPC-MPCC*), yet does not suffer from the large computational effort required to compute a CPC trajectory. The *CPC-MPC* trajectory is the CPC tracked by a standard MPC approach, for comparison.

approaches the time-optimal trajectory, as also suggested by Fig. 4, i.e., maximizing the progress along the MPCC horizon provides the short term time-optimality, whereas a PMM time-optimal reference path to track provides the controller with the long term information that results in a behavior similar to a true time-optimal policy. This can also be seen in Fig. 10, where the lap times of the MPCC tracking a PMM reference (labeled as *PMM-MPCC*) approaches the theoretical lower bound by only a small difference.

In Fig. 10 one can also notice how the tracked CPC trajectory using a standard MPC approach (labeled as *CPC-MPC*) has almost identical lap times as the *Reference CPC*. This is expected because standard MPC approaches use a reference that is sampled by time. The planned trajectory dictates at which times every reference state will be visited, meaning that the lap time is fully defined by the reference trajectory.

### C. Time Delay Study

Time delays are commonplace in any real system and can be originated by different sources, such as communication, sensors, or filtering algorithms, among others. In this section, we evaluate how our controller reacts to the presence of a time delay in the control loop. To do so, we manually introduce a delay in the measurement of position, velocity, and attitude in our simulation. In Fig. 11 we show how both the proposed MPCC and a standard MPC react to different values of this delay, from 0 ms to 60 ms.

One can notice how the proposed MPCC can successfully complete the track in the presence of up to 50 ms delay. The explanation lies in the fact that MPCC does not use a time-sampled reference, whereas MPC does. Standard MPC methods need a reference that consists not only of a sequence of states but also of a timestamp for each state. This reference is fixed at the planning stage and cannot be changed online, limiting the capabilities of the approach when a significant

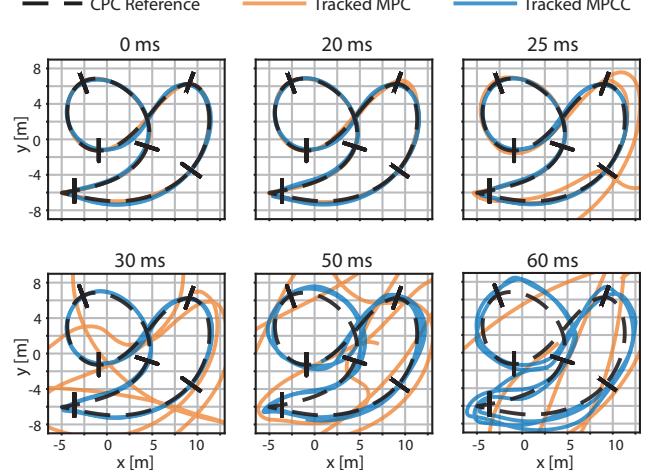


Fig. 11: Flight performance comparison between the proposed MPCC and a standard MPC when tracking the CPC trajectory for different delays on the state estimate. The proposed MPCC can handle delays of up to 50 ms while still passing through the gates, whereas a standard MPC controller fails to track the reference at 25 ms delay.

delay is present. However, MPCC benefits from the liberty to select the best states at runtime and therefore can react (up to some extent) to a delay in the loop.

## VI. EXPERIMENT: REAL-WORLD DRONE RACING

To further show the capabilities of the proposed MPCC for drone racing in the real world, in this section, we deploy the algorithm on the physical platform, shown in Fig. 12. This platform has been built in-house from off-the-shelf components and consists of an NVIDIA Jetson TX2<sup>1</sup> board that bridges the control commands via a Laird RM024<sup>2</sup> wireless low-latency module. Our method runs in an offboard desktop computer equipped with an Intel(R) Core(TM) i7-8550 CPU @ 1.80GHz. A Radix FC board that contains the Betaflight<sup>3</sup> firmware is used as a low level controller. This low-level controller takes as inputs body rates and collective thrusts. The proposed MPCC, however, uses all the dynamics of the system down to rotor thrusts. Therefore, for the real-world implementation, we use body rates and collective thrusts as inputs. It is, however, still advantageous that the MPCC solves the optimization problem, including full state dynamics. This way, the body rates and collective thrust are generated such that they lead to states that respect the real motor constraints.

