ADVANCED INTELLIGENT SYSTEMS
Open Access
www.advintellsyst.com

Low Computational Cost for Multiple Waypoints Trajectory Planning: A Time-Optimal-Based Approach

Da-hui Lin-Yang, Francisco Pastor, and Alfonso J. García-Cerezo*

In the field of mobile robots, achieving minimum time in executing trajectories is crucial for applications like delivery, inspection, and search and rescue. In this article, a novel time-optimal planner based on optimization methods is introduced. Despite the high computational cost associated with such methods, the solution calculates time-optimal multi-waypoint trajectories, achieving results in the order of milliseconds. The proposed method formulates a time-optimal trajectory using the Pontryagin's maximum principle as a policy. By utilizing a point mass model, the planner generates trajectories that are adaptable to different robot models. The approach incorporates a definition of a search space to guarantee convergence while considering the system limits. Simulation and real-world experiments are performed to validate the feasibility of our method with different configurations. Simulation results compared to a benchmark method demonstrate our approach's superior performance in terms of computational time, achieving near-optimal solutions. In addition, in the real-world experiments, the integration of the method into practical applications is validated.

1. Introduction

The challenge of achieving efficient navigation in the context of mobile autonomous robotic systems remains an ongoing problem within the field of robotics research. There are several navigation problems where the performance is directly related to the time spent to reach a goal, considering energy consumption issues, or the urgent requirement for fast arrival. These requirements are needed in fields such as planetary exploration, delivery, racing,[1] or search and rescue.[2]

The objective of a time-optimal problem is to minimize the time (T) required to complete a particular task or to fulfill an objective. Several works have addressed the time-optimal problem for autonomous navigation using optimization methods,[3,4] searching the minimum time according to certain constraints. Although they provide high-performance solutions, time optimization methods may not be suitable for real-time applications due to their high computational cost. Their computation times, which are on the order of seconds, pose a significant limitation, rendering them unsuitable for closed-loop execution. This capability is crucial for achieving advanced interaction and reactivity in dynamic environments, as exemplified in autonomous driving scenarios.[5–8]

Sometime optimization problems do not only focus on a single goal, but they rather involve multiple intermediate waypoints that the robot must visit in a certain order. In these particular cases, additional techniques must be included. The multi-waypoint problem can be addressed by implementing online allocation techniques, as shown in ref. [9]. This method simultaneously solves the time-optimal and time-allocation problems, providing a minimum time trajectory. However, this approach introduces additional computational cost per iteration.

Another approach is introduced in ref. [10], where Foehn et al. proposed a sampling method for time-optimal planning with a sequence of waypoints. In this method, the planner generates primitive trajectories over the waypoints to find the optimal solution. These trajectories are constructed based on a closed-form solution using a point mass model (PMM).[11] This approach allows to sample trajectories with minimal computation cost providing a solution in a low time. Nevertheless, as random sampling is employed, the method could result in suboptimal trajectories. Recent works use reinforcement learning techniques[12] to combine the planning and the control task for predefined racing circuits. These approaches present outstanding results; however, they are dependable on the training data and must be retrained for new configurations.

Although optimization-based planners generate optimal solutions, computational times are generally too high for closed-loop applications. In this work, we present a time-optimal planner using optimization-based methods with low computational cost. In addition, the trajectory planner can generate trajectories for multiple waypoints. Our approach formulates the time-optimal trajectory mathematically using a bang-bang policy and the Pontryagin's maximum principle. We also consider the robot's representation as a PMM. With this simplification, the planner is able to adapt to different robotic systems, providing primitive trajectories which are tracked by lower level algorithms. Afterward, we initialize PMM trajectories to connect pairs of

D. Lin-Yang, F. Pastor, A. J. García-Cerezo
Robotics and Mechatronics Lab, Institute for Mechatronics Engineering & Cyber-physical Systems (IMECH.UMA)
Universidad de Málaga
29071 Málaga, Spain
E-mail: dahuilinyang@uma.es

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/aisy.202400363.

DOI: 10.1002/aisy.202400363

waypoints, and then, we optimize the trajectories to seek the minimum time.

Summarizing, the main contributions of this work are as follows. 1) A time-optimal method that solves a multi-waypoint trajectory planning with low computation cost. To do so, we optimize a sequence of primitive trajectories subject to waypoints using the bang-bang policy and the PMM. 2) An initialization that establishes a search space for the optimization algorithm to guarantee the convergence of the planner considering the limits of the system. 3) Evaluation and integration of the proposed method with two experiments carried out in a simulation environment and in a real-world scenario.

This article is structured as follows: Section 2 introduces related works on trajectory planning and, specifically, time-optimal planning, followed by allocation methods for multi-waypoints. Section 3 presents the mathematical formulation of the time-optimal trajectory planning method, along with the proposed optimization strategy. Section 4 details the experiments conducted in both simulation and real-world environments to evaluate our method. Section 5 discusses the result of the experiments and validates the proposed planner. Finally, Section 6 relates the conclusions and future research directions of the article.

2. Related Works

2.1. Efficient Trajectory Planning

The goal of trajectory planning is to find a sequence of states that allows the robot to move from an initial configuration to a goal, while satisfying specific criteria. Numerous trajectory planning techniques have been developed, each with distinct trade-offs in terms of complexity, efficiency, and safety. The artificial potential fields method[13] is a commonly used approach that offers a computationally efficient solution to safe navigation within an environment. Nevertheless, the formation of local minima can get the robot trapped, limiting its reliability and applicability. The dynamic window approach[14] is another efficient method for local trajectory planning, focusing on generating collision-free paths using velocity commands. However, its effectiveness diminishes in highly dynamic environments.

Recent advances have focused on model predictive control (MPC), which provides real-time trajectory optimization and collision avoidance solutions, as demonstrated in ref. [15]. Despite its effectiveness, the high computational cost of the MPC can restrict its application in environments with complex dynamics. An example of these type of environments is the multi-robot planning problem, where coordination and computational efficiency are critical. To address this issue, a distributed motion planning approach based on model predictive contouring control (MPCC) has been proposed in ref. [16].

Machine-learning approaches have also gained significant attention, offering adaptive and data-driven solutions for trajectory planning. For instance, Sánchez et al.[17] explored the potential of reinforcement learning in motion planning for dynamic and uncertain environments. Other approaches propose hybrid solutions, as presented in ref. [18]. This work combines the robustness of the MPCC with the computational efficiency of long short-term memory networks, enabling real-time decision-making in complex and dynamic environments. Although the aforementioned methods focus on optimizing control inputs or identifying the shortest path, none of them are explicitly designed to minimize the trajectory time.

