drones

MDPI

Article

Utilizing the Finite Fourier Series to Generate Quadrotor Trajectories Through Multiple Waypoints

Yevhenii Kovryzhenko * and Ehsan Taheri

Department of Aerospace Engineering, Auburn University, Auburn, AL 36849, USA; etaheri@auburn.edu
* Correspondence: yzk0058@auburn.edu

Abstract: Motion planning is critical for ensuring precise and efficient operations of unmanned aerial vehicles (UAVs). While polynomial parameterization has been the prevailing approach, its limitations in handling complex trajectory requirements have motivated the exploration of alternative methods. This paper introduces a finite Fourier series (FFS)-based trajectory parameterization for UAV motion planning, highlighting its unique capability to produce piecewise infinitely differentiable trajectories. The proposed approach addresses the challenges of fixed-time minimum-snap trajectory optimization by formulating the problem as a quadratic programming (QP) problem, with an analytical solution derived for unconstrained cases. Additionally, we compare the FFS-based parameterization with the polynomial-based minimum-snap algorithm, demonstrating comparable performance across several representative trajectories while uncovering key differences in higher-order derivatives. Experimental validation of the FFS-based parameterization using an in-house quadrotor confirms the practical applicability of the FFS-based minimum-snap trajectories. The results indicate that the proposed FFS-based parameterization offers new possibilities for motion planning, especially for scenarios requiring smooth and higher-order derivative continuity at the expense of minor increase in computational cost.

Keywords: UAV; trajectory; optimization; guidance; minimum-snap; finite-Fourier series; polynomial; quadrotor; quadratic programs; motion planning

Academic Editor: Heng Shi

Received: 11 December 2024
Revised: 15 January 2025
Accepted: 17 January 2025
Published: 20 January 2025

Citation: Kovryzhenko, Y.; Taheri, E. Utilizing the Finite Fourier Series to Generate Quadrotor Trajectories Through Multiple Waypoints. *Drones* **2025**, *9*, 77. https://doi.org/10.3390/drones9010077

1. Introduction

In the modern world of rapidly evolving and growing use of small unmanned aerial vehicles (UAVs) [1–4], path- and motion-planning algorithms serve as an integral element of any mission or product design related to the field of robotics and applications related to aerospace vehicles [5,6]. Path planning generally encompasses the functions of the overall system coordination. It establishes an interface between the machine and a human operator, who generally defines a global task for the vehicle or its purpose. The commands and interactions are then followed automatically to relay the information downstream to all the vehicle's subsystems, each executing a well-defined task. For this exact reason, path planning is generally regarded as the highest level of control that enables autonomy [7].

It is traditional to separate front-end path planning (i.e., generation of waypoints) and back-end motion planning or trajectory optimization that takes into account dynamic feasibility [8]. From a mission-designer perspective, it is convenient to think of path planning as an approach to defining a set of tasks for the vehicle to execute. For example, one can consider a map of local terrain features, such as varying elevation, structures, trees, or other factors [7]. A set of tasks can be defined (e.g., payload delivery [9–12], building inspection [13], remote sensing [14], firefighting [15], and even cell network extension [16]).

All the aforementioned tasks can be summarized by a set of requirements to traverse from an initial waypoint to a target destination.

Path planning (or front-end) focuses on finding a feasible path irrespective of vehicle dynamics and time, defined as a discrete set of waypoints between the initial and final points (or a collection of points of interest). Conversely, motion planning (or the back-end) builds upon these discrete waypoints, incorporating dynamic feasibility to perform end-to-end trajectory optimization. There are a wide variety of path-planning methods, ranging from sampling-based to search-based [17] algorithms. One popular sampling-based algorithm is Randomly-Exploring Random Tree (RRT) [18]. The fundamental principles of this algorithm are based on exploring randomly generated trees of possible intersections and junctions, seeking to connect the point of origin or initialization with the target or destination [19,20]. However, sampling-based methods suffer greatly from their random nature and cannot guarantee convergence for real-time applications and are often replaced or complemented by rapid geometry-based re-planing routines [21].

Recently, learning-based approaches [22,23], which bypass reliance on mathematical models, have shown promise for tackling real-time applications in unstructured environments. This shift in approach provides flexibility for handling unknown system dynamics or disturbances, complementing traditional model-based methods [24].

Trajectory optimization [25] for quadrotors relies heavily on the concept of differential flatness [26,27]. Mellinger and Kumar [28] established a connection between the states of an underactuated quadrotor system and four *flat outputs*. Differential flatness simplifies the trajectory optimization problem, reducing it to optimizing only translational position coordinates (x, y, z), the heading (yaw) angle ψ, and their higher-order derivatives [29]. By minimizing the fourth time derivative squared of position coordinates and ensuring that the path and its derivatives are continuous up to the fourth order, it is possible to obtain a continuous trajectory using piece-wise polynomials [30].

The resulting minimum-snap optimization problem can be solved by formulating the original cost function as a QP problem [31], where the solution yields a set of optimal polynomial coefficients that are valid along their respective intervals. The resulting QP problems can be further augmented by incorporating linear equality and inequality constraints to represent along-the-path and boundary limitations on the trajectory, and the resulting QP problems can be solved using standard QP solvers.

Among some of the common practical applications of path-planning blended with trajectory optimization is on-board trajectory generation in a cluttered environment [32–34]. For example, Hong et al. [35] have combined the A^* path-planning algorithm with a polynomial-based, minimum-snap trajectory optimization routine and demonstrated how such a fusion of algorithms can be used for real-time collision avoidance when flying through the forest. For path-planning purposes, they have used a 3D LiDAR system for generating point clouds and used an onboard companion computer (*NVIDIA TX2*) to compute the high-level reference command to the control system. An equivalent combination of a 3D LiDAR sensor and an on-board companion computer was used by Zhou et al. in [36] to compute collision-free trajectories on the fly, but now indoors, and flying a smaller quadrotor. Similar applications are proposed and demonstrated for path-planning tasks of multi-agent systems [37] and exploration of indoor environments [38].

Instead of treating the optimal set of polynomial coefficients as unknown decision variables, polynomial coefficients can be indirectly incorporated into the solution. Ritcher et al. [39] showed that minimum-snap trajectories can be obtained by reformulating the QP problem as an unconstrained QP problem. However, this time, a direct analytical solution can be obtained without the need to resort to iterative numerical solvers. They have shown that, by leveraging the differential flatness property, a guaranteed feasible

solution can be obtained analytically, as long as the involved derivatives are sufficiently bounded or enough time is available for the execution of the maneuver. Their solution strategy has shown excellent numerical stability and computational speed for long-range trajectories that consist of many segments. The differential flatness property combined with the fact that the so-called flat outputs consist of three important configuration space parameters (i.e., quadrotor position) provides a convenient and powerful tool for rapid design of collision-free, time-optimal trajectories even in dense indoor environments [40]. The appealing form of the polynomial-based parameterization [41] provides a simple interface for expressing a path of varied complexity and degrees of freedom, as well as allowing for an intuitive implementation of optimization algorithms [42]. Although several optimization methods have been developed for quadrotors, as shown by Kreciglowa et al. in [43], minimum-snap optimization leads to the most power-efficient solution for quadrotor flights (when compared with minimum-acceleration and minimum-jerk solutions). The path-planning and trajectory optimization tasks for quadrotors, which consist of many segments and with real-time considerations, is an ongoing research topic [44–48].

Recent advancements in trajectory planning and optimization have significantly improved safety, efficiency, and feasibility in dynamic and cluttered environments. Shi et al. [49] introduced a multi-waypoint trajectory planning framework utilizing uniform B-spline curves combined with bidirectional A^* search to generate safe and kinodynamically feasible trajectories. Their approach further optimizes waypoint sequences through fast marching and ant colony optimization techniques, making it highly effective for navigation through complex environments. While this framework emphasizes safety and waypoint optimization, it lacks a focus on trajectory smoothness and energy efficiency, which are central to the minimum-snap formulations in this work.

Penicka et al. [50] proposed a hierarchical, sampling-based approach for planning minimum-time trajectories, particularly for drone racing and search-and-rescue applications. By combining topological pathfinding with kinodynamic planning, their method excels in generating collision-free trajectories optimized for high-speed flight. However, their focus on minimum-time trajectories diverges from this work's emphasis on minimizing higher-order derivatives, which directly impacts control effort and power consumption. Furthermore, their approach targets specific high-speed use cases, while the methods presented here are broadly applicable to diverse scenarios requiring smooth and energy-efficient quadrotor maneuvers.

Foehn et al. [51] established a benchmark in time-optimal quadrotor flight, demonstrating trajectories that outperform professional human pilots. While their work achieves exceptional speed performance, the trajectory smoothness and energy efficiency considerations—critical for prolonged autonomous operations—are not the primary focus. By contrast, the minimum-snap optimization method presented here prioritizes smooth reference trajectories, making it more suitable for tasks where control effort and energy conservation are essential.

Liu et al. [52] introduced a three-body cooperative active defense guidance law, leveraging Zero-Effort-Miss and Zero-Effort-Velocity notions to minimize errors in time estimates for interception scenarios. While their work is highly innovative in addressing cooperative guidance and energy efficiency within the defense context, its applicability to quadrotor trajectory planning is limited, as it focuses on dynamic interactions among multiple agents rather than optimizing single-agent trajectories through waypoints. This study, in contrast, addresses the challenges of waypoint-based trajectory generation by combining minimum-snap optimization with time-allocation techniques, offering a practical balance between computational efficiency and trajectory dynamic feasibility.

This paper is inspired by some previous research, with the original work demonstrating the application of the FFS method for generating continuous-thrust spacecraft

trajectories by Taheri et al. [53,54]. The FFS method was further improved in [55] by reformulating the FFS relations in a compact matrix representation, making the FFS method suitable for programming purposes. In Ref. [56], it is shown that control (maximum thrust magnitude) path constraints can also be handled using the FFS method. Benefiting from its high computational efficiency and flexibility in incorporating various constraints, the FFS method was adopted for shaping spacecraft trajectories with various coordinate representations and propulsion systems [57–60]. The most recent applications of the FFS method are for solving spacecraft multiple gravity-assist low-thrust trajectory optimization problems [61] and feasibility demonstration for quadrotor motion planning [62]. The preliminary version of this work has been published as a master's thesis [63] and also used for online trajectory re-planning demonstrated in [64]. The method is also suitable for trajectory tracking using a data-driven $\mathcal{H}_\infty$ controller [65].

In this work, we discuss the advantages and disadvantages of selecting FFS-based parameterization for minimum-snap motion planning of quadrotors. The inherent properties of the FFS method, such as piecewise infinite differentiability, offer significant potential for addressing scenarios requiring higher-order derivative constraints. For instance, some classes of flight vehicles, including conventional fixed-wing aircraft, demand the satisfaction of derivatives beyond snap. The fifth derivative of position, for example, is critical for accurately expressing all the states and control outputs of such systems, as outlined in Chp. 14.0.4.1 of [66]. The flexibility and smoothness inherent in FFS-based parameterization make it a strong candidate for these complex motion-planning tasks, particularly when smooth transitions across trajectory segments are essential.

One of the primary objectives of this study is to provide a detailed comparison of the FFS- and polynomial-based parameterizations for motion planning of quadrotors. For fixed-time, minimum-snap, motion-planning problems, we demonstrate that the FFS-based parameterization enables the formulation of trajectory optimization problems as QP problems. Furthermore, we derive an analytic solution for the unconstrained QP problem, showcasing the computational advantages of the FFS approach in scenarios where efficiency and accuracy are paramount.

A significant contribution of this work lies in the exploration of time allocation strategies within the context of FFS-based parameterization. We leverage the analytic solution to formulate and solve time-allocated, minimum-snap, multi-segment trajectories, introducing a unique property of the method: the ability to adjust total flight time as a post-processing step without recomputing the trajectory. This capability ensures computational efficiency while preserving optimality, a feature that is not commonly addressed in the existing literature on motion planning.

In addition to theoretical contributions, we present a fair one-to-one comparison of FFS- and polynomial-based formulations across five classes of trajectories. This includes computational results and experimental validation using a custom-built quadrotor, ensuring the practical relevance of our findings. To facilitate reproducibility, detailed position and heading angle boundary conditions, as well as time of flight data for the fixed-time cases, are provided in Appendix A. By addressing both theoretical and practical aspects, this work aims to bridge the gap between analytical trajectory optimization methods and real-world applications.

This paper is organized as follows. In Section 2, the trajectory generation algorithm is broken down into two parts. First, a solution to an unconstrained QP problem with a fixed-time allocation is derived in Section 2.1. Next, the problem formulation is augmented with a time-allocation algorithm derived in Section 2.2. Section 2 is concluded by providing some details about testing the two algorithms, hardware implementation, and details regarding the implementation process in Section 2.3. Computational results for a set of fixed-time

problems (Section 3.1) and time-allocation problems (Section 3.2), that benchmark the two parameterization methods against each other, are presented in Section 3. Finally, the results of the experimental validation are presented in Section 4. Section 5 concludes the paper by summarizing the most important results and key findings.

2. Methods

2.1. Formulation of Fixed-Time, Minimum-Snap Trajectory Optimization Problems

By leveraging the concept of differential flatness for quadrotors [28], it is possible to perform motion planning along each axis (i.e., x, y, and z) of the motion independently. Additionally, the heading angle, ψ, can be considered as a separate fourth dimension. Let $P(t, \boldsymbol{p})$ represent an expression for each of the flat outputs, such that it is differentiable with respect to time up to n_{dt} times, where $\boldsymbol{p}$ is a vector of design variables. The general approach is to minimize the square of the n_{dt}-th derivative of $P(t, \boldsymbol{p})$ over the entire interval and for each of the axes of motion. While motion planning of robotic joints, limbs, and manipulators [67,68] require minimization of the third derivative squared (or minimum-jerk optimization with $n_{\mathrm{dt}} = 3$), motion planning for multirotor must consider minimization of the fourth derivative of the position squared. The Lagrange form cost functional for these classes of problems (for one segment) can be written as follows:

$$\underset{\boldsymbol{p}}{\text{minimize}} \qquad J = \int_{t_{\mathrm{i}}}^{t_{\mathrm{f}}} \left[\frac{d^{n_{\mathrm{dt}}}}{dt^{n_{\mathrm{dt}}}} P(t, \boldsymbol{p}) \right]^2 dt, \tag{1}$$

in which $t \in [t_{\mathrm{i}}, t_{\mathrm{f}}]$. The same cost functional is used along each dimension (i.e., x, y, z, and heading angle) and the total cost required to be minimized is the summation of the costs for each axis of motion. The problem defined in Equation (1) is a standard variational problem and the calculus of variations techniques can be used to find an extremal solution.