For state estimation, we use a VICON<sup>4</sup> system with 36 cameras in one of the world’s largest drone flying arenas ( $30 \times 30 \times 8$  m) that provide the platform with down to millimeter accuracy measurements of position and orientation.

### A. Implementation Details

In order to deploy our MPCC controller, problem (16) needs to be solved in real-time. To this end, we use the ACADO<sup>5</sup>

<sup>1</sup><https://developer.nvidia.com/EMBEDDED/jetson-tx2>

<sup>2</sup><https://www.lairdconnect.com/wireless-modules/ramp-ism-modules>

<sup>3</sup><https://betaflight.com/>

<sup>4</sup><https://www.vicon.com/>

<sup>5</sup><https://acado.github.io/>



Fig. 12: The quadrotor platform used for real-world experiments. The achievable thrust-to-weight ratio is 4. The platform is equipped with a Laird module for real-time wireless communication, off-the-shelf components, and reflecting markers for the VICON localization system.

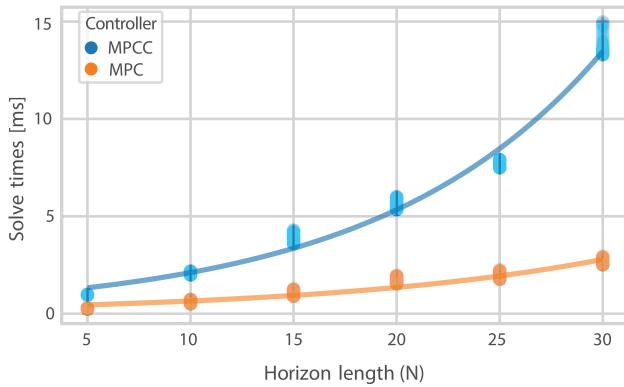


Fig. 13: Evolution of solver timings of MPCC (optimization problem (16)) and MPC (as in [12]), with respect to horizon length ( $N$ ). The optimization problems are solved on a desktop computer equipped with Intel(R) Core(TM) i7-8550 CPU @ 1.80GHz.

toolkit as a code generation tool, together with QPOASES<sup>6</sup> as a QP solver. The optimization problem is solved following a real-time iteration scheme [26] with a feedback rate of 100 Hz and a prediction rate of 16.6 Hz.

### B. Solver Timings

Fig. 13 shows the evolution of the solver timings for different horizon lengths, for both MPCC (optimization problem (16)) and a standard MPC (as in [12]). For MPCC to be able to call the solver every 10 ms, the maximum horizon length would be  $N = 25$ . However, that would leave the system with limited computation time for other modules that also need to run during the remaining time, such as the spline computation (explained in Section III-C). In order to ensure that the entire control pipeline runs smoothly at all times, we have selected a horizon length of  $N = 20$  for all our MPCC experiments in both simulation and the real world. This gives a solver time of about 5 ms.

<sup>6</sup><https://projects.coin-or.org/qpOASES>

### C. Baselines

To benchmark the performance of our approach in real-world flight, we compare it to (i) the fastest executed CPC trajectory in [12] and (ii) a world-class professional drone racing pilot (Timothy Trowbridge, whose list of achievements was previously reported in [12]).

It is important to remark that by definition, CPC trajectories take full advantage of the available actuator power at all times. Therefore they leave no actuation power for control authority under model mismatches and disturbances, and even the smallest deviation could potentially lead to a crash. While this might not be an issue in simulated environments, it definitely is in real-world flights. Because of this, in order to provide robustness and add a control authority safety margin, for the real flight experiments in [12] the authors planned the trajectory using 3.3 as thrust-to-weight ratio (TWR) but executed it with slightly more thrust, of around 4 TWR. To have a faithful comparison, we use exactly the same platform with the same limits in our experiments.

The platform used by the professional drone pilot is very similar to the one used to deploy our algorithm (shown in Fig. 12), where the Jetson TX2 and the Laird modules were substituted by an FPV camera system and an RC receiver. Reflective markers were also attached to the human platform to allow precise data collection. The TWR of this platform was matched to be the same as the one used for our real-world experiments.