2.2. Time-Optimal Planning

Two main approaches are considered for time-optimal planners in the autonomous navigation field: continuous polynomial representation and discrete state representation. Polynomial representations are commonly used to generate continuous time trajectories for differential flat systems.[19] These methods are highly valued for their efficiency and low computational cost, making them ideal for real-time applications. For instance, in ref. [20], a trajectory planner based on a polynomial method is implemented for aggressive quadrotor flights. The proposed solution efficiently generates trajectories in environments with obstacles. Another polynomial approach is presented in ref. [21]. This work implements a fast trajectory planner for autonomous vehicles, generating high-speed trajectories in unknown environments; thus, replanning can be conducted. Despite of their efficiency, these methods only provide suboptimal solutions for time-optimal planning due to their inherent smoothness.

Unlike the polynomial representation, discrete-time approaches employ mathematical models that emulate the dynamics of real systems. In the case of discrete-time approaches such as search and sampling methods, the trajectory is obtained by sampling the robot inputs and searching for the best sequence state over a time-discretized interval. First search and sampling methods employed a linearized model such as in ref. [22]. However, this solution is limited to being near the linearization zone. Recent approaches[23] have introduced a search method using motion primitives based on polynomial trajectories to enable aggressive quadrotor flights. Despite its high performance, the planner's computational time is too high to be used in cluttered environments. Other search and sampling approaches simplify the system's dynamics to reduce the computational time such as the algorithm proposed in ref. [24]. Similarly, in ref. [1], time-optimal trajectories are generated for remote controlled (RC) racing purposes using a bicycle model. However, employing too many discrete time steps is not recommended due to the increment of the computational cost. Summarizing, sampling and search methods are not suitable for systems with a significant number of state variables and many time-discretize steps since the computational time increases significantly.

Among the discrete approaches, optimization-based methods provide high-quality results that mathematically guarantee an optimal solution. In contrast to the polynomial method, optimization methods can choose any input value at each discrete time step within predefined bounds. In addition, optimization approaches guarantee an optimal solution compared to sampling and search algorithms. With all these considerations, it is reasonable to conclude that optimization methods are the most suitable for trajectory planners for complex tasks that need high-quality solutions.[3,4] However, their main drawback is the substantial amount of computational time they require. Other methods

formulate the time-optimal problem with a traverse-dynamics model, as shown in refs. [7,8]. These approaches consider the trajectory's arc length as a progress variable which has to be maximized, removing the time variable from the optimization. They have been proven to be computationally faster than traditional time-optimal formulations. In fact, Van Loock et al.[8] executed their algorithm with a suitable frequency for autonomous driving with high velocities. While significant strides have been made in the development of optimization strategies for time-optimal planning, the integration of these methodologies into real-time applications remains a formidable challenge.

2.3. Allocation Time Methods for Multi-Waypoints

Allocation time methods for multi-waypoints involve efficiently planning trajectories and adjusting speeds to ensure that they pass through the waypoint constraints. Some approaches use Heuristic algorithms[25] to set up the segment times for a polynomial planner. However, the solutions obtained from these methods are suboptimal. Another consideration is to use iterative optimization methods[20] to search for minimum time trajectory for flights. This type of planner can generate safe indoor routes efficiently, although the polynomial-based approach used for trajectory planning leads to near time-optimal solutions. A time-optimal trajectory planner based on optimization methods has recently been developed in the aerial drone racing field.[9] It uses complementary progress constraint (CPC) for the time allocation problem and the full quadrotor model, and thus, it obtains a truly time-optimal solution. Nevertheless, the computational time is considered as high, on the order of several minutes, for solving the optimization problem. In contrast, Foehn et al. also presented in ref. [10] a fast time-optimal planning based on sampled methods for the multi-waypoint problem. With this approach, the trajectories are generated in a faster computational time, but they cannot guarantee the minimum time-optimal solution.

3. Time-Optimal Formulation for Multiple Waypoints

The navigation problem goal is to reach an end state (x_e) from an initial state (x_s), generating a feasible trajectory (τ). The problem is commonly represented mathematically as in Equation (1):

$$\underset{\tau}{\text{minimize}}\ T \tag{1a}$$

$$\text{s.t.}\quad \tau(0) = x_s \tag{1b}$$

$$\tau(T) = x_e \tag{1c}$$

$$\tau(t) \in \chi \text{ for } t \in [0, T] \tag{1d}$$

where T is the time spent to reach x_e and χ is a feasible set of trajectories to transition from the initial state to the final state. Note that χ is dependent on the equations used to model the system dynamics.

In this work, we assume the system to be a 3D point-mass model. Specifically, we represent the translational component of the system using motion primitives derived from a PMM. This assumption is commonly employed in the autonomous navigation field to generate higher-level trajectories for aerial robots[10] or autonomous vehicles.[26]

The dynamics of a 3D point-mass are represented as a double integrator ($\ddot{p} = u$), where the state of the system is position (p) and velocity (v), and the variable control is the acceleration (u). Since the point-mass model provides a simplified representation of the system's dynamics, we can solve the optimization problem (Equation (1)) individually per axis at first. Then, we unify the solutions through actuation scaling, which will be described in detail in Section 3.3.

3.1. Pontryagin's Maximum Principle

We solve the time-optimal problem using the Pontryagin's maximum principle,[27] which states that acceleration input ($u(t)$) for a point-mass has a form of a bang-bang control:

$$u^*(t) = \begin{cases} u_{\max} & \text{if } 0 \le t \le t_1 \\ -u_{\max} & \text{if } t_1 \le t \le T^* \end{cases} \tag{2}$$

The optimal control $u^*(t)$ for a point-mass takes on the maximum value $u_{\max}$ for the period of time t_1, and then it switches to the minimum value $-u_{\max}$ for the remaining time $t_2 = T^* - t_1$, or vice versa.

According to the previous policy, as proven in ref. [28], we can compute the trajectory between two points with the following expression:

$$p_1 = p_0 + v_0 t_1 + \frac{b}{2} u_{\max} t_1^2 \tag{3a}$$

$$v_1 = v_0 + b u_{\max} t_1 \tag{3b}$$

$$p_2 = p_1 + v_1 t_2 - \frac{b}{2} u_{\max} t_2^2 \tag{3c}$$

$$v_2 = v_1 - b u_{\max} t_1 \tag{3d}$$

The trajectory of Equation (3) defines the position (p_1 and p_2) and velocity (v_1 and v_2) at the times t_1 and t_2, given an initial state (p_0, v_0). In addition, b is a switching binary parameter that defines the input policy, taking the values of 1 or -1.