A more general approach would be considering linear equality and inequality constraints, which can then be reformulated as a constrained QP problem. The authors have explored such a solution strategy in [62]. While this method offers flexibility such as directly incorporating corridor and path constraints, the main limitation is the computational cost and number of iterations required to achieve a fixed-time solution. Instead, we consider only linear equality constraints to simplify the process and obtain a fixed-time solution analytically. In this work, we consider the quadrotor motion-planning problem. First introduced in [28], a quadrotor is a differently flat system. In other words, the quadrotor states (position, velocity, acceleration, attitude, and body frame angular rates) can all be expressed in terms of a set of flat output variables (i.e., spatial position and heading angle) and their derivatives. More importantly, the control output (that is, the required thrust and torque values) is a function of the flat output variables and their derivatives. A detailed derivation has been presented in [27], but the key takeaway is that the motor speed required to execute a certain trajectory is a direct function of position, heading angle, and their higher-order time derivatives (i.e., fourth derivative of position and second derivative of the heading angle). In Ref. [30], it is also shown that the minimum-jerk quadrotor cost functional has an interpretation as an upper bound of the product of the inputs (i.e., thrust produced by the propellers and the quadrotor angular rates). Therefore, to minimize power consumption on the i-th segment, we select $n_{\mathrm{dt}} = 4$ in Equation (1), and write it as,

$$\begin{aligned} \underset{\boldsymbol{p}_{\mathrm{i}}}{\text{minimize}} \qquad & J = \int_{t_0=0}^{t_{\mathrm{f}}=T_{\mathrm{i}}} \left[\frac{d^4}{dt^4} P(t, \boldsymbol{p}_{\mathrm{i}}) \right]^2 dt = \boldsymbol{p}_{\mathrm{i}}^{\top} \mathbb{Q}_{\mathrm{i}} \boldsymbol{p}_{\mathrm{i}}, \\ \text{s.t.,} \qquad & \mathbb{A}_{\mathrm{i}} \boldsymbol{p}_{\mathrm{i}} = \boldsymbol{d}_{\mathrm{i}}, \text{for } \mathrm{i} \in \{1, \ldots, n_{\mathrm{int}}\}, \end{aligned} \tag{2}$$

where the matrix $\mathbb{A}_\mathrm{i}$ maps the vector of design variables, $\boldsymbol{p}_\mathrm{i}$, to the constraint vector, $\boldsymbol{d}_\mathrm{i}$. The matrix $\mathbb{Q}_\mathrm{i}$ maps the vector of design variables $\boldsymbol{p}_\mathrm{i}$ to the integral of $P(t, T_\mathrm{i}, \boldsymbol{p}_\mathrm{i})$, and T_i is the time allocated for the i-th segment (in a multisegment trajectory) such that $0 \leq t \leq T_\mathrm{i}$. Note that T_i is assumed to be fixed and known. If more than one segment is used, external time is allocated for each interval with a total of n_int (number of intervals) such that a time allocation vector can be formed as $\boldsymbol{T} = [T_1, \ldots, T_{\mathrm{n_{int}}}]^\top$. Figure 1 shows a schematic summary of the definitions used throughout the paper. Each segment is represented by its own set of design variables, independent of other segments. In other words, everything, including the constraints, is applied to each axis of motion independently (there are no norm constraints). When multiple segments are present, the individual cost function and constraints for each segment, $\mathrm{i} \in \{1, \cdots, n_\mathrm{int}\}$ are used and added to express the relations in a block-diagonal manner as follows:

$$\begin{aligned} \underset{\boldsymbol{p}}{\text{minimize}} \quad & J = \begin{bmatrix} \boldsymbol{p}_1 \\ \vdots \\ \boldsymbol{p}_{n_\mathrm{int}} \end{bmatrix}^\top \begin{bmatrix} \mathbb{Q}_1 & [\mathbf{0}] & [\mathbf{0}] \\ [\mathbf{0}] & \ddots & [\mathbf{0}] \\ [\mathbf{0}] & [\mathbf{0}] & \mathbb{Q}_{n_\mathrm{int}} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ \vdots \\ \boldsymbol{p}_{n_\mathrm{int}} \end{bmatrix}, \\ \text{s.t.,} \quad & \begin{bmatrix} \mathbb{A}_1 & [\mathbf{0}] & [\mathbf{0}] \\ [\mathbf{0}] & \ddots & [\mathbf{0}] \\ [\mathbf{0}] & [\mathbf{0}] & \mathbb{A}_{n_\mathrm{int}} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_1 \\ \vdots \\ \boldsymbol{p}_{n_\mathrm{int}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_1 \\ \vdots \\ \boldsymbol{d}_{n_\mathrm{int}} \end{bmatrix}, \end{aligned} \tag{3}$$

where $\boldsymbol{p}^\top = [\boldsymbol{p}_1^\top, \cdots, \boldsymbol{p}_{n_\mathrm{int}}^\top]$ and $\mathbb{Q}_i$ and $\mathbb{A}_i$ correspond to the i-th segment and are specific to the parameterization of $P(t, \boldsymbol{p}_i)$. The constraint vector, $\boldsymbol{d}^\top = [\boldsymbol{d}_1^\top, \cdots, \boldsymbol{d}_{n_\mathrm{int}}^\top]$, consists of fixed and free boundary conditions, some of which are assumed to be specified (e.g., position at each waypoint, initial and final velocity, acceleration, etc.), but the rest are free (e.g., velocity, acceleration, and higher-order derivatives at each intermediate waypoint) and will have to be solved for. The solution steps are outlined in Section 2.1.3, but first, we have to present trajectory parameterization methods. First, we review the polynomial parameterization and compare the relations to the FFS-based parameterization.

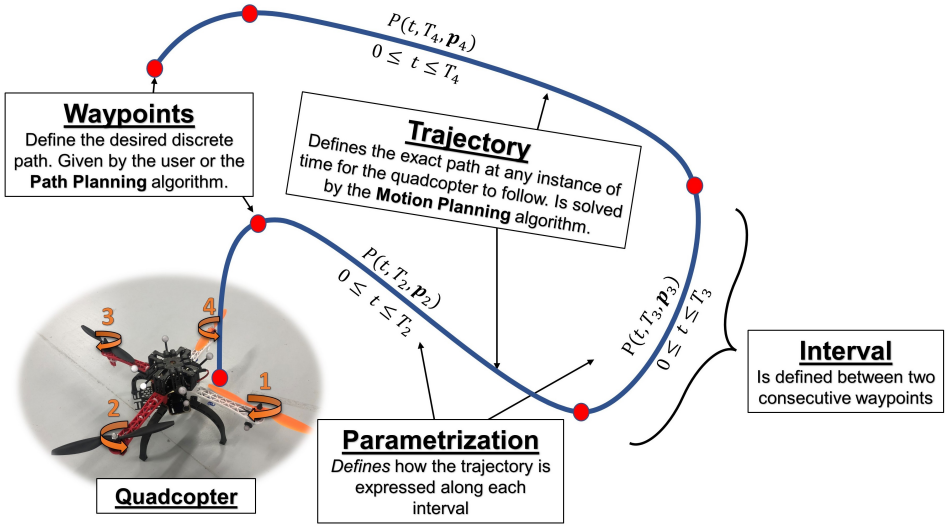


Figure 1. Definition of a representative multi-segment/interval trajectory. "interval" and "segment" are used interchangeably.

2.1.1. Generalization of the Polynomial and FFS Parameterizations

The number of boundary constraints defines the minimum number of parameters for either method. For a single-interval, single-axis, minimum-snap trajectory, there are always

ten constraints to be satisfied, including: initial and final position, velocity, acceleration, jerk, and snap values. Therefore, there must be, at least, ten parameters in the general form of the $P(t, \boldsymbol{p}_\mathrm{i})$ (i.e., $\boldsymbol{p}_\mathrm{i} \in \mathbb{R}^{n \geq 10}$). Equality constraints on derivatives allow for enforcing continuity up to snap when multiple segments are considered. However, an additional pair of equality constraints on the fifth derivative can be considered to ensure that the snap is both continuous and smooth. This would set the minimum number of parameters to twelve per axis of motion per interval. However, in this work, we are primarily focusing on standard minimum-snap optimization with constraints applied only up to snap; therefore, only ten parameters are required for each interval on a single dimension. In other words, it is sufficient [28] to ensure trajectory is smooth up to snap of position, but snap itself does not need to be smooth, only continuous (the fifth derivative can be and usually is discontinuous, since no continuity constraints are enforced above snap). If smoothness is enforced on the level of snap, the trajectories may be overly conservative, since it would be equivalent to enforcing continuity of the rate of change of motor speed [27]; therefore, it is not considered for quadrotor trajectory optimization.

The procedure to calculate the required mapping matrices is identical for both the polynomial and FFS parameterizations. The constraint matrix $\mathbb{A}_\mathrm{i}$ is simply the Jacobian of the path $P(t, \boldsymbol{p}_\mathrm{i})$ and its derivatives with regard to the design vector, $\boldsymbol{p}_\mathrm{i}$, which is then evaluated at the initial time $t_0 = 0$ and final time $t_\mathrm{f} = T_\mathrm{i}$. The quadratic mapping matrix, $\mathbb{Q}_\mathrm{i}$, is one-half of the Hessian of the cost function (Equation (2)). Throughout this work, we can assume $\boldsymbol{p}_\mathrm{i} \in \mathbb{R}^{10\times 1}$, $\mathbb{A}_\mathrm{i} \in \mathbb{R}^{10\times 10}$ and $\mathbb{Q}_\mathrm{i} \in \mathbb{R}^{10\times 10}$ for both parameterizations. These dimensions correspond to the i-th segment of a multi-segment trajectory.

Polynomial Parameterization

The most common parameterization for solving these classes of QP problems is a minimum-order polynomial function, which can be written as follows:

$$\begin{aligned} P(t, T_\mathrm{i}, \boldsymbol{p}_\mathrm{i}) &= a_0 + a_1 \frac{t}{T_\mathrm{i}} + a_2 \left(\frac{t}{T_\mathrm{i}}\right)^2 + a_3 \left(\frac{t}{T_\mathrm{i}}\right)^3 + \cdots + a_n \left(\frac{t}{T_\mathrm{i}}\right)^n, \\ \text{where} \qquad \boldsymbol{p}_\mathrm{i} &= [a_0, a_1, a_2, a_3, \ldots, a_n]^\top, \end{aligned} \tag{4}$$

where time t is scaled by the segment time T_i allocated for the i-th segment and polynomial order is $n = 9$ for constraint derivative order of four ($n_\mathrm{dt} = 4$). Mapping matrices for this formulation are derived in the following manner. The time-dependent constraint matrix, $\mathbb{A}_\mathrm{i}(t, T_\mathrm{i})$, can be derived by taking the gradient (with regard to $\boldsymbol{p}_\mathrm{i}$) of the vector consisting of polynomial representation of the path and its derivatives (ninth-order polynomial, fourth-order derivative) as follows:

$$\mathbb{A}_\mathrm{i}(t, T_\mathrm{i}) = \nabla_{\boldsymbol{p}_\mathrm{i}} \begin{bmatrix} P(t, T_\mathrm{i}, \boldsymbol{p}_\mathrm{i}) \\ \frac{d}{dt} P(t, T_\mathrm{i}, \boldsymbol{p}_\mathrm{i}) \\ \vdots \\ \frac{d^{n_\mathrm{dt}}}{dt^{n_\mathrm{dt}}} P(t, T_\mathrm{i}, \boldsymbol{p}_\mathrm{i}) \end{bmatrix} =$$

$$\begin{bmatrix} 1 & \frac{t}{T_\mathrm{i}} & \frac{t^2}{T_\mathrm{i}^2} & \frac{t^3}{T_\mathrm{i}^3} & \frac{t^4}{T_\mathrm{i}^4} & \frac{t^5}{T_\mathrm{i}^5} & \frac{t^6}{T_\mathrm{i}^6} & \frac{t^7}{T_\mathrm{i}^7} & \frac{t^8}{T_\mathrm{i}^8} & \frac{t^9}{T_\mathrm{i}^9} \\ 0 & \frac{1}{T_\mathrm{i}} & \frac{2\,t}{T_\mathrm{i}^2} & \frac{3\,t^2}{T_\mathrm{i}^3} & \frac{4\,t^3}{T_\mathrm{i}^4} & \frac{5\,t^4}{T_\mathrm{i}^5} & \frac{6\,t^5}{T_\mathrm{i}^6} & \frac{7\,t^6}{T_\mathrm{i}^7} & \frac{8\,t^7}{T_\mathrm{i}^8} & \frac{9\,t^8}{T_\mathrm{i}^9} \\ 0 & 0 & \frac{2}{T_\mathrm{i}^2} & \frac{6\,t}{T_\mathrm{i}^3} & \frac{12\,t^2}{T_\mathrm{i}^4} & \frac{20\,t^3}{T_\mathrm{i}^5} & \frac{30\,t^4}{T_\mathrm{i}^6} & \frac{42\,t^5}{T_\mathrm{i}^7} & \frac{56\,t^6}{T_\mathrm{i}^8} & \frac{72\,t^7}{T_\mathrm{i}^9} \\ 0 & 0 & 0 & \frac{6}{T_\mathrm{i}^3} & \frac{24\,t}{T_\mathrm{i}^4} & \frac{60\,t^2}{T_\mathrm{i}^5} & \frac{120\,t^3}{T_\mathrm{i}^6} & \frac{210\,t^4}{T_\mathrm{i}^7} & \frac{336\,t^5}{T_\mathrm{i}^8} & \frac{504\,t^6}{T_\mathrm{i}^9} \\ 0 & 0 & 0 & 0 & \frac{24}{T_\mathrm{i}^4} & \frac{120\,t}{T_\mathrm{i}^5} & \frac{360\,t^2}{T_\mathrm{i}^6} & \frac{840\,t^3}{T_\mathrm{i}^7} & \frac{1680\,t^4}{T_\mathrm{i}^8} & \frac{3024\,t^5}{T_\mathrm{i}^9} \end{bmatrix}. \tag{5}$$

Since we are only considering the boundary conditions, we can directly evaluate Equation (5) at the boundaries and calculate the constraint mapping matrix $\mathbb{A}_i(T_\mathrm{i})$ that only depends on the total time (which is fixed) as follows:

$$\mathbb{A}_\mathrm{i}(T_\mathrm{i}) = \begin{bmatrix} \mathbb{A}_\mathrm{i}(t=0,T_\mathrm{i}) \\ \mathbb{A}_\mathrm{i}(t=T_\mathrm{i},T_\mathrm{i}) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{T_\mathrm{i}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{2}{T_\mathrm{i}^2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{6}{T_\mathrm{i}^3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{24}{T_\mathrm{i}^4} & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & \frac{1}{T_\mathrm{i}} & \frac{2}{T_\mathrm{i}} & \frac{3}{T_\mathrm{i}} & \frac{4}{T_\mathrm{i}} & \frac{5}{T_\mathrm{i}} & \frac{6}{T_\mathrm{i}} & \frac{7}{T_\mathrm{i}} & \frac{8}{T_\mathrm{i}} & \frac{9}{T_\mathrm{i}} \\ 0 & 0 & \frac{2}{T_\mathrm{i}^2} & \frac{6}{T_\mathrm{i}^2} & \frac{12}{T_\mathrm{i}^2} & \frac{20}{T_\mathrm{i}^2} & \frac{30}{T_\mathrm{i}^2} & \frac{42}{T_\mathrm{i}^2} & \frac{56}{T_\mathrm{i}^2} & \frac{72}{T_\mathrm{i}^2} \\ 0 & 0 & 0 & \frac{6}{T_\mathrm{i}^3} & \frac{24}{T_\mathrm{i}^3} & \frac{60}{T_\mathrm{i}^3} & \frac{120}{T_\mathrm{i}^3} & \frac{210}{T_\mathrm{i}^3} & \frac{336}{T_\mathrm{i}^3} & \frac{504}{T_\mathrm{i}^3} \\ 0 & 0 & 0 & 0 & \frac{24}{T_\mathrm{i}^4} & \frac{120}{T_\mathrm{i}^4} & \frac{360}{T_\mathrm{i}^4} & \frac{840}{T_\mathrm{i}^4} & \frac{1680}{T_\mathrm{i}^4} & \frac{3024}{T_\mathrm{i}^4} \end{bmatrix}. \tag{6}$$