### D. Results

A video of the experimental results of this work can be found in <https://youtu.be/mHDQckqdg4>. These experiments serve as proof of concept of the real-world applicability of the proposed controller.

Fig. 14 shows our results in the top row and the baselines in the bottom row. From left to right and top to bottom, we show the real flight data from our MPCC tracking the PMM reference, our MPCC tracking the CPC reference, a standard MPC tracking the CPC reference (as in [12]), and a world-class professional human expert. The CPC time-optimal reference has been added with transparency to the PMM-MPCC plot and to the human expert plot for comparison purposes only. It is interesting to notice how the PMM-MPCC executed trajectory approaches the theoretical full model time-optimal (labeled as *CPC reference*), as already suggested in the simulation experiments (Fig. 9).

Fig. 15 shows the lap time distribution of our MPCC approach (tracking both CPC and PMM references) compared to those of the standard MPC tracking a CPC reference and the human expert. As an ablation study, we show how our approach performs for different values of  $\mu$ . As shown by the original MPCC formulation in (6), the progress weight contributes mainly to making the trajectory go faster by trading it off for contour error. In this same figure, we notice that when using the CPC path as reference to the MPCC controller, the lap times are faster than when using the PMM as reference, in concordance with the results and conclusions from the ablation

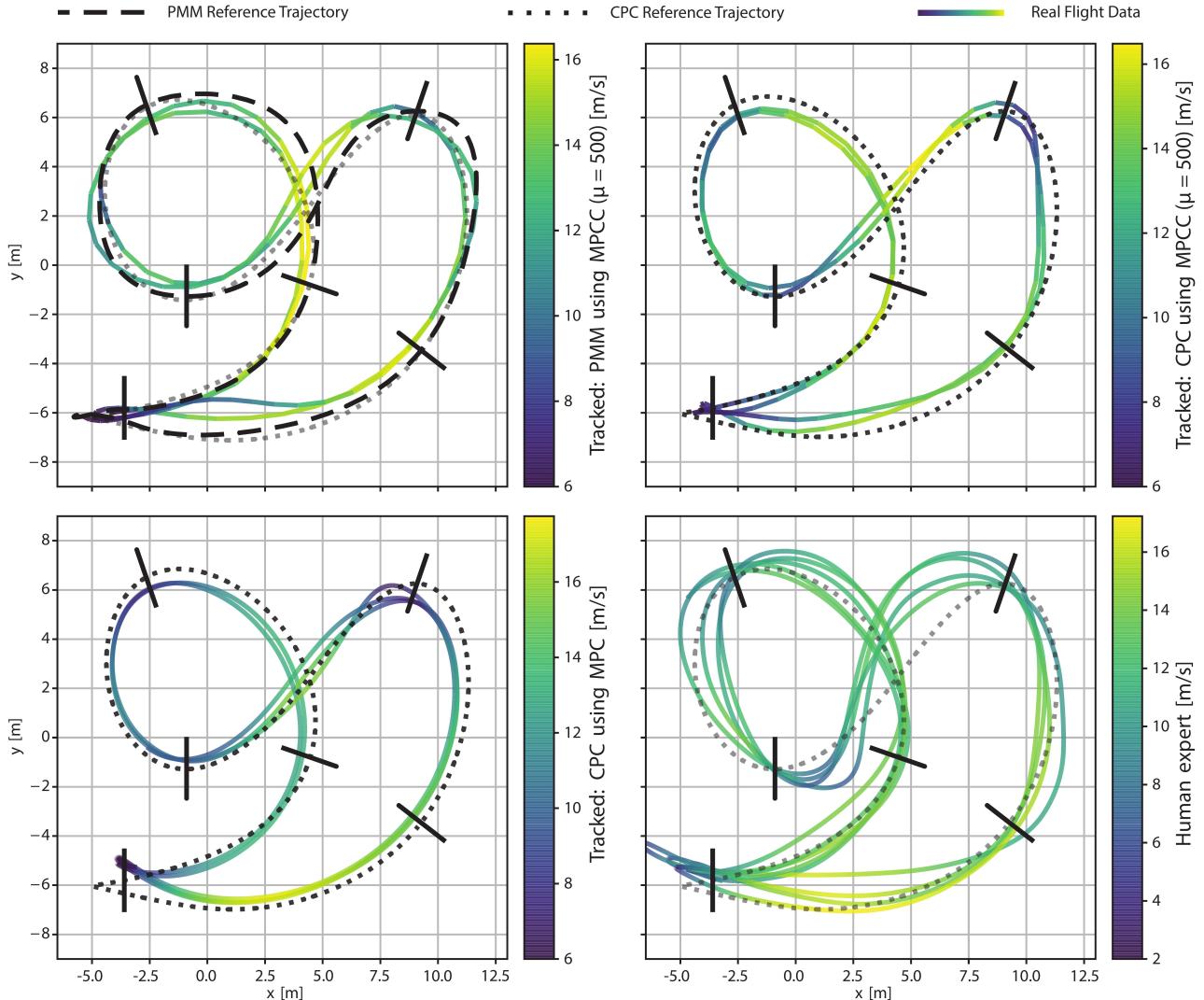


Fig. 14: Real world experiment of the MPCC tracking a time-optimal trajectory generated using a point mass model (PMM Reference, top left), and of the MPCC tracking a true time-optimal trajectory (CPC Reference, top right), both using  $\mu = 500$ . We compare our results with the executed time-optimal trajectory (CPC tracked by MPC, bottom left), and with the best human expert from [12] (bottom right). Note how when using MPCC as a controller, the platform stays for longer times at high speeds regime.

study performed in Section V-B. In our experiments for larger progress costs than  $\mu = 500$ , the error around the gates starts being significantly larger, resulting in the platform passing too far from the gates and leading to crashes against the gate frames.