3.2. Time-Optimal Problem for Multi-Waypoints

The time-optimal problem in Section 3 is reformulated, so the resulting trajectory goes through a sequence of waypoints before reaching the end state. To do so, a trajectory τ with duration T is now defined as a subset of trajectories (τ_i) with their corresponding times (T_i^*). Considering the number of waypoints (**wp**) to visit as N, we describe the trajectory as follows:

$$\begin{matrix} \tau = \{\tau_0, \tau_1, \ldots, \tau_i, \ldots, \tau_N\} \\ T = \{T_0^*, T_1^*, \ldots, T_i^*, \ldots, T_N^*\} \end{matrix} \; i \in 0, 1, 2, \ldots, N \tag{4}$$

Hereinafter, we refer to the subindex i as every subtrajectory of τ. The solution of the optimization must satisfy the following requirements. 1) The trajectory $\tau(t)$ must be continuous over the time interval $t \in [0, T]$. Consequently, the position of every $\tau_i(T_i^*)$ must be coincident with $\tau_{i+1}(0)$. 2) Similarly, the velocity

has to be continuous over the time interval $t \in [0, T]$. Therefore, the velocities of $\tau_i(T_i^*)$ and $\tau_{i+1}(0)$ are coincident. 3) The subtrajectory τ_0 must start with the actual system state and the subtrajectory τ_N must finish with the end state.

Following these statements, the optimization of a time-optimal multi-waypoint problem is formulated as follows:

$$\underset{\tau_0, \tau_1, \tau_2, \tau_3 \ldots}{\text{minimize}} T = \sum_{i=0}^{N} T_i^* \tag{5a}$$

$$\text{s.t.} \quad \tau_0(0) = x_s \tag{5b}$$

$$\tau_N(T_N^*) = x_e \tag{5c}$$

$$\tau_i(t_i) \in \chi \text{ for } t_i \in [0, T_i^*] \tag{5d}$$

$$\tau_i^p(0) = wp_i \tag{5e}$$

$$\tau_{i-1}(T_{i-1}^*) = \tau_i(0) \text{ for } i \in [1, 2, \ldots, N] \tag{5f}$$

The subset of trajectories τ_i is subject to the definition presented in Equation (3), constraining the solution exclusively to a bang-bang policy. Consequently, T depends exclusively on the velocities in the waypoints.

The optimization problem presented in Equation (5) is addressed with the PMM, where each axis is solved independently. The resulting expression is formulated for an optimization problem, leading to

$$\underset{\chi}{\text{minimize}} \; T = \sum_{i=0}^{N} t_{1_i} + t_{2_i} \tag{6a}$$

$$\text{s.t.} \quad \tau_0(0) = x_s \tag{6b}$$

$$\tau_N(t_{1_N} + t_{2_N}) = x_e \tag{6c}$$

$$p_{2_i} = wp_i \tag{6d}$$

$$p_{2_{i-1}} = p_{0_i} \tag{6e}$$

$$v_{2_{i-1}} = v_{0_i} \tag{6f}$$

$$v_{\min_i} \leq v_{2_i} \leq v_{\max_i} \tag{6g}$$

$$0 \leq t_{1_i} \tag{6h}$$

$$0 \leq t_{2_i} \tag{6i}$$

$$(3) \text{ for all } \tau_i \tag{6j}$$

where χ is the whole set of variables $t_{1_i}, t_{2_i}, p_{1_i}, v_{1_i}, v_{2_1}$ that belongs to every subtrajectory τ_i; and the parameters are the initial state (x_s), final state (x_e), and the positions of the intermediate waypoints. To ease the notation, hereinafter, we refer to x_e as the N waypoint position with null velocity.

Algorithm 1 describes the whole process of generating a new trajectory. The algorithm needs the set of N waypoints and the initial state of the robot as inputs. We create three lists, each corresponding to an axis containing the spatial coordinates of the N waypoints (line 1). Then, we calculate the initialization values and specific limits to create a search space π (lines 2 and 4) with **CalculateInitialGuess** (**Algorithm 2**). Once the optimization process is performed on every axis (line 5), a **Scaling** procedure is executed to synchronize the previously computed optimized times (line 6), denoted as T^x, T^y, and T^z, with the superindex referring to each axis. This **Scaling** process will be further explained in Section 3.3.

Algorithm 1. Optimal 3D trajectory generation.

Input: List of Waypoints **wp**, Initial State x_s
Output: Trajectory τ
$(\mathbf{wp}^x, \mathbf{wp}^y, \mathbf{wp}^z) \leftarrow$ **Split**(**wp**)
$\mathbf{b}, \mathbf{v}_{\min}, \mathbf{v}_{\max}, \tau^o \leftarrow$ **CalculateInitialGuess**(**wp**, x_s)
for $k \in [x, y, z]$ **do**
  $\pi \leftarrow$ **SetSearchSpace**($[\mathbf{b}^k, \mathbf{v}_{\min}^k, \mathbf{v}_{\max}^k, \tau^{o^k}]$)
  $(T^k, \tau^k) \leftarrow$ Solve with eq. (5) $\cap\, \pi$
$\tau \leftarrow$ **Scaling**(τ, T)
def ***Scaling***(τ, T):
  for $i \in [0, 1, \ldots, N]$ **do**
    $T_{\max_i} \leftarrow \max(T_i^x, T_i^y, T_i^z)$
    for $k \in [x, y, z]$ **do**
      if $T_{max_i} > T_i^k$ **then**
        $b_{s_i}^k \leftarrow$ Solve for (3) (6) with $\tau_i^k, T_{\max_i}^k$
        $\tau_i^k \leftarrow$ generate from $T_{\max_i}, b_{s_i}^k$
return τ

Algorithm 2. Initial Guess Velocities.

Input: List of Waypoints **wp**, Initial State x_s
Output: Initial Guess (τ^o)
def CalculateInitialGuess(**wp**, x_s):
  $\mathbf{v}^o \leftarrow$ *Orientation and Module*
  for $k \in [x, y, z]$ **do**
    $T^k, \tau^{ko} \leftarrow$ solv. eq. (3) with $\mathbf{wp}^k, x_s^k, \mathbf{v}^{ko}$
  $\mathbf{b}, \mathbf{v}_{\min}, \mathbf{v}_{\max} \leftarrow$ **GetSearchSpaceFrom**(τ^o)
  return $\mathbf{b}, \mathbf{v}_{\min}, \mathbf{v}_{\max}, \tau^o$

The **CalculateInitialGuess** function and the parameters $v_{\min_i}, v_{\max_i}, b_i$ are defined in the Section 3.4. Note that our planner uses a reduced list of waypoints and the current robot state to obtain the trajectory. The reduced list contains only the following three consecutive waypoints, solving the planning with a receding horizon approach and reducing computational cost without significant loss of performance, as demonstrated in ref. [10].