The quadratic mapping matrix $\mathbb{Q}_\mathrm{i}(T_\mathrm{i})$ is computed by taking the Hessian of the cost function given in Equation (2) with respect to the design vector, $\boldsymbol{p}_\mathrm{i}$, and by applying a factor of $\frac{1}{2}$, which can be written as follows:

$$\mathbb{Q}_\mathrm{i}(T_\mathrm{i}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{576}{T_\mathrm{i}^7} & \frac{1440}{T_\mathrm{i}^7} & \frac{2880}{T_\mathrm{i}^7} & \frac{5040}{T_\mathrm{i}^7} & \frac{8064}{T_\mathrm{i}^7} & \frac{12096}{T_\mathrm{i}^7} \\ 0 & 0 & 0 & 0 & \frac{1440}{T_\mathrm{i}^7} & \frac{4800}{T_\mathrm{i}^7} & \frac{10800}{T_\mathrm{i}^7} & \frac{20160}{T_\mathrm{i}^7} & \frac{33600}{T_\mathrm{i}^7} & \frac{51840}{T_\mathrm{i}^7} \\ 0 & 0 & 0 & 0 & \frac{2880}{T_\mathrm{i}^7} & \frac{10800}{T_\mathrm{i}^7} & \frac{25920}{T_\mathrm{i}^7} & \frac{50400}{T_\mathrm{i}^7} & \frac{86400}{T_\mathrm{i}^7} & \frac{136080}{T_\mathrm{i}^7} \\ 0 & 0 & 0 & 0 & \frac{5040}{T_\mathrm{i}^7} & \frac{20160}{T_\mathrm{i}^7} & \frac{50400}{T_\mathrm{i}^7} & \frac{100800}{T_\mathrm{i}^7} & \frac{176400}{T_\mathrm{i}^7} & \frac{282240}{T_\mathrm{i}^7} \\ 0 & 0 & 0 & 0 & \frac{8064}{T_\mathrm{i}^7} & \frac{33600}{T_\mathrm{i}^7} & \frac{86400}{T_\mathrm{i}^7} & \frac{176400}{T_\mathrm{i}^7} & \frac{313600}{T_\mathrm{i}^7} & \frac{508032}{T_\mathrm{i}^7} \\ 0 & 0 & 0 & 0 & \frac{12096}{T_\mathrm{i}^7} & \frac{51840}{T_\mathrm{i}^7} & \frac{136080}{T_\mathrm{i}^7} & \frac{282240}{T_\mathrm{i}^7} & \frac{508032}{T_\mathrm{i}^7} & \frac{9144576}{11\,T_\mathrm{i}^7} \end{bmatrix}. \tag{7}$$

Both the linear constraint mapping matrix, $\mathbb{A}_\mathrm{i}(T_\mathrm{i})$, and quadratic mapping matrix, $\mathbb{Q}_\mathrm{i}(T_\mathrm{i})$, are valid for cases where only one interval and one axis of motion are considered. For more complex trajectories, corresponding $\mathbb{A}(\boldsymbol{T})$ and $\mathbb{Q}(\boldsymbol{p})$ matrices can be obtained as given in Equation (3). This standard parameterization method is used as a basis of comparison against the FFS-based parameterization.

FFS Parameterization

Similarly, we can use FFS to parameterize each dimension of motion as follows:

if n_dt is odd:

$$\begin{aligned} P(t,T_\mathrm{i},\boldsymbol{p}_\mathrm{i}) = \frac{a_0}{2} + \sum_{k=1}^{n_\mathrm{r}} \Bigg\{ & a_\mathrm{k} \cos\left(\frac{1}{2}k\pi\frac{t}{T_\mathrm{i}}\right) \\ & + b_{\mathrm{k}-1} \sin\left(\frac{1}{2}k\pi\frac{t}{T_\mathrm{i}}\right) \Bigg\} + b_{\mathrm{n_r}} \cos\left(\frac{1}{2}n_\mathrm{r}\pi\frac{t}{T_\mathrm{i}}\right), \end{aligned} \tag{8}$$

if n_{dt} is even:

$$P(t, T_{\mathrm{i}}, \boldsymbol{p}_{\mathrm{i}}) = \frac{a_0}{2} + \sum_{k=1}^{n_{\mathrm{r}}} \left\{ a_{\mathrm{k}} \cos\left(\frac{1}{2} k\pi \frac{t}{T_{\mathrm{i}}}\right) + b_{\mathrm{k}-1} \sin\left(\frac{1}{2} k\pi \frac{t}{T_{\mathrm{i}}}\right) \right\} + b_{\mathrm{n_r}} \sin\left(\frac{1}{2} n_{\mathrm{r}} \pi \frac{t}{T_{\mathrm{i}}}\right), \tag{9}$$

where the design vector is $\boldsymbol{p}_{\mathrm{i}} = [a_0, \ldots, a_{\mathrm{n_r}}, b_0, \ldots, b_{\mathrm{n_r}}]$ with an abuse of notation. Since we have chosen to enforce constraints up to the fourth derivative ($n_{\mathrm{dt}} = 4$ and $n_{\mathrm{r}} = 4$), we only need Equation (9). However, Equation (8) can be used for the cases when order of constants and/or cost function is odd (from example, for minimum-jerk or minimum-crackle optimization). The two mapping matrices are derived in the same manner as for the polynomial parameterization. The constraint mapping matrix $\mathbb{A}_{\mathrm{i}}(t, T_{\mathrm{i}})$ is found by taking the gradient of the vector of derivatives with respect to the vector of design variables $\boldsymbol{p}_i$. A constant mapping matrix $\mathbb{A}_{\mathrm{i}}(T_{\mathrm{i}})$ is found by evaluating $\mathbb{A}_{\mathrm{i}}(t, T_{\mathrm{i}})$ at the boundaries and staking the results vertically. The constant constraint mapping matrix corresponding to Equation (9) is

$$A_{\mathrm{i}}(T_{\mathrm{i}}) = \begin{bmatrix} \mathbb{A}_{\mathrm{i}}(t=0, T_{\mathrm{i}}) \\ \mathbb{A}_{\mathrm{i}}(t=T_{\mathrm{i}}, T_{\mathrm{i}}) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\pi}{2\,T_{\mathrm{i}}} & \frac{\pi}{T_{\mathrm{i}}} & \frac{3\,\pi}{2\,T_{\mathrm{i}}} & \frac{2\,\pi}{T_{\mathrm{i}}} & \frac{5\,\pi}{2\,T_{\mathrm{i}}} \\ 0 & -\frac{\pi^2}{4\,T_{\mathrm{i}}^2} & -\frac{\pi^2}{T_{\mathrm{i}}^2} & -\frac{9\,\pi^2}{4\,T_{\mathrm{i}}^2} & -\frac{4\,\pi^2}{T_{\mathrm{i}}^2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{\pi^3}{8\,T_{\mathrm{i}}^3} & -\frac{\pi^3}{T_{\mathrm{i}}^3} & -\frac{27\,\pi^3}{8\,T_{\mathrm{i}}^3} & -\frac{8\,\pi^3}{T_{\mathrm{i}}^3} & -\frac{125\,\pi^3}{8\,T_{\mathrm{i}}^3} \\ 0 & \frac{\pi^4}{16\,T_{\mathrm{i}}^4} & \frac{\pi^4}{T_{\mathrm{i}}^4} & \frac{81\,\pi^4}{16\,T_{\mathrm{i}}^4} & \frac{16\,\pi^4}{T_{\mathrm{i}}^4} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & -1 & 0 & 1 & 1 & 0 & -1 & 0 & 1 \\ 0 & -\frac{\pi}{2\,T_{\mathrm{i}}} & 0 & \frac{3\,\pi}{2\,T_{\mathrm{i}}} & 0 & 0 & -\frac{\pi}{T_{\mathrm{i}}} & 0 & \frac{2\,\pi}{T_{\mathrm{i}}} & 0 \\ 0 & 0 & \frac{\pi^2}{T_{\mathrm{i}}^2} & 0 & -\frac{4\,\pi^2}{T_{\mathrm{i}}^2} & -\frac{\pi^2}{4\,T_{\mathrm{i}}^2} & 0 & \frac{9\,\pi^2}{4\,T_{\mathrm{i}}^2} & 0 & -\frac{25\,\pi^2}{4\,T_{\mathrm{i}}^2} \\ 0 & \frac{\pi^3}{8\,T_{\mathrm{i}}^3} & 0 & -\frac{27\,\pi^3}{8\,T_{\mathrm{i}}^3} & 0 & 0 & \frac{\pi^3}{T_{\mathrm{i}}^3} & 0 & -\frac{8\,\pi^3}{T_{\mathrm{i}}^3} & 0 \\ 0 & 0 & -\frac{\pi^4}{T_{\mathrm{i}}^4} & 0 & \frac{16\,\pi^4}{T_{\mathrm{i}}^4} & \frac{\pi^4}{16\,T_{\mathrm{i}}^4} & 0 & -\frac{81\,\pi^4}{16\,T_{\mathrm{i}}^4} & 0 & \frac{625\,\pi^4}{16\,T_{\mathrm{i}}^4} \end{bmatrix}. \tag{10}$$

The quadratic mapping matrix, $\mathbb{Q}_{\mathrm{i}}$, is found using the exact same process as the one for the polynomial representation, with the final expression written as follows:

$$\mathbb{Q}_{\mathrm{i}}(T_{\mathrm{i}}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{\pi^8}{512\,T_{\mathrm{i}}^7} & \frac{\pi^7}{24\,T_{\mathrm{i}}^7} & 0 & -\frac{2\,\pi^7}{15\,T_{\mathrm{i}}^7} & \frac{\pi^7}{256\,T_{\mathrm{i}}^7} & \frac{\pi^7}{12\,T_{\mathrm{i}}^7} & \frac{81\,\pi^7}{256\,T_{\mathrm{i}}^7} & \frac{8\,\pi^7}{15\,T_{\mathrm{i}}^7} & \frac{625\,\pi^7}{768\,T_{\mathrm{i}}^7} \\ 0 & \frac{\pi^7}{24\,T_{\mathrm{i}}^7} & \frac{\pi^8}{2\,T_{\mathrm{i}}^7} & \frac{243\,\pi^7}{40\,T_{\mathrm{i}}^7} & 0 & -\frac{\pi^7}{24\,T_{\mathrm{i}}^7} & 0 & \frac{243\,\pi^7}{40\,T_{\mathrm{i}}^7} & \frac{64\,\pi^7}{3\,T_{\mathrm{i}}^7} & \frac{3125\,\pi^7}{168\,T_{\mathrm{i}}^7} \\ 0 & 0 & \frac{243\,\pi^7}{40\,T_{\mathrm{i}}^7} & \frac{6561\,\pi^8}{512\,T_{\mathrm{i}}^7} & \frac{486\,\pi^7}{7\,T_{\mathrm{i}}^7} & -\frac{81\,\pi^7}{256\,T_{\mathrm{i}}^7} & -\frac{81\,\pi^7}{20\,T_{\mathrm{i}}^7} & \frac{2187\,\pi^7}{256\,T_{\mathrm{i}}^7} & \frac{648\,\pi^7}{7\,T_{\mathrm{i}}^7} & \frac{50625\,\pi^7}{256\,T_{\mathrm{i}}^7} \\ 0 & -\frac{2\,\pi^7}{15\,T_{\mathrm{i}}^7} & 0 & \frac{486\,\pi^7}{7\,T_{\mathrm{i}}^7} & \frac{128\,\pi^8}{T_{\mathrm{i}}^7} & -\frac{2\,\pi^7}{15\,T_{\mathrm{i}}^7} & -\frac{32\,\pi^7}{3\,T_{\mathrm{i}}^7} & -\frac{486\,\pi^7}{7\,T_{\mathrm{i}}^7} & 0 & \frac{6250\,\pi^7}{9\,T_{\mathrm{i}}^7} \\ 0 & \frac{\pi^7}{256\,T_{\mathrm{i}}^7} & -\frac{\pi^7}{24\,T_{\mathrm{i}}^7} & -\frac{81\,\pi^7}{256\,T_{\mathrm{i}}^7} & -\frac{2\,\pi^7}{15\,T_{\mathrm{i}}^7} & \frac{\pi^8}{512\,T_{\mathrm{i}}^7} & \frac{\pi^7}{12\,T_{\mathrm{i}}^7} & 0 & -\frac{8\,\pi^7}{15\,T_{\mathrm{i}}^7} & 0 \\ 0 & \frac{\pi^7}{12\,T_{\mathrm{i}}^7} & 0 & -\frac{81\,\pi^7}{20\,T_{\mathrm{i}}^7} & -\frac{32\,\pi^7}{3\,T_{\mathrm{i}}^7} & \frac{\pi^7}{12\,T_{\mathrm{i}}^7} & \frac{\pi^8}{2\,T_{\mathrm{i}}^7} & \frac{81\,\pi^7}{20\,T_{\mathrm{i}}^7} & 0 & -\frac{625\,\pi^7}{84\,T_{\mathrm{i}}^7} \\ 0 & \frac{81\,\pi^7}{256\,T_{\mathrm{i}}^7} & \frac{243\,\pi^7}{40\,T_{\mathrm{i}}^7} & \frac{2187\,\pi^7}{256\,T_{\mathrm{i}}^7} & -\frac{486\,\pi^7}{7\,T_{\mathrm{i}}^7} & 0 & \frac{81\,\pi^7}{20\,T_{\mathrm{i}}^7} & \frac{6561\,\pi^8}{512\,T_{\mathrm{i}}^7} & \frac{648\,\pi^7}{7\,T_{\mathrm{i}}^7} & 0 \\ 0 & \frac{8\,\pi^7}{15\,T_{\mathrm{i}}^7} & \frac{64\,\pi^7}{3\,T_{\mathrm{i}}^7} & \frac{648\,\pi^7}{7\,T_{\mathrm{i}}^7} & 0 & -\frac{8\,\pi^7}{15\,T_{\mathrm{i}}^7} & 0 & \frac{648\,\pi^7}{7\,T_{\mathrm{i}}^7} & \frac{128\,\pi^8}{T_{\mathrm{i}}^7} & \frac{5000\,\pi^7}{9\,T_{\mathrm{i}}^7} \\ 0 & \frac{625\,\pi^7}{768\,T_{\mathrm{i}}^7} & \frac{3125\,\pi^7}{168\,T_{\mathrm{i}}^7} & \frac{50625\,\pi^7}{256\,T_{\mathrm{i}}^7} & \frac{6250\,\pi^7}{9\,T_{\mathrm{i}}^7} & 0 & -\frac{625\,\pi^7}{84\,T_{\mathrm{i}}^7} & 0 & \frac{5000\,\pi^7}{9\,T_{\mathrm{i}}^7} & \frac{390625\,\pi^8}{512\,T_{\mathrm{i}}^7} \end{bmatrix}. \tag{11}$$

A few key features of the mapping matrices for the two parameterizations are noteworthy.