When comparing our MPCC method to the CPC tracked by MPC, we observe that even though the tracked MPCC trajectory has a slightly lower maximum velocity (color bars in Fig. 14), the lap times are better for our MPCC approach (see Fig. 15,  $MPCC \mu = 500$  vs.  $MPC$ ). This can be explained by the fact that our MPCC approach can effectively take better advantage of the control authority safety margin (already introduced in Section VI-C). The proposed MPCC controller benefits from the freedom to decide the future states of the platform online, meaning that it is able to adapt and command full thrust when possible or re-gain control authority when necessary. Consequently, our MPCC approach results in higher speeds for longer periods of time, as can be noted in the col-

ored profile in Fig. 14 and in the flight speed histogram in Fig. 16. In contrast, the MPC approach is based on minimizing the error with respect to the pre-computed reference. Therefore, in order to fly faster, a new reference planned with higher TWR would need to be generated, effectively reducing the control authority safety margin and increasing the likelihood of failure. As a side note, these effects are not noticeable in our simulated environments (Section V-B) because there are no major disturbances in our simulation. Therefore, there is no need to add a control authority safety margin, meaning that the *MPC* approach is able to track the trajectory with the same TWR with which it was planned.

Additionally, as shown in Section V-C, the proposed MPCC approach is more robust at handling delays than standard MPC. Since all real systems are prone to have delays (in our experiments, mainly coming from the communication involving the motion capture system and the LAIRD module), this also potentially contributes to the advantage of our MPCC

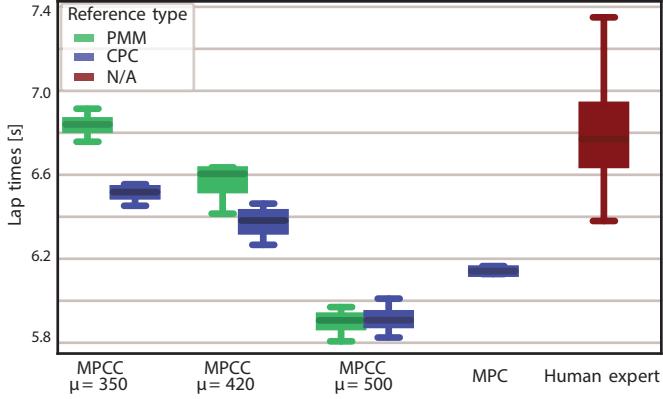


Fig. 15: Lap times of our MPCC approach tracking both the PMM and the CPC references, for different values of progress cost  $\mu$ . We compare it to the execution of the CPC trajectory by a standard MPC controller and to a human expert flying the same track.