3.3. Unifying Trajectory Times: Scaling Approach

Let us define T_i^k as a list with the optimal times of each axis of a certain subtrajectory τ_i, thus, $T_i^k = (T_i^x, T_i^y, T_i^z)$. Hereinafter, the superindex k refers to each axis. A τ_i subtrajectory is generally different for each axis, therefore, T_i^x, T_i^y, and T_i^z will also differ. To ensure the trajectory planning visits every waypoint, T_i^k must be coincident on each axis. A scaling solution is applied,

matching times to the larger traveling time. The next procedure is explained with a subtrajectory τ_i, however, it must be applied to the whole trajectory τ. Considering the maximum time of all the axes ($T_{\max_i} = \max(T_i^k)$) of a τ_i, we slow down the faster axes' policies to impose the same duration for all axes. For this purpose, we introduce a new variable (b_{s_i}) that replaces the binary parameter b of Equation (3). The value of the variable is obtained by solving the Equation (3) augmented with Equation (7) for each axis:

$$t_{1_i}^k + t_{2_i}^k = T_{\max_i} \tag{7}$$

The solution for b_{s_i} must fall within the range $[-1, 1]$ to guarantee that acceleration inputs are slowed down.

3.4. Defining Search Space

In this section, we introduce an initialization that ensures the convergence of Algorithm 1. We assume that the optimal solution is near to an initial guess (τ^o), which is based on certain initial velocities defined for the set of waypoints ($\mathbf{v}^o$):

$$\mathbf{v^o} = [{v_1}^o, \ldots, {v_i}^o, \ldots, {v_N}^o] \tag{8}$$

Therefore, we establish a local search around $\mathbf{v}^o$ by creating a bounding box, constraining the search space. The selection of $\mathbf{v}^o$ is not random and follows certain specifications.

The system's velocity space is defined as a sphere and represents the range of velocities that a certain system can achieve within its physical limitations, defining the maximum velocity as $v^{\max}$. We propose a search space as a box centered in a velocity ${v_i}^o \in \mathbf{v}^o$. **Figure 1** illustrates a two-axis simplification of the velocity space and the search space.

The domain of the search space (π) must always be inside the velocity space of the system. Therefore, the velocity ${v_i}^o$ and the size of the boxes are defined according to the maximum velocity of the system. Leading these considerations, the bounding box $\mathcal{B}(v)_i$ of the local search is defined as follows:

$$\mathcal{B}(v)_i : = \{v \in \mathbb{R}^3 : \overset{3}{\forall} |v - {v_i}^o| \leq v_l\} \tag{9}$$

We define each ${v_i}^o$ by setting its module and its orientation. First, we impose $|{v_i}^o| = \frac{2}{3} v^{\max}$ and $v_l = \frac{\sqrt{3}}{6} v^{\max}$ to ensure that the bounding box $\mathcal{B}(v)_i$ remains inside the velocity space, and also to maximize the search space. These values were chosen to achieve a balance between both considerations. Then, we propose two approaches to establish the orientation of each ${v_i}^o$ according to two different applications. Each definition of the orientation will be further discussed in Section 4.

The bounding box constraints are defined as linear constraints and introduced to the optimization problem. To do so, we substitute Equation (6) constraints with the bounding box constraints:

$$\begin{aligned} v_{\max_i}^k &= v_i^{ko} + v_l \\ v_{\min_i}^k &= v_i^{ko} - v_l \end{aligned} \tag{10}$$

Additionally, we calculate the values of $b_i^k \in \mathbf{b}_i$ by searching the minimum time solution solving Equation (3) for each pair of waypoints. The optimal b_i^k is the value that gives the minimum $T_i^k = t_{1_i}^k + t_{2_i}^k$. The variable b_i^k (Section 3.1) is defined as a binary value for the initialization procedure.

Once each constraint $\mathcal{B}(v)_i$ and the values $\mathbf{b}_i$ of the whole trajectory τ^o are calculated, the search space π is completely defined: Algorithm 2 resumes the aforementioned initialization method, and its output is employed in line 2 from Algorithm 1 to set the local optimization.

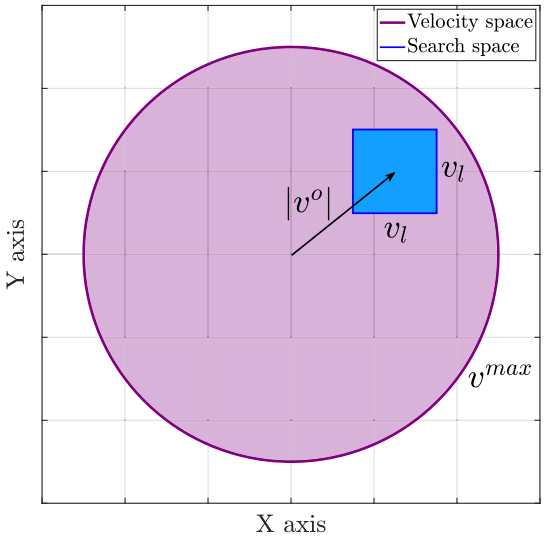

Figure 1. Velocity space is defined (purple circle) by the set of velocities that the system is able to reach, being $v^{\max}$ the module of the maximum velocity allowed. Meanwhile, the search space is defined as the region inside this domain (blue box) centered in ${v_i}^o$ (black arrow).

4. Experiments and Results

Two experiments are carried out to prove the feasibility of the method. First, we obtain the planned trajectories using the presented method. A piecewise function of seventh-order polynomials approximates the PMM trajectories to generate feasible trajectories for the quadrotor. Then, the resulting trajectories are executed with two different trajectory trackers. The first experiment is performed in a simulated aerial racing circuit. The circuit presents several waypoints or gates that must be traversed following a specific sequence. The second experiment is conducted in the physical world with a real quadrotor. This real-world experiment introduces practical challenges and unforeseen uncertainties, complementing the results obtained from the simulation.