2.1.2. Key Differences Between Polynomial and FFS Parameterizations

If the time allocated for each segment of the trajectory is known and fixed, a minimizer for the quadratic cost function given in Equation (3) can be computed analytically by following the steps outlined in Section 2.1.3. This implies that, for a fixed set of boundary conditions and a fixed global time, there exists a unique solution. By adjusting the allocation of total time across segments while keeping the global time and boundary conditions constant, a family of solutions can be computed for a set of position waypoints. This approach has been explored in [39] and is reviewed further in Section 2.2.

Both parameterizations—polynomial and FFS—utilize fixed-size mapping matrices, as defined in Equations (6) and (7) for polynomials and Equations (10) and (11) for FFS. The elements of these matrices depend only on the segment time, T_i. Using the matrix decomposition:

$$\tilde{\mathbb{Q}}_\mathrm{i}(T_\mathrm{i}) = \mathbb{A}_\mathrm{i}^{-\top}(T_\mathrm{i})\mathbb{Q}_\mathrm{i}(T_\mathrm{i})\mathbb{A}_\mathrm{i}^{-1}(T_\mathrm{i}), \tag{12}$$

the quadratic cost for either parameterization can be computed efficiently. This decomposition, derived in Section 2.1.3, applies equivalently to both parameterizations, with their differences fully encoded in the mapping matrices $\mathbb{A}_\mathrm{i}(T_\mathrm{i})$ and $\mathbb{Q}_\mathrm{i}(T_\mathrm{i})$.

Although the mapping matrices for both parameterizations have distinct sparsity patterns, the application of Equation (12) results in a fully dense matrix $\tilde{\mathbb{Q}}_\mathrm{i}(T_\mathrm{i}) \in \mathbb{R}^{10\times 10}$. This means that the computational cost of evaluating the quadratic cost function is similar for both. However, differences in sparsity patterns become more evident in the constraint mapping matrices $\mathbb{A}_\mathrm{i}(T_\mathrm{i})$, where 45% of elements are non-zero for polynomials compared to 47% for FFS. When these matrices are inverted, the density increases to 55% for polynomials but reaches 100% for FFS. This increased density in the inverse mapping matrix makes the FFS-based parameterization computationally more demanding, requiring approximately twice the effort compared to the polynomial parameterization, as shown in Table 1. The higher computational cost also limits the scalability of the FFS-based parameterization.

Table 1. Summary of the fixed-time solutions.

Name	Method	n_dt	n_dim	n_int	n_coefs	T (s)	T_solve (ms)	J	P (W)
Simple	Polys	4	1	1	10	3	0.37	301.68	425.60
	FFS	4	1	1	10	3	0.62	400.04	426.36
3 Blocks	Polys	4	3	4	120	9	0.85	90.07	1269.97
	FFS	4	3	4	120	9	1.25	93.85	1270.27
Square	Polys	4	2	8	160	10	0.91	1174.49	1417.68
	FFS	4	2	8	160	10	1.26	1287.14	1418.52
Circle	Polys	4	2	7	140	10	0.85	1840.02	1435.62
	FFS	4	2	7	140	10	1.22	2004.76	1437.65
Eight	Polys	4	4	9	360	30	1.38	1.05	4200.38
	FFS	4	4	9	360	30	2.48	1.15	4200.42

The parameter n_dt (the order of derivative considered) introduces additional distinctions between the two parameterizations. For both polynomial and FFS parameterizations, increasing n_dt expands the size of the mapping matrices, thereby increasing the computational burden and numerical instability. However, polynomial parameterization retains numerical stability due to the compact structure of its basis functions. In contrast, the FFS

parameterization experiences amplified numerical instability as n_{dt} increases. This instability arises from the growth of coefficients in the trigonometric terms of Equations (8) and (9), which makes the inversion of the mapping matrix less stable. For $n_{\mathrm{dt}} \geq 5$, these issues can lead to inaccuracies in the computed solutions, particularly for problems requiring precise continuity of higher-order derivatives. This limitation constrains the practical application of the FFS parameterization in scenarios demanding high-order derivative continuity. Nevertheless, in this work, we assume the minimum order required for minimum-snap optimization and do not introduce additional terms for either polynomial or FFS parameterization. This approach aligns with standard practices in the literature.

Despite these challenges, the FFS-based parameterization offers some theoretical benefits, particularly its inherent "infinite differentiability". This feature ensures smooth trajectories and high-order derivative continuity, making FFS-based parameterization advantageous for applications where such smoothness is critical. However, as noted in Section 2.2, the computational burden and numerical sensitivity must be carefully managed, especially when solving for complex trajectories or applying stringent time constraints.

In summary, the polynomial-based parameterization is computationally more efficient and numerically robust, particularly for higher-order derivatives or large-scale problems. It is well-suited for scenarios requiring fast computations or practical robustness. Conversely, the FFS-based parameterization provides smoother trajectories due to its infinite differentiability, making it advantageous for precision-demanding applications. However, its higher computational cost and numerical instability, especially for higher-order derivatives, necessitate careful consideration and management in real-world applications.

2.1.3. Deriving Analytic Solution for Fixed-Time Problem

The goal of this section is to minimize the quadratic cost function in Equation (2), subject to a set of linear equality constraints represented by the vector $\boldsymbol{d}$. These constraints define various boundary conditions for the trajectory, including position p, velocity v, acceleration a, jerk j, and snap s. The method is flexible, allowing any subset of these boundary conditions to be specified while leaving the remaining conditions unspecified. This adaptability makes the approach suitable for diverse application scenarios.

In this work, we focus on rest-to-rest maneuvers, where all derivatives (p, v, a, j, s) are set to zero at the start and end waypoints. While this assumption simplifies the optimization process and aligns with many prior studies, it is not a limitation of the method. Alternative scenarios, such as non-zero initial or final velocities or accelerations, can be accommodated by specifying the corresponding constraints.

For a single interval along one axis of motion, the vector $\boldsymbol{d}$ represents the boundary conditions, organized as pairs of initial and final values for each derivative. For the k-th interval, the boundary condition vector is as follows:

$$\boldsymbol{d}_k = \left[p_{\mathrm{k}}, v_{\mathrm{k}}, a_{\mathrm{k}}, j_{\mathrm{k}}, s_{\mathrm{k}}, p_{\mathrm{k+1}}, v_{\mathrm{k+1}}, a_{\mathrm{k+1}}, j_{\mathrm{k+1}}, s_{\mathrm{k+1}}\right]^{\top}, \tag{13}$$

where subscripts k and $k+1$ denote the initial and final waypoints of the interval. For multi-segment trajectories, intermediate waypoints typically enforce only positional continuity, leaving higher-order derivatives unspecified. These intermediate conditions are incorporated into the constraint vector $\boldsymbol{d}$ as needed.

To efficiently solve this problem, constraints are divided into two categories: fixed ($\boldsymbol{d}_{\mathrm{F}}$) and free ($\boldsymbol{d}_{\mathrm{P}}$). Fixed constraints include specified boundary conditions and continuity requirements for multi-segment trajectories ($n_{\mathrm{int}} > 1$). In this study, rest-to-rest conditions fix all derivatives at the start and end waypoints, while intermediate waypoints typically contribute only positional constraints for smooth segment transitions.

To reorganize these constraints, a mapping matrix $\mathbb{M}$ is introduced. This matrix serves two purposes, including, (1) it reorders constraints into fixed and free categories; (2) it ensures continuity of derivatives at intermediate waypoints for multi-segment trajectories. The matrix $\mathbb{M}$ is defined as follows:

$$\mathbb{M} = \mathbb{M}_{\mathrm{c}}\mathbb{M}_{\mathrm{r}}, \tag{14}$$

where $\mathbb{M}_{\mathrm{r}} \in \mathbb{R}^{n_{\mathrm{c}} \times n_{\mathrm{c}}}$ is a reordering matrix and $\mathbb{M}_{\mathrm{c}}$ is a continuity matrix. The reordering matrix $\mathbb{M}_{\mathrm{r}}$ places fixed constraints $\boldsymbol{d}_{\mathrm{F}}$ first, followed by free constraints $\boldsymbol{d}_{\mathrm{P}}$. The continuity matrix $\mathbb{M}_{\mathrm{c}}$ enforces consistency of derivatives across shared waypoints for smooth transitions. For a single-interval trajectory, $\mathbb{M}_{\mathrm{c}}$ reduces to an identity matrix. For multi-segment trajectories, $\mathbb{M}_{\mathrm{c}}$ introduces additional rows to duplicate constraints at shared waypoints.

The reorganized constraints can be expressed as follows:

$$\boldsymbol{d} = \mathbb{M}\begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}^{\top} & \boldsymbol{d}_{\mathrm{P}}^{\top}\end{bmatrix}^{\top}. \tag{15}$$

Using the reorganized constraints, the design variable vector $\boldsymbol{p}$ (the trajectory coefficients) is given by

$$\boldsymbol{p} = \mathbb{A}^{-1}\boldsymbol{d} = \mathbb{A}^{-1}\mathbb{M}\begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}^{\top} & \boldsymbol{d}_{\mathrm{P}}^{\top}\end{bmatrix}^{\top}. \tag{16}$$

Substituting Equation (16) into the quadratic cost function reformulates it as,

$$J = \boldsymbol{p}^{\top}\mathbb{Q}\boldsymbol{p} = \begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}\\ \boldsymbol{d}_{\mathrm{P}}\end{bmatrix}^{\top} \underbrace{\mathbb{M}^{\top}\mathbb{A}^{-\top}\mathbb{Q}\mathbb{A}^{-1}\mathbb{M}}_{\mathbb{R}} \begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}\\ \boldsymbol{d}_{\mathrm{P}}\end{bmatrix}, \tag{17}$$

where $\mathbb{R}$ is partitioned into four sub-blocks:

$$J = \begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}\\ \boldsymbol{d}_{\mathrm{P}}\end{bmatrix}^{\top} \begin{bmatrix}R_{\mathrm{FF}} & R_{\mathrm{FP}}\\ R_{\mathrm{PF}} & R_{\mathrm{PP}}\end{bmatrix} \begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}\\ \boldsymbol{d}_{\mathrm{P}}\end{bmatrix}. \tag{18}$$

Finally, by minimizing the cost function with respect to the free constraints $\boldsymbol{d}_{\mathrm{P}}$, the optimal set of coefficients $\boldsymbol{p}^{*}$ is determined. This procedure applies equally to both polynomial- and FFS-based parameterizations, with differences encoded in the matrices $\mathbb{A}$ and $\mathbb{Q}$, which depend on the parameterization method and time allocation. To accomplish this, we expand Equation (18) to obtain the following expression:

$$J = \boldsymbol{d}_{\mathrm{F}}^{\top}R_{\mathrm{FF}}\boldsymbol{d}_{\mathrm{F}} + \boldsymbol{d}_{\mathrm{F}}^{\top}R_{\mathrm{FP}}\boldsymbol{d}_{\mathrm{P}} + \boldsymbol{d}_{\mathrm{P}}^{\top}R_{\mathrm{PF}}\boldsymbol{d}_{\mathrm{F}} + \boldsymbol{d}_{\mathrm{P}}^{\top}R_{\mathrm{PP}}\boldsymbol{d}_{\mathrm{P}}. \tag{19}$$

We can then find the minimum of the cost function as $\partial \boldsymbol{J}/\partial \boldsymbol{d}_{\mathrm{P}} = \boldsymbol{d}_{\mathrm{F}}^{\top}R_{\mathrm{FP}} + R_{\mathrm{PF}}\boldsymbol{d}_{\mathrm{F}} + 2R_{\mathrm{PP}}\boldsymbol{d}_{\mathrm{P}} = 2R_{\mathrm{PF}}\boldsymbol{d}_{\mathrm{F}} + 2R_{\mathrm{PP}}\boldsymbol{d}_{\mathrm{P}} = \boldsymbol{0}$, where both R_{PF} and R_{FP} are assumed to be transposes of each other ($R_{\mathrm{PF}} = R_{\mathrm{FP}}^{\top}$). The vector of free derivatives can be written as,

$$\boldsymbol{d}_{\mathrm{P}}^{*} = -R_{\mathrm{PP}}^{-1}R_{\mathrm{PF}}\boldsymbol{d}_{\mathrm{F}} = -R_{\mathrm{PP}}^{-1}R_{\mathrm{FP}}^{\top}\boldsymbol{d}_{\mathrm{F}}. \tag{20}$$

The optimal set of design parameters can be obtained by substituting the result back into Equation (16):

$$\boldsymbol{p}^{*} = \mathbb{A}^{-1}\mathbb{M}\begin{bmatrix}\boldsymbol{d}_{\mathrm{F}}^{\top}, \boldsymbol{d}_{\mathrm{P}}^{*\top}\end{bmatrix}^{\top}. \tag{21}$$

At this point, we have found an optimal solution to a fixed-time unconstrained problem without any iterations. The exact same procedure is applied to both polynomial- and FFS-based parameterizations. In fact, the only differences between the two parameterizations are reflected in the values of the mapping matrix, $\mathbb{A}$. Recall that both the quadratic mapping matrix, $\mathbb{Q}$, and linear mapping matrix, $\mathbb{A}$, are only functions of the total time allocated for each segment. In fact, there exists a simple decoupling for both. Consider a single-interval, single-axis case for simplicity. Matrices $\mathbb{Q}$ and $\mathbb{A}$ can be re-written as the following products: $\mathbb{Q} = \mathbb{Q}_s\bar{\mathbb{Q}}\mathbb{Q}_s$ and $\mathbb{A} = \mathbb{A}_s\bar{\mathbb{A}}$, where $\mathbb{Q}_s$ is a square positive diagonal time-scaling matrix and $\bar{\mathbb{Q}}$ is a constant square matrix. Similarly, $\mathbb{A}_s$ is a square symmetric time-scaling matrix (not diagonal) for constant square matrix $\bar{\mathbb{A}}$. In fact, $\bar{\mathbb{Q}}$ and $\bar{\mathbb{A}}$ are only defined by the cost function and the parameterization method and are invariant to constraints or time allocation and are never changed unless the order of derivatives considered is altered. The cost function given in Equation (18) can be rewritten as follows:

$$\begin{aligned} J &= \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^\top \mathbb{M}^\top \mathbb{A}^{-\top}\mathbb{Q}\mathbb{A}^{-1}\mathbb{M}\begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^\top \mathbb{M}^\top \overbrace{\bar{\mathbb{A}}_s\mathbb{Q}_s}^{\mathbb{S}} \underbrace{\bar{\mathbb{A}}^{-1}\bar{\mathbb{Q}}\bar{\mathbb{A}}^{-\top}}_{\overline{\mathbb{Q}}} \overbrace{\bar{\mathbb{A}}_s\mathbb{Q}_s}^{\mathbb{S}}\mathbb{M}\begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^\top \mathbb{M}^\top\mathbb{S}\overline{\mathbb{Q}}\mathbb{S}\mathbb{M}\begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}. \end{aligned} \tag{22}$$

What is intriguing is that the scaling pattern given by $\mathbb{S}(\boldsymbol{T}) = \mathbb{A}_s(\boldsymbol{T})\mathbb{Q}_s(\boldsymbol{T})$ is independent of the parameterization method used and is only defined by the cost function, or the order of the derivative considered (minimum jerk, minimum snap, etc.). This indicates that a major part of the problem, $\overline{\mathbb{Q}}$, remains completely invariant to time and, with additional manipulations, can lead to the following time-scaled form:

$$\min \bar{J} = \bar{\boldsymbol{p}}^\top\mathbb{Q}\bar{\boldsymbol{p}} = \begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}^\top \mathbb{M}^\top\overline{\mathbb{Q}}\mathbb{M}\begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}, \text{ where } \bar{\boldsymbol{p}} = \mathbb{S}^{-1}\boldsymbol{p} = \mathbb{Q}_s^{-1}\bar{\mathbb{A}}^{-1}\mathbb{M}\begin{bmatrix} \boldsymbol{d}_F \\ \boldsymbol{d}_P \end{bmatrix}. \tag{23}$$

The problem given in Equation (23) can be solved without time factored into the solution for $\bar{\boldsymbol{p}}$ and the time-dependent coefficients can be recovered using $\mathbb{S}(\boldsymbol{T})$. However, this approach does not appear to offer any advantages over directly solving for the coefficients with time factored in. Although it may appear simpler, the original problem must still be solved to account for the new free constraints, so it presents no computational advantage. Since both $\mathbb{A}_s$ and $\mathbb{Q}_s$ contain elements with time raised to large and low powers simultaneously, the combined contribution of those elements (after solving for the free constraints) amplify the numerical error and final trajectory (in time domain) may have visible discontinuities at some of the points. This effect is demonstrated with the fixed-time solutions and their time-optimal counterparts in the subsequent sections.