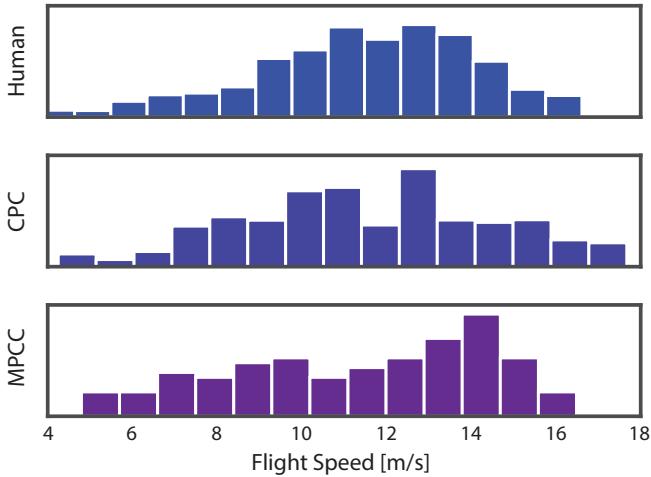


Fig. 16: Histogram of flight speed of different control approaches. From top to bottom: human expert trajectory, CPC trajectory tracked with standard MPC, and CPC trajectory tracked with the proposed MPCC. Note how the MPCC controller spends more time at higher speeds.

approach.

When comparing the autonomously flown trajectories to the human expert, one can notice how the human trajectories are opening more and overshooting in most turns. This is related to the vision-based first-person view flying behavior since human pilots need to orient the camera toward the upcoming waypoints by applying coordinated yaw, pitch, and roll commands. Moreover, our recent work indicates that human pilots use long prediction horizons related to visual-motor response latencies in the order of 220 ms [51].

## VII. DISCUSSION

### A. Convexity

The convexity of the cost function (3) depends solely on the convexity of  $\|e_{c,k}\|_{q_c}^2$ , since  $\rho\theta_N$  is an affine term of an optimization variable and therefore does not affect convexity. It follows that optimization problem (16) is non-convex due to (i) as  $p^d(\theta_k)$  is formulated as a set of third-order splines,

$e(\theta_k)$  is also a third-order polynomial, and  $t(\theta_k)$  is of second order. Hence, the norm of the contour (10) and lag (9) errors result in polynomials of degrees 24 and 12, respectively. And (ii) the rotational dynamics of the system (11) are non-linear. Consequently, problem (16) is a Non-Linear Program (NLP), and different strategies can be taken to solve it in real-time. In the literature, this is commonly approached by manually carrying out the linearization of the errors and the dynamics around the previous prediction, effectively converting the NLP into a Quadratic Program (QP) [21], [20]. In other approaches, an interior point method solver is chosen to solve the NLP directly [41]. In our work, the optimization problem follows an RTI scheme [26]. The non-linear problem is directly fed to ACADO, which carries out a linearization at every time step to convert the NLP into a QP (preparation phase) that is then solved by QPOASES (feedback phase).

### B. Tuning

Controller tuning is one of the main challenges among the control community. The sensitivity to control tuning is particularly pronounced when talking about aggressive, high-speed quadrotor flight [52]. Although in the cost function from problem (16) there are many different weights, most of these parameters act as regularization terms ( $Q_w, r_{\Delta v}, R_{\Delta f}$ ) not to allow the controller to choose arbitrarily big control inputs; and  $q_l$ , which is chosen big enough such that the error approximations are sufficiently small. Once these terms are tuned, they are kept constant for all different conditions in both simulation and real-world experiments.

That leaves only two different scalar values to tune,  $q_c$  and  $\mu$ . Therefore, the benefits of using MPCC as control strategy in comparison to standard methods are not only related to the liberty the controller offers by choosing the best sampling times and states but also that it significantly reduces the dimensionality of the hyperparameter space and, therefore, the tuning effort. Additionally, due to this low dimensionality, data-driven approaches that find the optimal tuning given a higher level objective can be explored in the future. For instance, reinforcement-learning-based approaches have shown to be promising in previous work [53].