4.1. Experiment I: Simulation

In this experiment, the trajectory planner is executed in a simulated environment using a racing quadrotor, specifically the *kingfisher* quadcopter from ETH Zurich University. The simulation is run in the Agilicious framework.[29] This simulator is highly valued due to its capacity to obtain virtually identical results when employed in real-world experiments. In addition, it implements

the latest state-of-the-art algorithms for agile flight. The simulation is performed in a laptop with an *Intel Core i7-9750H* processor and a *GeForce GTX 1650* graphics processing unit with 16 GB of RAM.

The controller used for tracking the trajectory is a nonlinear MPC (NMPC), particularly the one described in ref. [30]. This NMPC uses the full nonlinear model of the quadcopter, enabling the vehicle to be controlled with body rate and thrust commands. The implementation is performed using the *ACADOS* framework[31] with the high-performance interior-point method solver. To ensure that the body rates commanded by the NMPC are achieved, the quadrotor also uses an incremental nonlinear dynamic inversion controller in an inner loop.

The simulation consists of a racing circuit defined by seven gates that must be crossed in a specific order, and the quadrotor must pass through these gates as quickly as possible. Each gate is defined as a waypoint. We chose the orientation of each $v_i^o \in \mathbf{v}^o$ as the normal vector of the gates to avoid collisions. The other parameters are summarized in **Table 1**.

The experiment pretends to compare our solution performance with a baseline method. We consider as a baseline the results obtained from the planner proposed in ref. [9], which is also evaluated in the same circuit. This planner is a time-optimal planner that solves the time-allocation problem with CPC. To the best of our knowledge, this is the highest-quality time-optimal planner.

Table 1. Experiment I planning parameters.

Parameter	Value	Description
v_{max}	21.5 ms^{-1}	Maximum velocity
$\|v_i^o\|$	14.3 ms^{-1}	Initial velocity
v_l	6.21 ms^{-1}	Constraint velocity
u_{max}	26 ms^{-2}	Maximum acceleration

4.1.1. Trajectory Planning

The PMM trajectory is obtained following Algorithm 1 with predefined constraints following the drone acceleration per axis specifications (u_{max}) presented in Table 1.

Figure 2 shows the computed linear accelerations during a full lap (orange line), with timestamps for each gate. Then, the trajectory is segmented and approximated by a seventh-order polynomial expression (blue line). To enable the tracking of our polynomial trajectory using NMPC, we use the quadrotor dynamics planning solution introduced by Richter et al.,[20] which yields the sequential quadrotor states and controls (thrust and body rates). The thrust and body rate commands, including the limits of the quadrotor, are presented in Fig**ure 3**.

4.1.2. Comparative Analysis with Baseline Method

In **Figure 4**, we illustrate the trajectories followed by both the CPC and our approximated trajectory when applied to the racing circuit. The trajectories shown belong to a nominal lap, but the entire simulation consists of three laps. The first and last laps are used to speed up and slow down the quadrotor, respectively, so that the transition effects do not impact the timing measurements for tracking experiments (Section 4.1.3). The three-lap simulation is repeated 20 times to have meaningful statistical performance metrics.

Table 2 presents a comparative analysis of the trajectory generation methods based on the polynomial and CPC trajectories illustrated in Figure 4. Furthermore, the PMM trajectory is also included in the table for comparison purposes. The time spent to complete a full lap of the circuit referred to as lap time is employed as a primary metric to evaluate trajectory quality. Supplementary metrics, such as error gate, that measure the minimum distance between a given waypoint and the trajectory, are also included to enhance the evaluation. Note that the computational time is measured differently for the CPC method. The

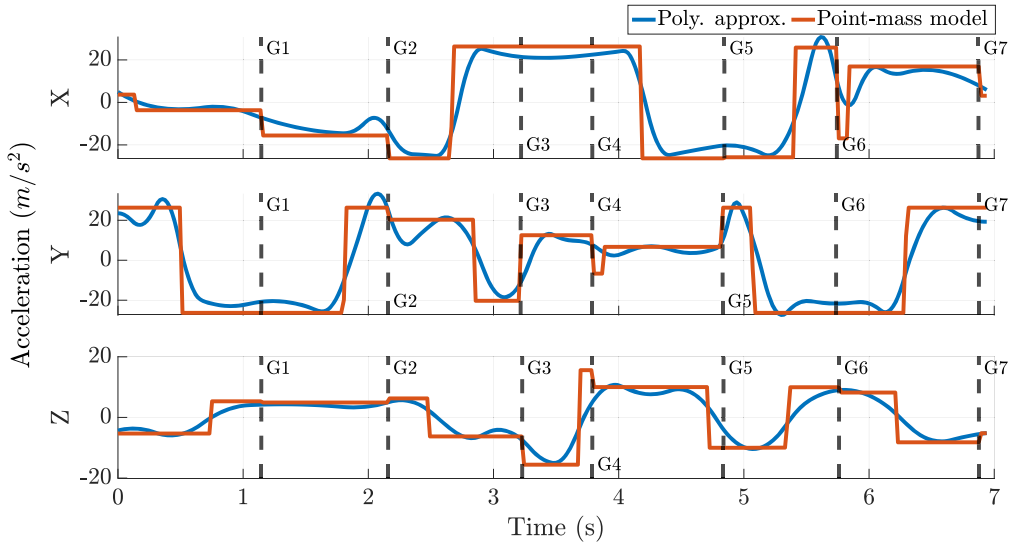

Figure 2. Linear accelerations for planned trajectories across all axes (*X*, *Y*, *Z*) illustrated with PMM solution in orange and approximated trajectory in blue for a full lap on the racing circuit.

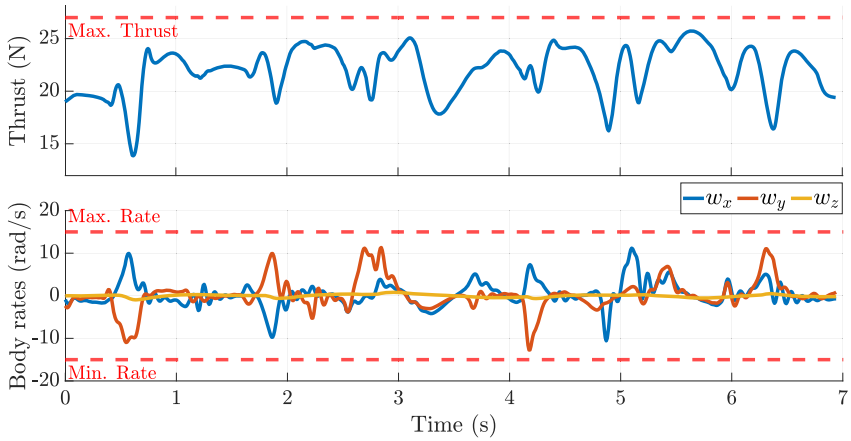

Figure 3. Thrust and rate commands derived from the approximated trajectory of the racing circuit within the confines of the quadrotor model's limitations.