Please note that the solutions to the minimum-snap optimization problems are stationary/extremal solutions, since first-order necessary conditions are used to obtain the functional form of the solution [28]. The solutions of the resulting QPs are, however, global solutions, as QPs are convex optimization problems, and any local solution to a convex optimization problem is the unique global solution. In this paper, extremal and optimal are used interchangeably.

The time-decoupled formulation derived in Equation (23) provides important insights: (1) the only time-dependent terms are the boundary conditions, specifically the derivatives; (2) a unique globally optimal solution exists for a given set of boundary conditions; and (3) when specific segment times are not enforced by fixed velocity, acceleration, or higher-derivative constraints along the trajectory, a globally optimal solution valid across different

total flight times can still be determined. This is contingent upon maintaining a constant ratio of time allocated per segment. Notably, scaling fixed-time problems by a time factor of 10 enables stable solutions that were previously unattainable with the unscaled FFS-based parameterization. This scaling approach is equally applicable to the FFS- and polynomial-based parameterizations, but is crucial for ensuring the robustness of the FFS parameterization to be on par with polynomial methods.

The FFS-based parameterization offers unique advantages, such as piecewise infinite differentiability, but it also poses challenges, including a susceptibility to overfitting and higher computational complexity. To mitigate overfitting, the number of Fourier coefficients is limited to the minimum necessary to ensure trajectory smoothness, avoiding over-fitting that could degrade performance. Computational complexity is addressed by leveraging matrix sparsity during optimization, significantly reducing the impact of dense computations inherent to the FFS-based parameterization. This ensures the FFS-based parameterization remains both computationally feasible and robust for practical applications.

2.2. Time-Allocation Problem

In the previous section, it was assumed that the total time allocated for each interval is constant. If the allocation is performed correctly, an optimal trajectory can be directly obtained for that sequence of time segments. However, in practice, the trajectory designer may not know the exact timing for every waypoint of interest. More importantly, the global time for the entire trajectory often serves as a more critical design variable than the specific time allocation for each individual interval.

Consider the same setup as the fixed-time problem discussed earlier, where a sequence of waypoints is given, and intermediate boundary conditions (if any) are specified. In this case, there is no explicit time allocated for each interval, and the algorithm must determine the optimal time allocation. This approach allows for greater flexibility and adaptability in realistic scenarios where precise timing information is unavailable.

To address this, the quadratic cost function in Equation (2) is augmented with a linear term weighted by a constant k_{T}, which penalizes the total time allocation vector $\boldsymbol{T} = [T_1, \ldots, T_{\mathrm{n_{int}}}]^\top$:

$$J_T = \boldsymbol{p}^\top \mathbb{Q} \boldsymbol{p} + k_{\mathrm{T}} \sum_{\mathrm{i}=1}^{n_{\mathrm{int}}} T_{\mathrm{i}}, \qquad \text{s.t.} \quad T_i > 0, \tag{24}$$

where $T = \sum_{\mathrm{i}=1}^{n_{\mathrm{int}}} T_{\mathrm{i}}$ represents the global time for the entire trajectory, and T_{i} denotes the time of flight between waypoints i and $i+1$. The time penalty gain k_{T} can be adjusted by the user to increase or decrease the total flight time. Notably, the same value of k_{T} may yield different results for the two parameterizations (polynomial and FFS) despite identical solution schemes and inputs. This discrepancy arises because the FFS-based parameterization incurs a higher cost due to its denser mapping matrices. Consequently, a larger k_{T} gain is typically required for the FFS-based parameterization to match the global time of a polynomial-based solution.

The problem defined in Equation (24) is solved using quasi-Newton gradient-based NLP solvers such as *MATLAB*'s `fmincon`. Linear inequality constraints are imposed on the time allocated to each segment, and the gradient is computed numerically by solving the fixed-time problem iteratively. This formulation retains the unconstrained nature of the boundary conditions, ensuring computational efficiency, and enables flexible time allocation.

Proper time allocation is critical to achieving trajectory optimality, as it minimizes control effort and, consequently, power consumption (minimum-snap). For example, unconstrained optimization allows even the most demanding maneuvers to become significantly less resource-intensive by intelligently redistributing time along the trajectory. The result-

ing trajectories are an order of magnitude less demanding than equivalent maneuvers performed manually, demonstrating the method's practical applicability.

An alternative formulation can be used to avoid introducing the k_{T} gain, where the time allocation is expressed in scaled units with a fixed constraint on total flight time, T, as,

$$\bar{J}_T = \bar{\boldsymbol{p}}^\top \mathbb{Q} \bar{\boldsymbol{p}}, \qquad \text{s.t.} \quad \bar{T}_i > 0 \quad \text{and} \quad \sum_{\mathrm{i}=1}^{n_{\mathrm{int}}} \bar{T}_{\mathrm{i}} = 10, \tag{25}$$

where the problem is internally mapped into scaled time units, and the actual trajectory segment time t is recovered during polynomial or FFS evaluation through rescaling by the desired time of flight T. As previously mentioned, maintaining the ratio of per-segment time to total flight time $\left(\frac{T_i}{T} = \frac{\bar{T}_i}{10}\right)$ ensures that any solution can be re-evaluated for longer or shorter trajectories without additional optimization. This formulation improves convergence robustness in complex scenarios requiring very large or small total flight times. Even without this modification, the formulation in Equation (24) allows for total flight time to be updated as a post-processing step via rescaling. In other words, the following property holds true:

$$\frac{T_i}{T} = C_i, \qquad \text{where } C_i \text{ is constant,} \tag{26}$$

for a fixed set of boundary conditions for any time of flight. The continuity constraints for free boundaries are also respected on the entire trajectory if properly rescaled, without any loss of optimality or additional optimization. An exact same result can be obtained by resolving the scaled problem for a desired time of flight. However, for this study, Equation (24) was chosen to align with the cost function implemented in the *MATLAB* toolbox, ensuring reproducibility and validation by readers.

We tested the performance of this algorithm with hand-picked waypoints and integrated it with a higher-level randomized path-planning algorithm, RRT*. A virtual 3D grid was constructed to emulate the flying arena in our laboratory. Both simulation and experimental results are presented in Section 3. Additionally, this method can be extended to enforce realistic constraints, such as obstacle avoidance or corridor constraints, by augmenting the optimization with inequality conditions at the waypoints. With a sufficient number of waypoints, such constraints can be enforced during time allocation, redistributing time as needed to satisfy these requirements.

2.3. Simulation and Experimental Setup

For theoretical validation of the two methods, we have developed a 6DoF quadrotor simulation with its parameters given in [62]. System dynamics, control system model, and implementation details are identical to those in [63]. In summary, the simulation considers standard 6DoF rigid body dynamics of a quadrotor, with all four motors producing thrust perpendicular to the $x-y$ in a north–east–down body frame. The actuator dynamics is considered to be ideal (no actuator delay). Also, no disturbances such as wind, sensor noise, etc., are considered. To give some physical meaning to the numerical results presented later in the paper, we have extracted the power consumption from the simulated quadrotor flight and included it with the rest of the numerical results. Although the minimum-snap cost given in Equation (3) is directly representative of the power consumed by an ideal quadrotor, we have defined (with an abuse of notation) the power consumed as follows:

$$P = \int_{t_0}^{t_{\mathrm{f}}} \sum_{\mathrm{i}=1}^{4} k\omega_{\mathrm{i}}^3 dt, \qquad k = 3.0 \times 10^{-9}\ N\ m/RPM^2, \tag{27}$$

where ω_i is the motor speed in revolutions per minute (RPM). The quadrotor system dynamics with the control system described in [62] was simulated in order to obtain this theoretical power consumption. This work does not consider aerodynamic effects and does not account for propeller–blade interactions, which can be significant for multirotor vehicles at high speeds.

All the motion-planing algorithms and simulations were developed in *MATLAB* and *Simulink*. All experiments were performed using the facilities of the Aero-Astro Computational and Experimental (ACE) laboratory at Auburn University. For validating the motion-planning algorithms, a full-sized quadrotor was assembled and flown in the ACE laboratory (Figure 2). All trajectories presented in Section 3 were computed off-board using *MATLAB* environment. The laptop used for generating all the trajectories runs an *Intel*® i7-10750H 2.60 GHz CPU. Once optimizations were complete, all trajectories were uploaded to the quadrotor as text files and executed using a remote keyboard input from the ground computer to initiate the maneuver.

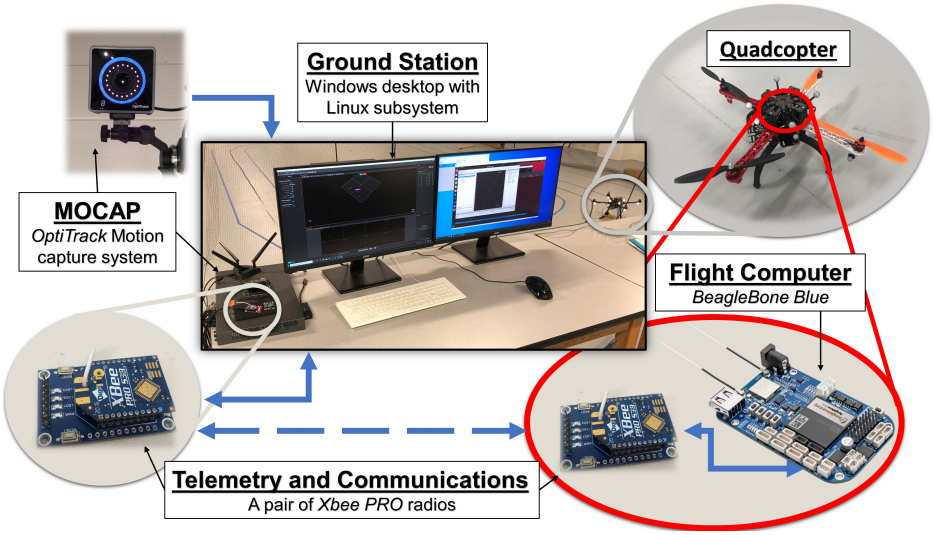


Figure 2. Hardware data flow schematics.

We used our own quadrotor firmware (ACE-pilot) written in C and C++. The onboard Linux computer, *BeagleBone*® *Blue*, was only used to run the flight control system and its peripherals. The communications with the ground station were handled by a pair of *Xbee Pro* radio modules, with a custom data-packet format. The position and heading of the quadrotor were not estimated on-board but sent from the ground station using an OptiTrack motion capture system. The trajectories were pre-computed offboard, checked for errors, prior to be loaded in a point-by-point format to the onboard firmware and used to assign in-flight position setpoints. No feedforward control was used for this work. To simplify the trajectory generation process, all controls were assumed to be performed in a local, initial position corrected, coordinate frame, which makes all trajectories (internally) start from zero (position and heading). The quadrotor was to log the initial state in the inertial frame when a new trajectory was to be executed and used the incoming trajectory references as offsets to the initial position. The same strategy was applied for continuous heading reference setpoints. This way, the same trajectory can be executed at any point (independently of the inertial position coordinate and heading angle). A simple path-following strategy was adopted. The reference position to the flight control system, $P_{\text{ref}}(t)$, was interpolated from a pre-computed optimal trajectory based on time-since-epoch, t, or the start of the trajectory. The deviation, Δ, from the intended path was defined as follows:

$$\Delta P_k(t) = P_{\text{ref}}(t) - P_k(t), \qquad \text{for,} \qquad k \in \{x,\ y,\ z,\ \psi\}, \tag{28}$$

where the state of the vehicle is estimated using *BeagleBone® Blue*'s built-in inertial-measurement unit and the motion capture system. Although there can be several enhancements made to the path-following strategy (e.g., considering perpendicular distance from the intended path to compensate for the time-lag), development of an advanced path-following strategy is considered to be out of the scope of this work, since it will only improve the overall tracking performance. The exact same inputs for trajectory generation, path-following strategy, control system, hardware, and overall setup have been maintained for all experimental flight test. The only difference is due to the parameterization method used to generate an optimal trajectory, which is converted into a set of reference position and heading reference points to the control system.

3. Numerical Results

3.1. Fixed-Time Solutions

We start by examining fixed-time solutions with an evenly-distributed time allocation. For the sake of simplicity, all trajectories discussed in this section are summarized in Table 1 and they are characterized in terms of the following parameters: (a) n_{dt}, which denotes the order of derivative considered for boundary conditions. Since we aim to compute minimum-snap trajectories, $n_{\text{dt}} = 4$ for both the cost function and the boundary conditions. This also sets the number of coefficients per interval to be ten; (b) n_{dim}, which denotes the number of dimensions for which motion-planning is performed. This number defines how many active degrees of freedom are used for optimization. Even though we are primarily interested in 3D motion, some trajectories consider constant altitude or only planar XY motion. Any set of waypoints that is zero or constant for a degree of freedom can be considered to always lead to a trivial solution and is removed from the optimization to reduce the computational time; (c) n_{int}, which denotes the number of intervals/segments between two waypoints for a single dimension. It is assumed that all dimensions have the same waypoint discretization and no optimization is performed to determine the optimal number of waypoints or intervals; (d) T, which denotes the total time allocated for the trajectory. Not to be mistaken for the vector of individual time segments $\boldsymbol{T}$ allocated for each interval. This defines the total time it physically takes to execute or fly the trajectory. For fixed-time problems, this quantity is decided by the trajectory designer and is fixed; (e) T_{solve}, which denotes the time it takes to compute a solution. This quantity defines how long it took to solve a complete minimum-snap motion-planning problem. This number does not account for initial user-input setup, solution evaluation or plotting; (f) J, which denotes the final value of the minimum-snap cost function. Trajectory design requirements are generally different and depend on the application, even for vehicles of the same class.