Another advantage of MPCC is that the speed at which the platform tracks the path can be selected by changing the progress weight  $\mu$  accordingly. In contrast, for a standard reference tracking control method (such as MPC), if the desired speed of the platform needs to be changed, the entire reference needs to be computed again.

### C. Limitations

Albeit the proposed controller has shown to achieve outstanding performance in highly agile quadrotor flight, it currently runs on an external desktop PC (Intel(R) Core(TM) i7-8550 CPU @ 1.80GHz) at 5 ms. Since the optimization problem (16) to be solved in each iteration is quite complex, it is currently difficult to run it onboard embedded computers like the NVIDIA Jetson TX2. This could be tackled by using new state-of-the-art solvers specifically designed for Model Predictive Control formulations that have shown to outperform QPOASES [54].

### VIII. CONCLUSION

This paper proposes a Model Predictive Contouring Control formulation in the framework of full quadcopter dynamics. We highlight and exploit the main advantages of this formulation and show that it is very well suited for the problem of drone racing. In particular, we use a sampling-based method to generate non-feasible, close-to-time-optimal trajectories that can then be tracked with our controller. Additionally, we encode the gates in the optimization problem by adapting the contouring weights in the cost function depending on the gates' location. We compare our controller to a standard MPC tracking full state time-optimal trajectories and show that they render superhuman performance in both simulation and real-world experiments, with the advantage that our approach could be re-planned in real-time. Additionally, we show that, when deployed in the real platform, our MPCC approach is able to take better advantage of the full actuator potential than a standard MPC approach tracking a time-optimal trajectory, resulting in better lap times.

The results from this work bring the field of autonomous drone racing closer to beating the best humans in a fair drone race. Although there are still challenges ahead mainly related to onboard state estimation and perception at high speeds, the proposed controller has proven to be a stable approach to fly very aggressive maneuvers at very high speeds.

Finally, the control method presented in this work can find utility in other applications. Specifically, the fact that our method can track non-feasible trajectories and that it allows the selection of desired speeds reduces the required computation effort needed for planning. This opens the door to, for example, several sampling-based approaches where the dynamic feasibility constraint could be dropped.