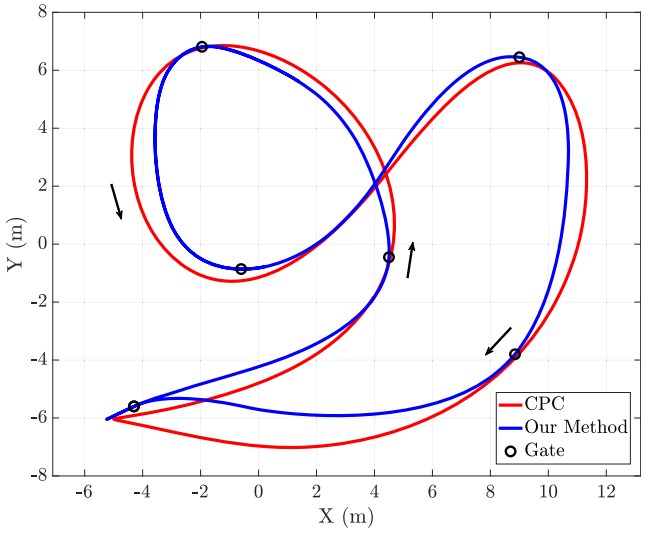

Figure 4. Visualizing trajectories in the racing circuit: our method (in blue) versus the CPC method (in red), with gate markers represented by black circles.

Table 2. Baseline comparison.

	CPC	PMM	Polynomial approximation
Computation time	30±5 min	3±0.5* ms	7±0.5* ms
Lap time [s]	6.1	6.4	6.94
Error gate [m]	Less than 0.3	0	0
Peak speed [ms^{-1}]	20	20	18.6

CPC trajectory's computational time refers to the circuit's full track. In contrast, our algorithm needs 3 ms to compute a PMM trajectory with three waypoints, followed by an additional 4 ms to obtain a feasible trajectory. It is also noteworthy that the lap time and the peak speed remained constant for the 20 experiments as the optimal solution is consistently achieved.

Figure 5 complements the trajectories shown in Figure 4 by displaying the CPC and the approximated trajectories, with a colorbar indicating the velocity profile. This visualization enhances understanding of velocity variations along the trajectories, offering valuable insights into their performance.

4.1.3. Tracking Performance

Our method's reference is depicted in **Figure 6** by a dotted line. Simultaneously, the tracked simulated position of the quadcopter is shown with a continuous line with a colorbar of its velocity. The simulated tracking exhibits an average error of 0.17 m concerning the referenced trajectory and with a maximum peak velocity of 18.3 ms^{-1}.

To measure the feasibility of the trajectory, we carried out an evaluation to determine the lap time achieved with this reference. To compute a reliable lap time, our analysis follows the statistical method provided by Foehn et al.[9] The performed lap time is 7.02 s against 6.94 s from the planned trajectory.

4.2. Experiment II: Real world

A real-world experiment is tested indoors using the *Optitrack* cameras as the motion capture system, with 5 *Optitrack PrimeX 13* and the same laptop used in Section 4.1. The real experiment consists of passing through a sequence of waypoints as soon as possible with a real quadrotor. The Crazyflie 2.1 quadcopter is used in conjunction with a laptop to transmit polynomial trajectories via the Crazyradio PA module. The high-level commander module generates a collection of states, including position, velocity, and acceleration, which a cascade PID controller should aim to achieve.

In this case, each v_i^o orientation for $n = 1 : N - 1$ is defined by the direction between i and $i + 1$ waypoints. **Table 3** summarizes the parameters used in the real-world experiment.

Our algorithm is employed in two novel circuits for the real quadcopter, instead of executing the previously simulated circuits due to the real-world limitations. In concrete, the real quadcopter has lower capabilities, particularly in terms of speed.

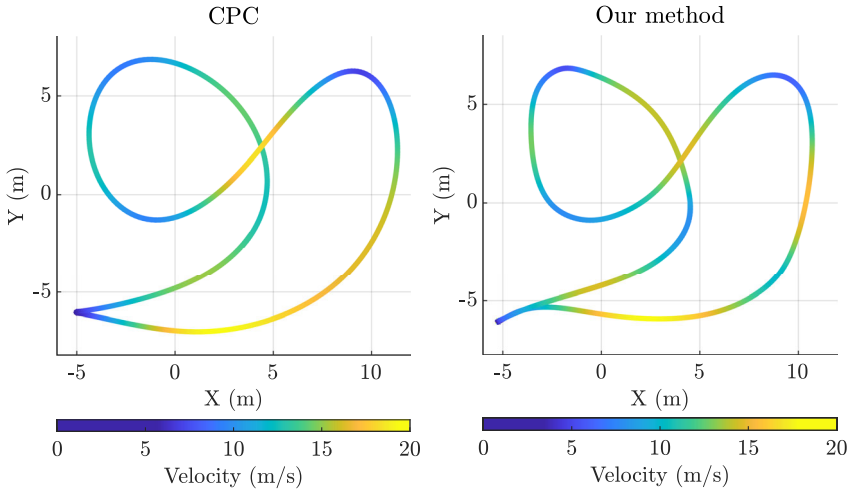

Figure 5. Velocity profiles of the CPC method (left) and our approach (right) on the racing circuit, with a colorbar indicating flight speeds.

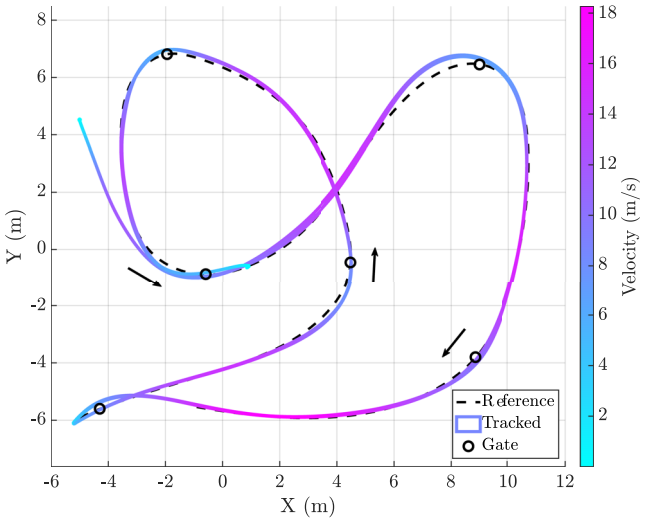

Figure 6. Comparison between the reference trajectory (dotted line) and simulated quadcopter (continuous line). In addition to, the velocity profile of the system is represented with a colorbar to contrast with the planned trajectory.