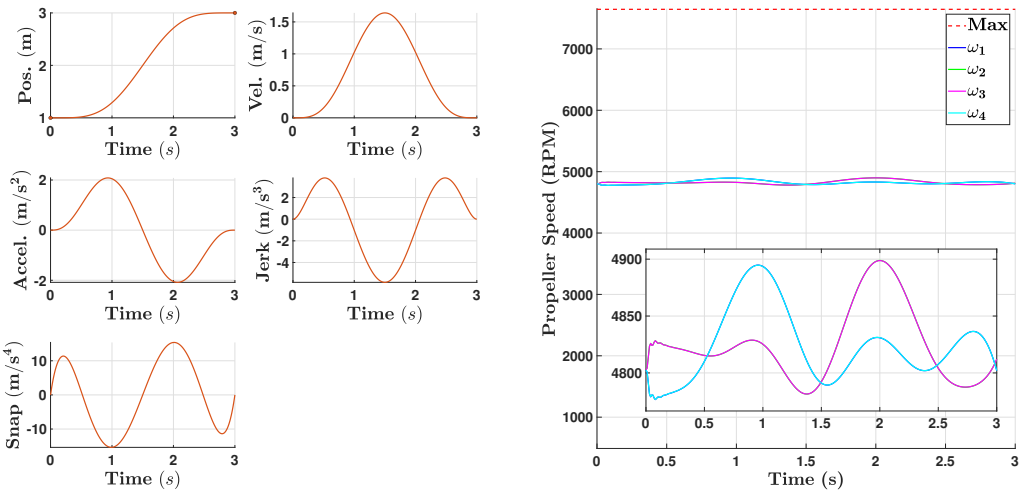
3.1.1. "Simple" Trajectory

This trajectory considers motion along a single axis and only between two waypoints. It serves as a baseline for computational speed comparison and control system performance for the different parameterizations. This is considered the simplest case because we directly solve the problem derived in Section 2.1.3 with mapping matrices being $\mathbb{A}(\boldsymbol{T}) = \mathbb{A}_1(T)$ and $\mathbb{Q}(\boldsymbol{T}) = \mathbb{Q}_1(T)$, where there is only one segment, so the time allocation vector is a scalar with only one element $\boldsymbol{T} = T = T_1$. The mapping matrix $\mathbb{M}$ is simply the diagonal matrix, since no reordering or duplication of intermediate constraints is needed. The input waypoints are the start and end points at one and three meters, respectively. All other axes

are held constant at their initialization values, which, in the case of the simulation, are all zeros. Trajectory optimization is performed on the X axis. First, the trajectory is solved outside the simulator and the required position, velocity, acceleration, jerk, and snap are evaluated using the optimal coefficients.

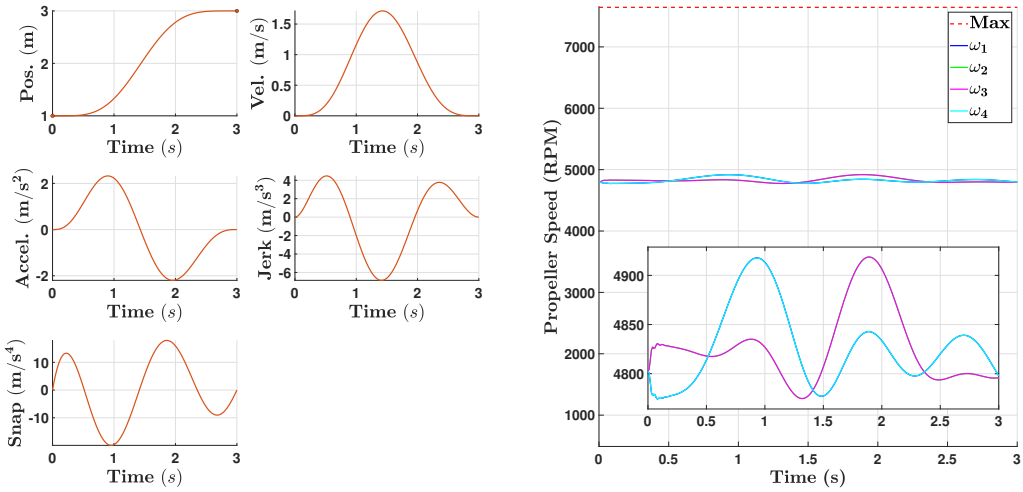
As can be seen in Figures 3a and 4a, the path and its derivatives are identical. The boundary-value problem is actually almost trivial in nature due to the assumptions made earlier such that the derivatives at the start and end points of the trajectory are all zeros and only the initial and final positions, in this case, are non-zero. The solution process is simple enough to be checked by hand and such a one-dimensional, single-interval example is quite common to be applied in practice, where a simple connecting arc between just two points is required; for example, for takeoff or landing. Although it is more common to use a cubic polynomial, the ninth-degree polynomial and similar complexity FFS solutions lead to almost identical solutions. The difference is that the cubic polynomial is not continuous up to the snap level. Sharp changes in motor speed are expected at the beginning and end of the cubic-polynomial solution. The resulting motor RPM profiles serve as the final arbiter for the performance of the control system and the trajectory generation algorithm.

For this reason, control system signal plots are omitted, but all four motor RPM profiles, labeled with different colors, are shown in Figures 3b and 4b. The sense of rotation of the propellers is given in Figure 1. Motor RPM profiles are identical for the two parameterizations and lead to the almost equivalent power consumption of 425.6 W for polynomial and 426.36 W for FFS. The values of the minimum-snap cost functions are 301.68 and 400.04 for polynomial and FFS parameterizations. Figures 3b and 4b contain the global plot in the background, where motor profiles are plotted with colored lines on the same scale as their maximum values (dashed red line). Since almost all trajectories do not show significant deviation of the motor RPM profile along the trajectory, a zoomed-in view of the same motor profiles is included on each of the plots. The vertical and horizontal axes are identical for both the background graph and a zoomed-in view. Due to the simple nature of this one-dimensional (1D) trajectory, the four motor RPM profiles are not fully distinguishable from each other, and only overlaid pairs are visible.



(**a**) Single axis information. (**b**) RPM of propellers vs. time.

Figure 3. Simulated results for the "Simple" trajectory with the polynomial parameterization.

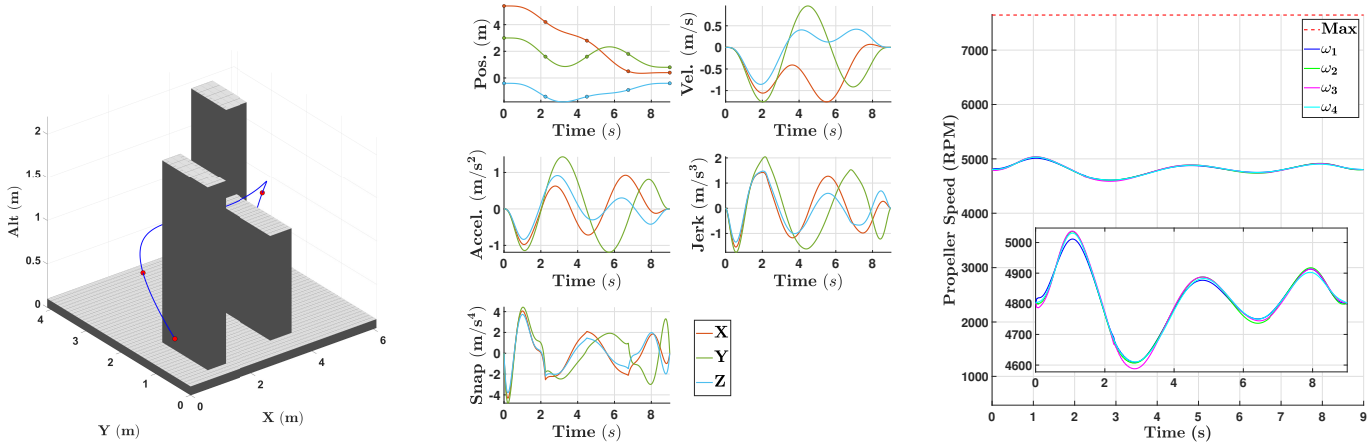


(**a**) Single axis information. (**b**) RPM of propellers vs. time.

Figure 4. Simulated results for the "Simple" trajectory with the FFS parameterization.

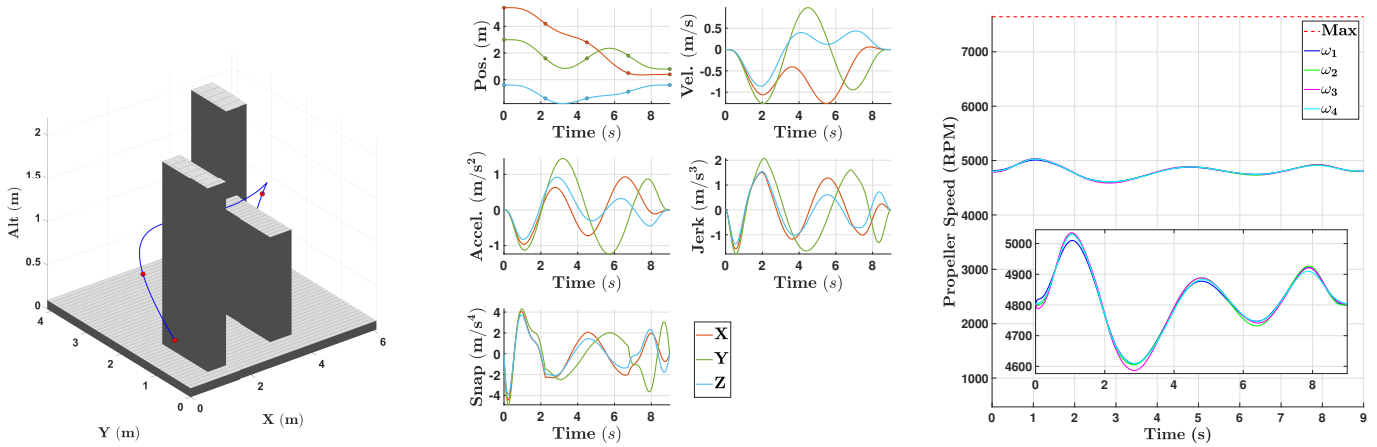
3.1.2. "3 Blocks" Trajectory

This trajectory was designed as an extension of the previous simple 1D case, where an open-contour set of points between the start and end goal is required to be joined by a feasible trajectory. First, a 3D grid with a resolution of one decimeter is constructed to reproduce the flyable laboratory space in a virtual environment. The objective was to guide the quadrotor from one corner of the lab to the other. To complicate the path-planning task, three static rectangular obstacles were placed between the starting point and the destination. Next, the random-search algorithm *RRT** of *MATLAB* was used to generate a feasible path connecting the start and end points in straight lines. Since there was no collision-free straight line path between the start and goal pose, the algorithm had to introduce intermediate waypoints in between to avoid any collision. The resulting set of waypoints was then used to compute the optimal trajectory using the algorithm derived earlier. This time, the motion was in 3D space with variable waypoints in X, Y, and Z axes. The heading angle was not considered for this problem, so it was assumed to be held constant. The input waypoints are marked with red dots on 3D plots shown in Figures 5a and 6a. Note that, for visualization purposes, the trajectory generated in NED coordinate frame with its Z axis pointing down was flipped to represent altitude or $Alt = -Z$.



(**a**) 3D view with obstacles. (**b**) Single axis information. (**c**) Motor speed.

Figure 5. Simulated results for the "3 Blocks" trajectory with the polynomial parameterization.



(**a**) 3D view with obstacles. (**b**) Single axis information. (**c**) Motor speed.

Figure 6. Simulated results for the "3 Blocks" trajectory with the FFS parameterization.

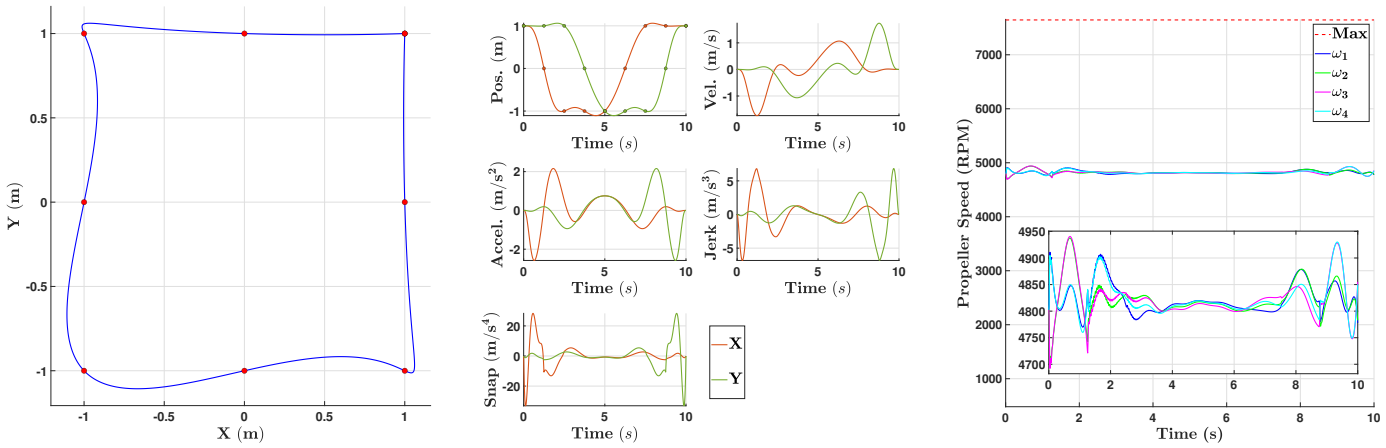
The resulting trajectories for the polynomial-based (Figure 5a) and FFS-based (Figure 6a) parameterizations are identical within the numerical precision. The shape does not look distorted for FFS parameterization and fully follows the polynomial solution. A very small, almost unnoticeable difference is present at the snap level between the two solutions (Figures 5b and 6b). The total power consumption is 1269.97 W and 1270.27 W for the polynomial and FFS parameterizations, respectively. The minimum-snap cost values are 90.07 for polynomial and 93.85 for the FFS method, which highlights that the cost function values do not directly translate to power consumption. The only general trend is that the cost value and total power consumption, both, are higher for the FFS parameterization, which is expected since, according to the principles of calculus of variations, the family of extremal solutions is represented by polynomials. Nevertheless, our objective is to present an alternative smooth trajectory generation method. Even though this trajectory requires the quadrotor to take off, fly over the obstacle, and dodge the other two, plenty of time has been given for the maneuver. This results in a very conservative change in motor profiles, as can be seen in Figures 5c and 6c. A zoomed-in view of the same figures shows a slight variation in all four motor RPM profiles. Although almost identical, motor RPM profiles corresponding to the polynomial solution have a very minor twitch at approximately 2.2 s, which is not present in the FFS solution. The two solutions are smooth and FFS again closely matches with the solution of the polynomial parameterization.