### REFERENCES

- [1] G. Loianno and D. Scaramuzza. Special issue on future challenges and opportunities in vision-based drone navigation. *J. Field Robot.*, 2020. doi: 10.1002/rob.21962.
- [2] Suchithra Rajendran and Sharan 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*, 143:102090, Nov 2020. ISSN 13665545. doi: 10.1016/j.tre.2020.102090. arXiv: 2103.01768.
- [3] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *Int. J. Robot. Research*, 2012. doi: 10.1177/0278364911434236.
- [4] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu. *IEEE Robotics and Automation Letters*, 2(2):404–411, 2017.
- [5] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 690–696, 2018.
- [6] Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico, Konstantinos Karydis, Nikolay Atanasov, Giuseppe Loianno, Davide Scaramuzza, Kostas Daniilidis, Camillo Jose Taylor, and Vijay Kumar. Fast, autonomous flight in gps-denied and cluttered environments. *J. Field Robot.*, 35(1):101–120, 2018. doi: 10.1002/rob.21774.
- [7] Boyu Zhou, Jie Pan, Fei Gao, and Shaojie Shen. RAPTOR: robust and perception-aware trajectory replanning for quadrotor fast flight. *IEEE Transactions on Robotics*, 2021.
- [8] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza. Alphapilot: Autonomous drone racing. *Robotics: Science and Systems (RSS)*, 2020. URL <https://link.springer.com/article/10.1007/s11370-018-00271-6>.
- [9] Shuo Li, Michaël MOI Ozo, Christophe De Wagter, and Guido CHE de Croon. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. *Robotics and Autonomous Systems*, 133, 2020.
- [10] Huan Nguyen, Mina Kamel, Kostas Alexis, and Roland Siegwart. Model predictive control for micro aerial vehicles: A survey. *arXiv:2011.11104 [cs]*, Nov 2020. URL <http://arxiv.org/abs/2011.11104>. arXiv: 2011.11104.
- [11] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, Jul 2020. ISBN 978-0-9923747-6-1. doi: 10.15607/RSS.2020.XVI.040. URL <http://www.roboticsproceedings.org/rss16/p040.pdf>.
- [12] Philipp Foehn, Angel Romero, and Davide Scaramuzza. Time-optimal planning for quadrotor waypoint flight. *Science Robotics*, 6(56), 2021. doi: 10.1126/scirobotics.abb1221. URL <https://robotics.sciencemag.org/content/6/56/eabb1221>.
- [13] H. Moon, J. Martinez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. Ozo, C. De Wagter, G. de Croon, S. Hwang, S. Jung, H. Shim, H. Kim, M. Park, T.-C. Au, and S. J. Kim. Challenges and implemented technologies used in autonomous drone racing. *J. Intell. Service Robot.*, 2019. doi: 10.1007/s11370-018-00271-6.
- [14] José Arturo Cocoma-Ortega and José Martínez-Carranza. Towards high-speed localisation for autonomous drone racing. In *Mexican International Conference on Artificial Intelligence*. Springer, 2019.
- [15] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and A. Kapoor. Airsim drone racing lab. *PMLR post-proceedings of the NeurIPS 2019's Competition Track*, 2020.
- [16] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.
- [17] CORDIS - European Comission. AgileFlight. <https://cordis.europa.eu/project/id/864042>. Accessed: 2021-7-30.
- [18] Gilhyun Ryou, Ezra Tal, and Sertac Karaman. Multi-fidelity black-box optimization for time-optimal quadrotor maneuvers. In *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, Jul 2020. ISBN 978-0-9923747-6-1. doi: 10.15607/RSS.2020.XVI.032. URL <http://www.roboticsproceedings.org/rss16/p032.pdf>.
- [19] Helmut Maurer. On optimal control problems with bounded state variables and control appearing linearly. *SIAM Journal on Control and Optimization*, 15(3):345–362, 1977.
- [20] D. Lam, C. Manzie, and M. Good. Model predictive contouring control. In *49th IEEE Conference on Decision and Control (CDC)*, page 6137–6142, Dec 2010. doi: 10.1109/CDC.2010.5717042.
- [21] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, Sep 2015. ISSN 01432087. doi: 10.1002/oca.2123. arXiv: 1711.07300.
- [22] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010.