Table 3. Experiment II planning parameters.

Parameter	Value	Description
v_{max}	0.9 ms^{-1}	Maximum velocity
$\lvert v_i^o \rvert$	0.6 ms^{-1}	Initial velocity
v_l	0.26 ms^{-1}	Constraint velocity
u_{max}	2.2 ms^{-2}	Maximum acceleration

Moreover, the capture volume defined by the motion capture system is limited. Therefore, the proposed circuits are generated according to these characteristics. Concerning real-world experiments, we introduce two flight patterns as circuits: an eight-shaped and a circle. The circle is defined by eight equidistant waypoints placed at the same height with a radius of one meter. The eight-shaped circuit is defined by six waypoints distributed at three different altitudes to increase the circuit's complexity.

We send the coefficients that describe the polynomial trajectory to the high-level commander, which generates the commands to control the quadrotor (thrust and body rates).

Figure 7 displays the circuit waypoints (dots) and the resulting trajectory (dotted line) that is passed to the high-level commander, while the tracking position is depicted using a colorbar indicating the achieved speed. The lap time planned for the circle circuit is 5.8 s, whereas the Crazyflie completed it in 5.98 s. In contrast, the lap time for the polynomial approach in the eight-shaped circuit is 7.48 s, while the tracked lap time extended to 7.62 s. The mean tracking errors of the circle circuit and the eight-shaped circuit are 2.39 and 4.25 cm, respectively.

5. Discussion

The following subsections provide a detailed analysis of the proposed trajectory planning, highlighting its performance in terms of optimality, computational efficiency, and feasibility.

5.1. Optimality and Feasibility

In this subsection, the optimality and feasibility of our method's solution will be discussed based on the results presented in Section 4.1.1. As shown in Figure 2, our solution is optimal as it follows a bang-bang policy between gates, ensuring that the trajectory satisfies the conditions of Pontryagin's maximum principle. It is noteworthy that certain segments of the trajectory exhibit a lack of switching time ($t_2 = 0$), transitioning the bang-bang policy into a single-saturation action (e.g., *X*-axis acceleration from gate 1 to 2). In addition, Figure 2 also shows that the polynomial approximation follows the PMM accelerations with high accuracy. However, considering that the trajectory is feasible, there are some discrepancies. Therefore, the polynomial approximation is a near-time optimal solution. Regarding Figure 3, it can be observed how the thrust and rate commands

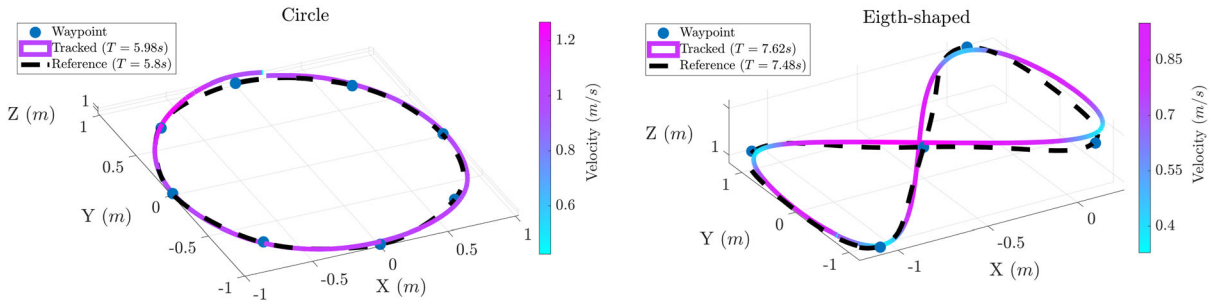

Figure 7. Waypoints represented by dots define the different circuits. Meanwhile, the planned trajectories are depicted by dotted lines. The tracked results are shown with a colorbar describing the velocity in each instant.

of the quadcopter, on several occasions, take values close to the imposed limits; however, they never exceed them.

5.2. Comparative Benchmarking

Here, we will discuss the performance of our method compared to a baseline method with the results obtained in Section 4.1.2. On one hand, it is reasonable to conclude that the PMM and the CPC solutions yield similar trajectories. This statement is supported by the results presented in Table 2, which show the same peak speed values and almost equivalent lap times. On the other hand, the approximated trajectory exhibits a reduced performance in contrast to the PMM trajectory, as expected. Still, the trajectory maintains its quality compared to the CPC solution, as presented in Figure 5. This performance difference underscores a trade-off between optimality and computational efficiency. Although our method results in slightly longer lap times, it significantly reduces the computational overhead.

Note that our solution is subject to the waypoints by definition; thus, the waypoints are part of the planned trajectory. In addition, the CPC method does not have a hard constraint concerning these waypoints, in fact, the planned trajectory can deviate from them significantly (a maximum of 0.3m[9]). It is also worth noting that our method limits the speed in the waypoints due to the search space, whereas the CPC method does not have this limitation. This restriction is presented in Figure 5, where our trajectory presents lower velocities than the CPC trajectory in the areas around waypoints. Considering these two constraints, it is evident that our method will generally present slightly higher lap times (6.94 s compared to 6.1 s of the baseline method).

5.3. Trajectory Tracking

This subsection relates the evaluation of both the simulation and the real-world experiment results. While the contributions of this work are not focused on achieving optimal control tracking, it is important to test the planned trajectories to validate our method. Figure 6 illustrates the tracking performance of the quadrotor utilizing the NMPC. Due to the tracking errors caused by the control delay included in the simulation and unmodeled drag forces, the simulated quadrotor takes ≈0.08 s longer than the planned time to complete a lap. Even with the presented drawbacks, the tracking of the simulated circuit is performed with a mean error of the order of centimeters, proving the viability of the trajectory.

The real-world experiments validate the robustness of our method, considering that the tracking is performed successfully, as shown in Figure 7. In both experiments, the Crazyflie is able to track the planned trajectories while subject to our method conditions, maintaining the velocity at the waypoints within the defined limits. The mean tracking error of both circuits is, similarly as in the simulation experiments, of the order of centimeters. Note that Crazyflie's time performance is slightly inferior to the simulated experiment (0.18 and 0.24 s for the real experiments compared to 0.08 s for the simulated experiment).