3.1.3. "Square" Trajectory

Next, we consider a somewhat more standard trajectory. The waypoints for this trajectory define the four corners of a square. Additional waypoints are placed midway between each of the corners to constrain the shape and enforce a motion close to straight lines. This square path is completely in the horizontal $X - Y$ plane, with time being evenly discretized between all the waypoints. Altitude and heading are assumed to remain constant. This trajectory aims to decouple the motion along the X and Y axes and forces the optimal solution to approximate straight lines and corners. For this closed trajectory, the start and end waypoints are at the same location, $X = 1.0$ m and $Y = 1.0$ m, which also defines the first corner of a 2×2 meter square. The motion is counterclockwise.

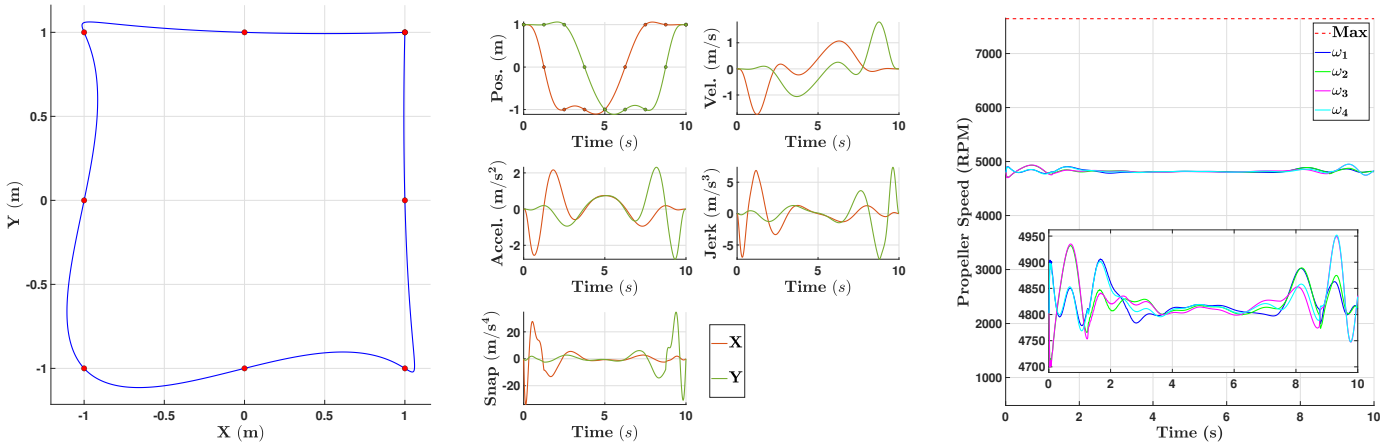
Similarly to the test cases shown earlier, the resulting trajectories are identical, within some numerical error, for the two parameterizations. As can be seen on 2D plots in Figures 7a and 8a, the differences in the two shapes are indistinguishable. However, some very minor differences exist between the snap profiles shown in Figures 7b and 8b at about 1.2 and 8.5 s. Although very similar, the FFS solution might appear to be somewhat

smoother because it lacks the same small spikes on the snap level of the polynomial solution. This very minor fluctuation (at 1.2 s) is also reflected in the profiles of the motor RPM profiles (Figures 7c and 8c). Of course, the effect is minimal and is most likely purely attributed to the numerical convergence of the two methods. As is shown later, similar numerical spikes in the snap are also present for some of the FFS solutions. The power consumption (although negligible) is still in favor of the polynomial solution with a value of 1417.68 W, while the FFS solution consumes 1418.52 W of power. The minimum-snap cost values are 1174.49 for polynomial and 1287.14 for FFS methods.



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 7. Simulated results for the "Square" trajectory with the polynomial parameterization.



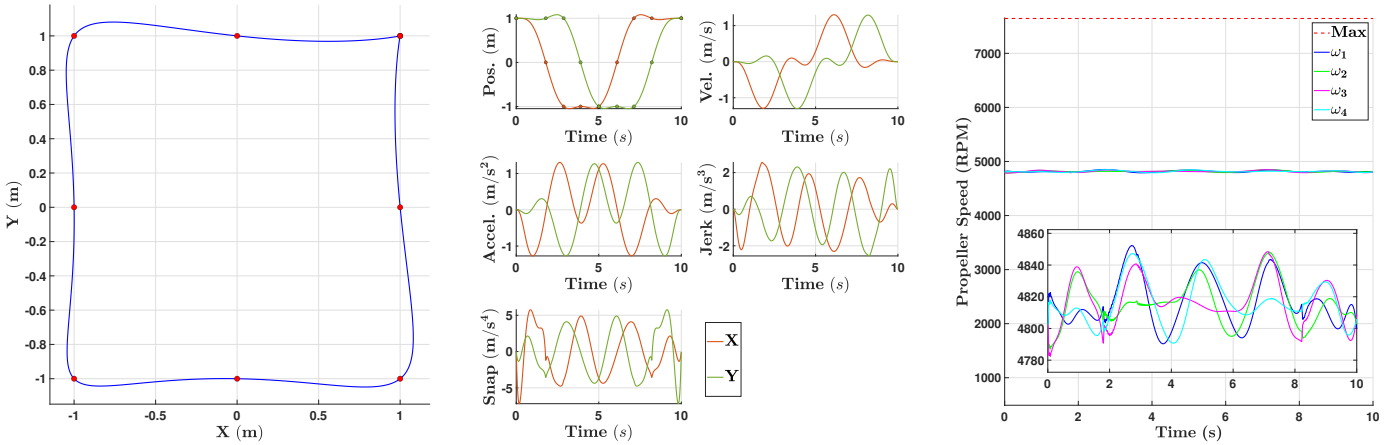
(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 8. Simulated results for the "Square" trajectory with the FFS parameterization.

Note that the method is essentially trying to match all boundary conditions (each waypoint), defined by position, velocity, acceleration, etc., at a particular point in space (the red markers in Figure 7a,b and every other similar trajectory plot). For all the problems presented in this paper, the position of each waypoint has been pre-determined or fixed, while derivatives (at those specific locations) are all free (except from the very first and last waypoint). The curves that connect all those waypoints sequentially are, technically, arbitrary (there are no additional constraints along each interval). It was also assumed that the vehicle starts from rest and comes to rest at the first and last waypoint, respectively.

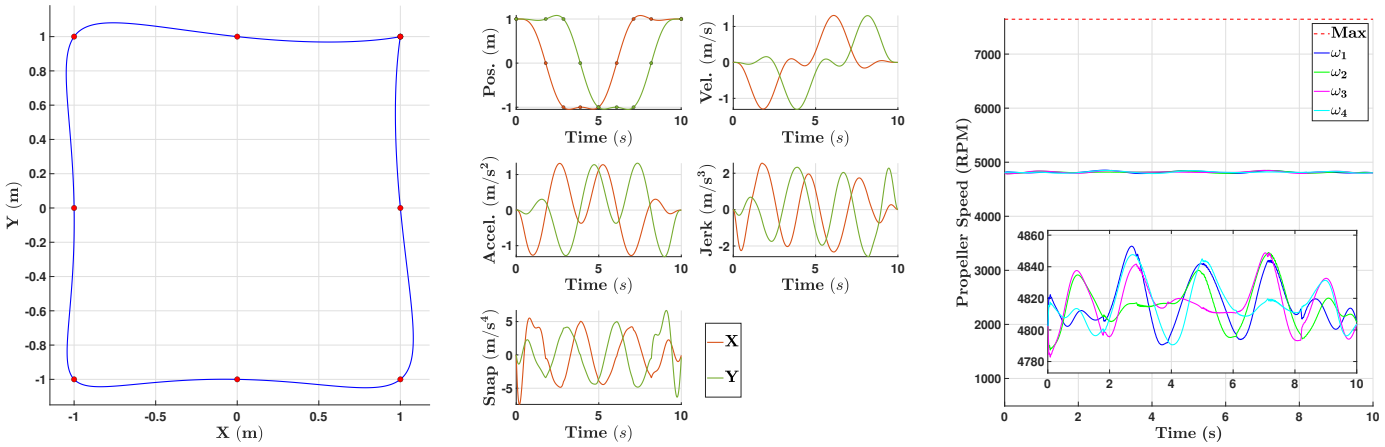
The fixed-time solution (Section 2.1.3) assumes that the time has been pre-determined before solving the fixed-time problem (hence the name) and each waypoint must be reached at that specified time instance, while minimizing the snap. Obviously, this is very restrictive

and requires some "smart" way of selecting the time allocation vector. In this case, time has been simply linearly discretized between 0 and 10 s (to highlight the benefits of the time allocation solution for this exact problem, see Figures 9 and 10) .



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

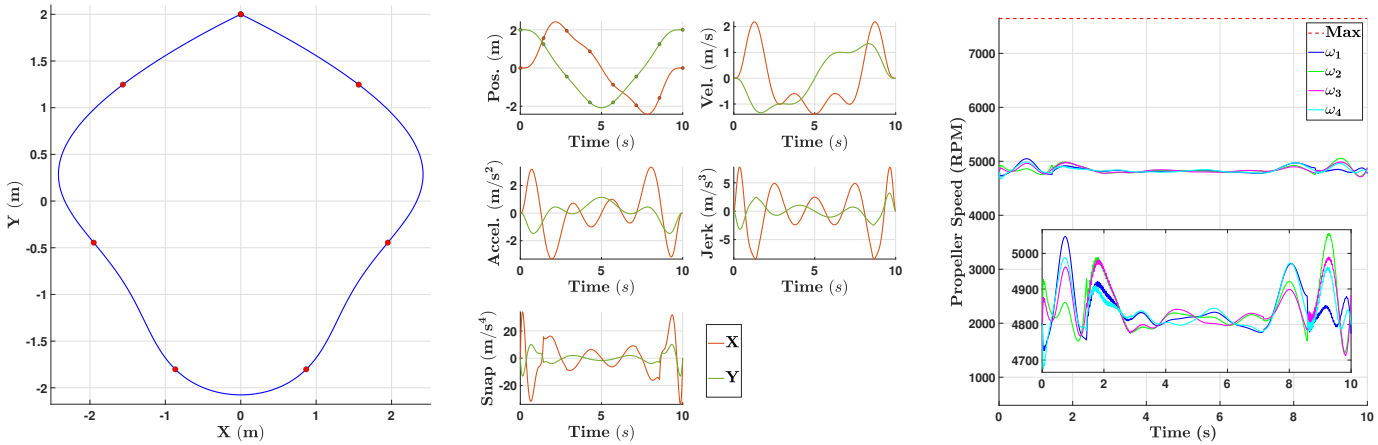
Figure 9. Simulated results for the time-allocated "Square" trajectory with polynomial parameterization.



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

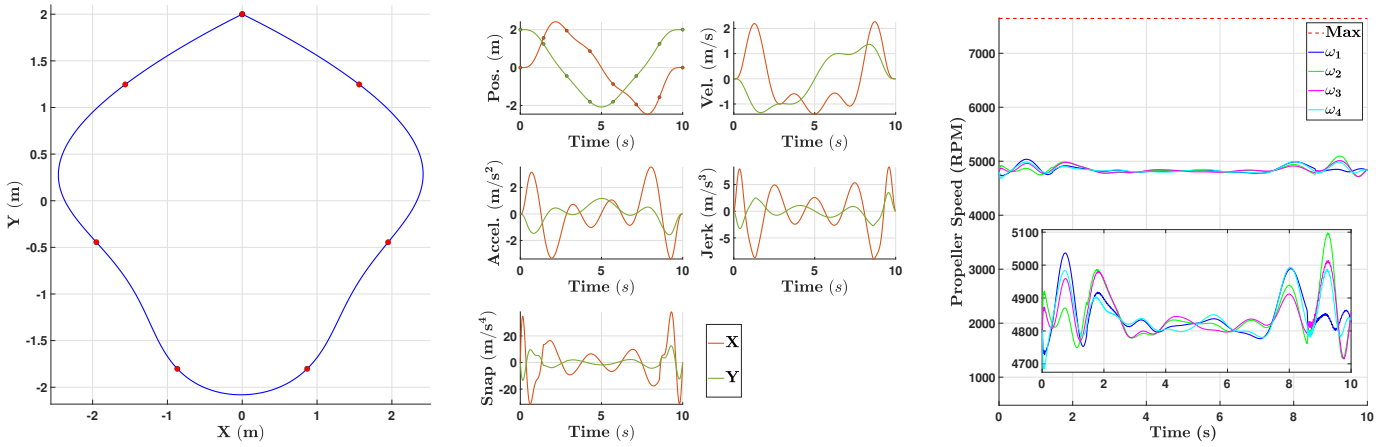
Figure 10. Simulated results for the time-allocated "Square" trajectory with FFS parameterization.

The rationale emphasized above also leads to the initially unexpected shape of the square trajectory (and presented later "circle" trajectory in Figures 11 and 12). In essence, the vehicle has to start from hover, speed up, fly thought the intermediate waypoints, and come to a full stop at the very last waypoint, while respecting the exact flight time at each of the waypoints along the trajectory. It should be intuitive that accelerating and decelerating are very power-demanding actions for a quadrotor. Ideally, acceleration and deceleration should be spread out along the trajectory by allocating more time for the particularly demanding intervals. The sharp turns (corners) are also suboptimal (especially if the velocity and other derivatives are not enforced at those corners); therefore, they should be smoothened if time allows, just like the high-speed racetracks avoid making exactly 90-degree turns. However, since the time has been evenly allocated, the initial (top) and final (right) intervals are almost straight, which indicates that there is not enough time for any deviation. On the other hand, the two intermediate intervals (left and bottom) have too much time assigned to them; therefore, the trajectory has to deviate from the straight path significantly.



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 11. Simulated results for the "Circle" trajectory with the polynomial parameterization.



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 12. Simulated results for the "Circle" trajectory with the FFS parameterization.

3.1.4. "Circle" Trajectory

Even though it may not look circular at all from the fixed-time paths shown in Figures 11a and 12a, the waypoints are tracing a perfect circular shape. This 2D trajectory is very similar to the "Square" one, with time allocated evenly for $t_f = 10$ s. The intent is again to obtain a simple circular path which, intuitively, should be the optimal shape for a set of waypoints that already trace a circle of a constant radius.