- [23] Emil Fresk and George Nikolakopoulos. Full quaternion based attitude control for a quadrotor. In *2013 European control conference (ECC)*, pages 3864–3869. IEEE, 2013.
- [24] Dario Brescianini and Raffaello D’Andrea. Tilt-prioritized quadrocopter attitude control. *IEEE Transactions on Control Systems Technology*, 28(2):376–387, 2018.
- [25] M. Bangura and R. Mahony. Real-time model predictive control for quadrotors. *IFAC World Congress*, 2014. doi: 10.3182/20140824-6-za-1003.00203.
- [26] M. Diehl, H. G. Bock, H. Diedam, and P. B. Wieber. Fast direct multiple shooting algorithms for optimal robot control. In *Fast motions in biomechanics and robotics*. Springer, 2006.
- [27] James Svacha, Kartik Mohta, and Vijay Kumar. Improving quadrotor trajectory tracking by compensating for aerodynamic effects. In *2017 international conference on unmanned aircraft systems (ICUAS)*, pages 860–866. IEEE, 2017.
- [28] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.
- [29] Ezra Tal and Sertac Karaman. Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. *IEEE Transactions on Control Systems Technology*, 2020.
- [30] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011. doi: 10.1109/ICRA.2011.5980409.
- [31] P. Foehn, D. Falanga, N. Kuppuswamy, R. Tedrake, and D. Scaramuzza. Fast trajectory optimization for agile quadrotor maneuvers with a cable-suspended payload. In *Robotics: Science and Systems (RSS)*, 2017.
- [32] M. Hehn, R. Ritz, and R. D’Andrea. Performance benchmarking of quadrotor systems using time-optimal control. *Auton. Robots*, March 2012. doi: 10.1007/s10514-012-9282-3.
- [33] W. Van Loock, G. Pipeleers, and J. Swevers. Time-optimal quadrotor flight. In *IEEE Eur. Control Conf. (ECC)*, 2013. doi: 10.23919/ECC.2013.6669253.
- [34] Sara Spedicato and Giuseppe Notarstefano. Minimum-time trajectory generation for quadrotors in constrained environments. *IEEE Transactions on Control Systems Technology*, 26(4):1335–1344, 2017.
- [35] R. Allen and M. Pavone. *A Real-Time Framework for Kinodynamic Planning with Application to Quadrotor Obstacle Avoidance*. 2016. doi: 10.2514/6.2016-1374.
- [36] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 2872–2879, Sep 2017. doi: 10.1109/IROS.2017.8206119.
- [37] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in  $se(3)$ . *IEEE Robotics and Automation Letters*, 3(3):2439–2446, Jul 2018. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2018.2795654.
- [38] Timothy R Jorris and Richard G Cobb. Three-dimensional trajectory optimization satisfying waypoint and no-fly zone constraints. *Journal of Guidance, Control, and Dynamics*, 32(2):551–572, 2009.
- [39] K Bousson and Paulo FF Machado. 4d trajectory generation and tracking for waypoint-based aerial navigation. *WSEAS* *Transactions on Systems and Control*, (3):105–119, 2013.
- [40] Timm Faulwasser, Benjamin Kern, and Rolf Findeisen. Model predictive path-following for constrained nonlinear systems. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 8642–8647. IEEE, 2009.
- [41] W. Schwarting, J. Alonso-Mora, L. Paull, S. Karaman, and D. Rus. Safe nonlinear trajectory generation for parallel autonomy with a dynamic vehicle model. *IEEE Transactions on Intelligent Transportation Systems*, 19(9):2994–3008, Sep 2018. ISSN 1558-0016. doi: 10.1109/TITS.2017.2771351.
- [42] Jialin Ji, Xin Zhou, Chao Xu, and Fei Gao. Cmpcc: Corridor-based model predictive contouring control for aggressive drone flight. *arXiv:2007.03271 [cs]*, Feb 2021. URL <http://arxiv.org/abs/2007.03271>. arXiv: 2007.03271.
- [43] Bruno Brito, Boaz Floor, Laura Ferranti, and Javier Alonso-Mora. Model predictive contouring control for collision avoidance in unstructured dynamic environments. *IEEE Robotics and Automation Letters*, 4(4):4459–4466, 2019.
- [44] Lie Tang and Robert G Landers. Predictive contour control with adaptive feed rate. *IEEE/ASME Transactions On Mechatronics*, 17(4):669–679, 2011.
- [45] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. *arXiv:2103.08624 [cs]*, Mar 2021. URL <http://arxiv.org/abs/2103.08624>. arXiv: 2103.08624.
- [46] Hongling Wang, Joseph Kearney, and Kendall Atkinson. Arc-length parameterized spline curves for real-time simulation. Jan 2002.
- [47] M. Faessler, A. Franchi, and D. Scaramuzza. Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories. *IEEE Robot. Autom. Lett.*, April 2018. doi: 10.1109/LRA.2017.2776353.
- [48] Michael Neunert, Cedric de Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, page 1398–1404. IEEE, May 2016. ISBN 978-1-4673-8026-3. doi: 10.1109/ICRA.2016.7487274. URL <http://ieeexplore.ieee.org/document/7487274>.
- [49] M. W. Mueller, M. Hehn, and R. D’Andrea. A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2013. doi: 10.1109/iros.2013.6696852.
- [50] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000. ISBN 1886529094.
- [51] Christian Pfeiffer and Davide Scaramuzza. Human-piloted drone racing: Visual processing and control. *IEEE Robotics and Automation Letters*, 6(2):3467–3474, Apr 2021. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2021.3064282.
- [52] Antonio Loquercio, Alessandro Saviolo, and Davide Scaramuzza. Autotune: Controller tuning for high-speed flight. *arXiv:2103.10698 [cs]*, Mar 2021. URL <http://arxiv.org/abs/2103.10698>. arXiv: 2103.10698.
- [53] Yunlong Song and Davide Scaramuzza. Learning high-level policies for model predictive control. page 8.
- [54] G. Frison and M. Diehl. Hpipm: a high-performance quadratic programming framework for model predictive control. *ArXiv*, 2020. doi: 10.1016/J.IFACOL.2020.12.073.