6. Conclusions and Future Works

This work presented a time-optimal trajectory planning based on optimization methods for a multi-waypoint navigation problem. The robot policy has been considered as a point mass with bang-bang accelerations, resulting in primitive trajectories that are optimized. The simulation experiments have shown that the performance of the proposed method was slightly lower in terms of lap times than the CPC; however, our approach has demonstrated a significantly faster computational time. In addition, the results of real-world experiments demonstrate the practical feasibility of our method, validating its implementation in real systems.

Although in the simulation experiments the quadrotor NMPC has followed the PMM trajectory successfully, the calculus of the references has affected the performance of the tracking, resulting in a higher lap time. Similar issues were presented in the real experiments. We will explore alternative algorithms, such as the MPCC algorithm, to enhance the time-optimal tracking performance, considering that it is one of the highest performance time-optimal trackers in the state of the art. Other interesting links to our work include autonomous navigation in cluttered environments, considering that our approach is suitable for dynamic replanning algorithms.

Acknowledgements

D.L.-Y. and F.P. contributed equally to this work. The authors would like to thank the Robotics and Perception Group, led by Prof. Davide Scaramuzza at the University of Zurich and ETH Zurich, for generously sharing detailed

racing circuit waypoints configuration. This research was funded by the University of Málaga, the Ministerio de Ciencia, Innovación y Universidades, Gobierno de España, grant no. PID2021-122944OB-I00.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

The data that support the findings of this study are openly available in Github at https://github.com/DahuiLin/DataRecordCrazyflieExperiments.git, reference number 0.

Keywords

motion plannings, optimization-based methods, time-optimal plannings

Received: May 7, 2024
Revised: October 3, 2024
Published online:

[1] A. Liniger, J. Lygeros, in *Proc. of the 18th Inter. Conf. on Hybrid Systems: Computation and Control*, Seattle, WA, April **2015**, pp. 1–10.
[2] P. Saeedi, S.-A. Sorensen, S. Hailes, in *IEEE Inter. Workshop on Safety, Security & Rescue Robotics (SSRR)*, IEEE, Piscataway, NJ, November **2009**, pp. 1–6.
[3] P. Q. Lee, V. Rajendran, K. Mombaur, *Front. Robot. AI.* **2022**, *9*, 898890.
[4] S. Gilroy, D. Lau, L. Yang, E. Izaguirre, K. Biermayer, A. Xiao, M. Sun, A. Agrawal, J. Zeng, Z. Li, K. Sreenath, in *IEEE 17th Inter. Conf. on Automation Science and Engineering (CASE)*, IEEE, Piscataway, NJ **2021**, pp. 2132–2139.
[5] L. Chen, D. Qin, X. Xu, Y. Cai, J. Xie, *Adv. Eng. Softw.* **2019**, *132*, 65.
[6] A. Mehmood, M. Liaquat, A. I. Bhatti, E. Rasool, in *5th Inter. Conf. on Control, Automation and Robotics (ICCAR)*, IEEE, Piscataway, NJ **2019**, pp. 331–335.
[7] D. Kogan, R. Murray, in *Conf. on Decision and Control (CDC)*, San Diego, CA, December **2006**.
[8] W. Van Loock, G. Pipeleers, J. Swevers, in *European Control Conf. (ECC)*, IEEE, Piscataway, NJ **2013**, pp. 1788–1792.
[9] P. Foehn, A. Romero, D. Scaramuzza, *Sci. Robot.* **2021**, *6*, 34.
[10] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, D. Scaramuzza, *Auton. Robot.* **2022**, *46*, 307.
[11] S. M. LaValle, *Planning Algorithms*, Cambridge University Press, Cambridge **2006**.
[12] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, D. Scaramuzza, *Nature* **2023**, *620*, 982.
[13] C. W. Warren, in *Proc., IEEE Inter. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **1990**, pp. 500–505.
[14] D. Fox, W. Burgard, S. Thrun, *IEEE Robot. Autom. Mag.* **1997**, *4*, 23.
[15] M. Castillo-Lopez, P. Ludivig, S. A. Sajadi-Alamdari, J. L. Sanchez-Lopez, M. A. Olivares-Mendez, H. Voos, *IEEE Robot. Autom. Lett.* **2020**, *5*, 3620.
[16] J. Xin, Y. Qu, F. Zhang, R. Negenborn, *Complex Syst. Model. Simul.* **2022**, *2*, 273.
[17] M. Sánchez, J. Morales, J. L. Martnez, *Sensors* **2023**, *23*, 3239.
[18] J. Xin, T. Xu, J. Zhu, H. Wang, J. Peng, *Adv. Intell. Syst.* **2024**, *6*, 2300703.
[19] J. Levine, *Analysis and Control of Nonlinear Systems: A Flatness-based Approach*, Springer Science & Business Media, Berlin **2009**.
[20] C. Richter, A. Bry, N. Roy, in *Robotics Research: The 16th Inter. Symp. ISRR*, Springer, Berlin **2016**, pp. 649–666.
[21] J. Tordesillas, B. T. Lopez, M. Everett, J. P. How, *IEEE Trans. Robot.* **2021**, *38*, 922.
[22] D. J. Webb, J. Van Den Berg, in *IEEE Inter. Conf. on Robotics and Automation*, IEEE, Piscataway, NJ **2013**, pp. 5054–5061.
[23] S. Liu, K. Mohta, N. Atanasov, V. Kumar, *IEEE Robot. Autom. Lett.* **2018**, *3*, 2439.
[24] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, M. Horn, in *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems (IROS)*, IEEE, Piscataway, NJ **2018**, pp. 4523–4530.
[25] M. Shomin, R. Hollis, in *IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems*, IEEE, Piscataway, NJ **2014**, pp. 3636–3641.
[26] S. Victor, J.-B. Receveur, P. Melchior, P. Lanusse, *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 4556.
[27] R. E. Kopp, *Mathematics in Science and Engineering*, Vol. *5*, Elsevier, Amsterdam **1962**, pp. 255–279.
[28] R. Penicka, D. Scaramuzza, *IEEE Robot. Autom. Lett.* **2022**, *7*, 5719.
[29] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, D. Scaramuzza, *Sci. Robot.* **2022**, *7*, eabl6259.
[30] S. Sun, A. Romero, P. Foehn, E. Kaufmann, D. Scaramuzza, *IEEE Trans. Robot.* **2022**, *38*, 3357.
[31] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, M. Diehl, *Math. Program. Comput.* **2021**, *14*, 147.