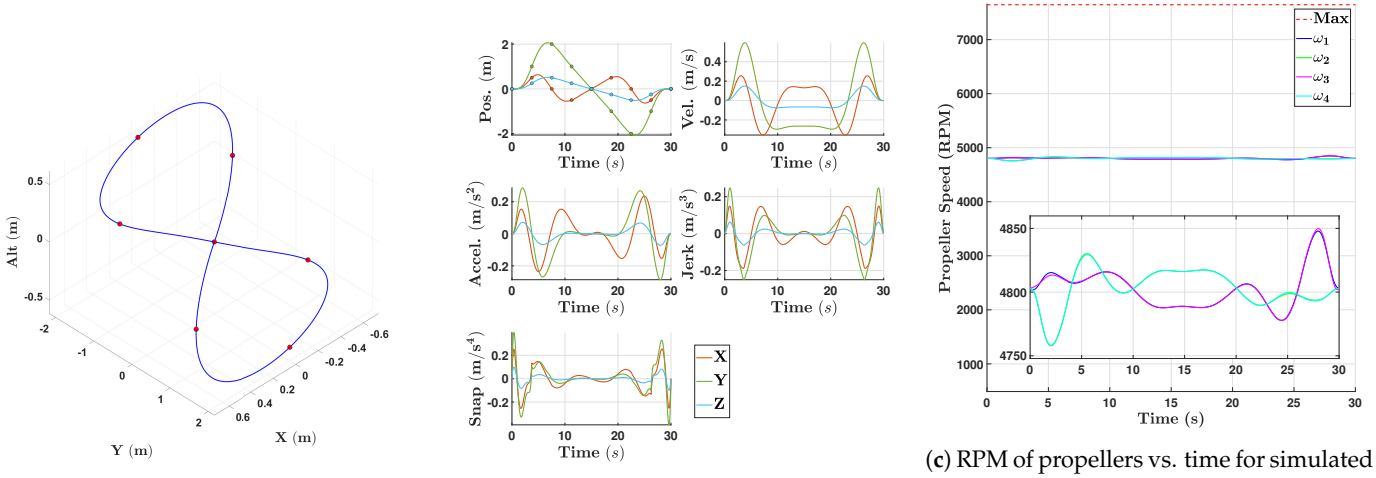
The results in Figures 11 and 12 show that both solutions are closely matching each other except for small differences on the snap level. These differences are then further reflected in the motor RPM profiles shown in Figures 11c and 12c. Minor numerical noise is visible across the polynomial trajectory but is missing for the FFS. The irregular shape of the two solutions in Figures 11a and 12a is due to the evenly discretized time, and can be fixed when the time-allocation problem is solved. The power consumption is in favor of the polynomial solution that obtained a value of 1435.62 W, while the FFS solution consumes 1437.65 W of power. The minimum-snap cost values are 1840.02 for polynomial and 2004.76 for FFS methods.

3.1.5. "Eight" Trajectory

This trajectory traces an "8"-like figure in an inclined plane. The plane itself is tilted so that the altitude is not constant. The heading angle is also set to approximately trace the center of the figure. This way, all four dimensions are engaged during the maneuver

and require trajectory optimization. This high dimensionality and complex 3D shape is intended to stress both the trajectory generation methods and the control system. Due to the inherent symmetry of the trajectory, even discretization (for $t_f = 30$ s) is close to optimal, and this feature was exploited to verify time allocation.

Even for this relatively complicated motion, the two parameterizations are closely matching (Figures 13 and 14). The total time allocated for this fixed-time solution was $T = 30$ s, which results in almost flat motor RPM profiles for the two solutions. No noticeable differences are present either in shape, path components, derivatives, or motor profiles. The power consumption is also in favor of the polynomial solution that obtained a value of 4200.38 W, while the FFS solution consumes 4200.42 W of power. The minimum-snap cost values are 1.05 for polynomial and 1.15 for FFS methods.

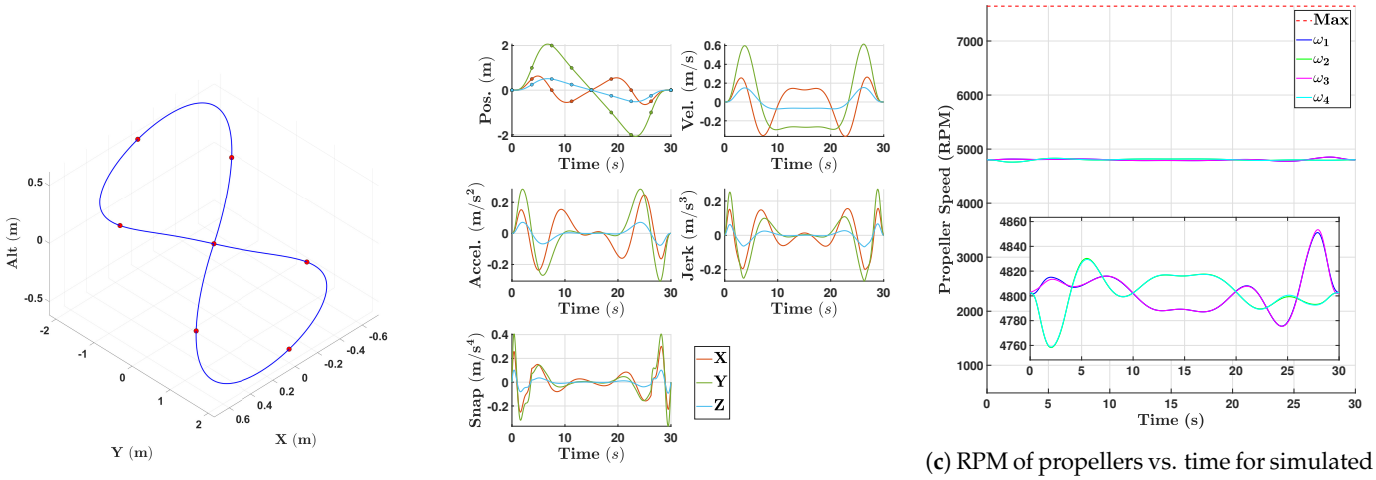


(**a**) 3D view of the path.

(**b**) Single axis information.

(**c**) RPM of propellers vs. time for simulated flight.

Figure 13. Simulated results for the "Eight" trajectory with the polynomial parameterization.



(**a**) 3D view of the path.

(**b**) Single axis information.

(**c**) RPM of propellers vs. time for simulated flight.

Figure 14. Simulated results for the "Eight" trajectory with the FFS parameterization.

3.1.6. Summary of the Fixed-Time Trajectories

All fixed-time solution metrics are summarized in Table 1. The solutions for the two different parameterizations are closely following each other and are mostly identical. As was expected, computational times are on the same order of magnitude, but are slower for the FFS parameterization. The power consumption is primarily dominated by the total time of flight, since most trajectories require only minor changes in motor speeds around the hover thrust. Although the difference is very small, FFS leads to a slightly higher

power consumption for all the trajectories considered. The minimum-snap cost values are following the same trends as power consummations, but the relative scale is unrelated.

3.2. Time-Allocated Solutions

This section continues the discussion of the results for the time-allocation formulation presented in Section 2.2. In addition to the parameters introduced earlier, the solutions are compared based on the time allocation weight coefficient, k_T, and a number of fixed-time iterations, n_{iter}, it took to converge to the optimal solution. The results are summarized in Table 2. All the inputs (trajectory waypoints and constraints) are the same as for the fixed-time solutions, the time allocation vector with evenly discretized time is used as an initial guess for the gradient-descent algorithm that attempts to find the optimal time allocation given trajectory-specific weight k_T. This weight is arbitrary and is determined by trial and error to match the desired global time $T(s)$ for each trajectory and parameterization combination.

Additional rounds of trajectory generation have been executed to study the effect of the time allocation weight, k_T. The choice of this gain determines the global time for executing the entire trajectory, but the values of the minimum-snap cost vary between the two parameterizations. For this reason, the same time weight k_T, with all other inputs being identical, leads to two different solutions. All graphical results included in this section are using different k_T values such that the global time for the two parameterizations match for each trajectory. Since it may also be interesting to compare the two methods when all inputs, including k_T, are identical, these results have also been included in Table 2.

As it was shown in [39], and explained in Section 2.2, all the time-optimal solutions for the same set of inputs, other than k_T, lead to the same path. By varying time allocation gain k_T, the global time can be adjusted, but the shape of the solution remains identical. The position components, when plotted with regard to time, pass through the exact same points for any reasonable k_T values. These points only shift in time, occurring either earlier or later in time. Their derivatives, on the identical plot, not only shift in time but also in their respective magnitudes. This adjustment, if an optimal solution was found, maintains the same position along the trajectory, but increases or decreases the respective velocity, acceleration, jerk, and snap along the path. For this reason, we can expect to always obtain the same shape of the time-allocation solution while adjusting time allocation gain k_T and keeping all other inputs the same. As we will show later, this property holds between the two different parameterizations. However, the errors due to numerical rounding, discussed in Section 2.1.2, can be more severe for one of the two parameterizations, which can lead to some differences between the two solutions. These differences often increase as the order of the derivatives grows. In fact, first symptoms usually occur on the level of snap; then, by constraining the trajectory even more (by increasing the time allocation weight k_T), undesirable large-frequency oscillations propagate to lower derivatives, up until the solution completely breaks down. The invariance of the trajectories to time allocation is quite an interesting characteristic and can be exploited. This means that the feasibility of the trajectories with regard to the collision-avoidance constraint takes precedence over the time-allocation problem and the first step (in any trajectory optimization) can be focused on the collision-avoidance step. More specifically, the time-allocation gain, k_T, directly affects the total time allocated for the trajectory and, therefore, directly affects how demanding the trajectory is to execute. Ref. [42] provides an insightful discussion of the connection between the calculus of variations in kinematics with the corresponding dynamics of the physical vehicle.

Table 2. Summary of the time-allocation solutions.

Name	Method	n_{dt}	n_{dim}	n_{int}	n_{coefs}	T (s)	T_{solve} (ms)	J	P (W)	k_T	n_{iter}
Simple	Polys	4	1	1	10	3.00	5.05	300.21	425.59	700	5
	FFS	4	1	1	10	3.10	9.09	310.99	439.68	700	6
	FFS	4	1	1	10	3.00	13.59	398.75	426.35	930	5
3 Blocks	Polys	4	3	4	120	10.00	305.00	8.79	1403.42	6.15	34
	FFS	4	3	4	120	10.10	252.69	8.88	1417.40	6.15	22
	FFS	4	3	4	120	10.00	285.33	9.53	1403.52	6.67	23
Square	Polys	4	2	8	160	10.00	423.99	200.02	1411.97	140	35
	FFS	4	2	8	160	10.04	609.74	200.87	1417.61	140	32
	FFS	4	2	8	160	10.00	563.42	205.89	1412.14	144	29
Circle	Polys	4	2	7	140	10.00	306.61	54.03	1420.25	37.8	30
	FFS	4	2	7	140	10.07	528.96	54.38	1430.07	37.8	32
	FFS	4	2	7	140	10.00	469.94	56.89	1420.67	39.8	27
Eight	Polys	4	4	9	360	12.00	515.74	85.72	1692.22	50	22
	FFS	4	4	9	360	12.05	895.02	86.11	1699.25	50	24
	FFS	4	4	9	360	12.00	892.07	88.81	1665.06	51.8	23

3.2.1. "Simple" and "3 Blocks" Trajectories

The time-allocation problem for the "Simple" trajectory is trivial, since there exists only one interval and the only optimization that occurs is matching the minimum-snap fixed-time cost with the time weight k_T. Since the time factor k_T was chosen such that it closely matches the original global time of three seconds, this solution is identical to the one presented earlier (fixed-time solution). The time-optimized solution for the "3 Blocks" trajectory, similar to the "Simple" trajectory, is visibly identical since the number of intervals is still low and even discretization is a good approximation of the optimal time allocation for this trajectory. Due to trajectory similarities, only results are summarized in Table 2.

3.2.2. "Square" Trajectory

For this problem, with fixed and even time allocation, we have been able to obtain only two straight edges and some irregularly shaped connections that vaguely resembled the other two edges of a square. Intuitively, it should be fairly obvious that an even distribution of time is very often not optimal for a quadrotor that starts from rest, performs some maneuvers, and returns back to a full halt at the end, especially for a fully symmetric path. A better time allocation should allow more time for the initial and final phases of a flight, where the vehicle must overcome its own inertia and add (or subtract) some velocity. If the global time is maintained constant, some time can be removed from the portions of the trajectory that do not require large changes in the flight path and shifted to more demanding phases. Fortunately, this task can be completely automated, and the time allocation algorithm does just what we expect intuitively.

As can be seen in Figures 9a and 10a, the shape of a square became more clear and regular. Even though the two edges that used to be straight, which corresponded to the initial and final segments of the trajectory, became more curved, the other two straightened out. The two parameterizations were able to achieve a close-to-symmetric square shape, which is also visibly identical for the two. The effects of the time allocation are clearly visible when comparing the acceleration plots of the fixed-time solutions (Figure 7b or Figure 8b) with the new optimal time-allocated solutions shown in Figures 9b and 10b. Large initial and final acceleration spikes that were required for the fixed-time solutions are distributed throughout the entire trajectory. This has lowered the requirements for the control system and allowed for smoother motor RPM profiles (Figures 9c and 10c). Even though the global time was maintained the same, proper time allocation not only restored

a more appealing shape but also made it possible to reduce the complexity of executing the maneuver. As a general trend, the differences between solutions obtained using different parameterizations are very similar. Very minor distinctions exist on the order of snap, and the FFS solution seems to produce smoother motor RPM profiles.

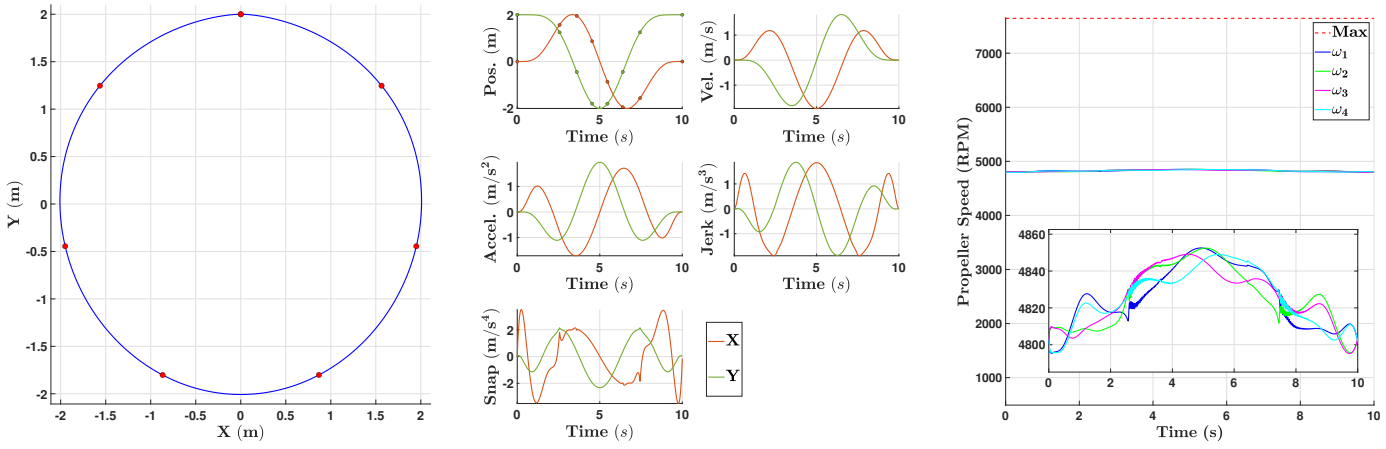
3.2.3. "Circle" Trajectory

Similar to the "Square" trajectory, the circular path was fully recovered when proper time allocation was applied (Figures 15a and 16a). As with the square trajectory, the recovered shapes for both parameterizations are visually identical, and the demands along the trajectory were reduced compared to the fixed-time solutions (see Figures 11 and 12).

Although the overall performance of the two parameterizations remains similar, key distinctions between them become more apparent in this trajectory. A comparison of the snap profiles (Figures 15b and 16b) reveals differing patterns of imperfections. For the polynomial solution, visible spikes persist at the joints between the first and second intervals and between the last and one-before-last intervals. In contrast, the FFS solution largely avoids these spikes or reduces them significantly. However, the FFS begins to encounter minor oscillations along the intermediate intervals, resembling numerical noise.

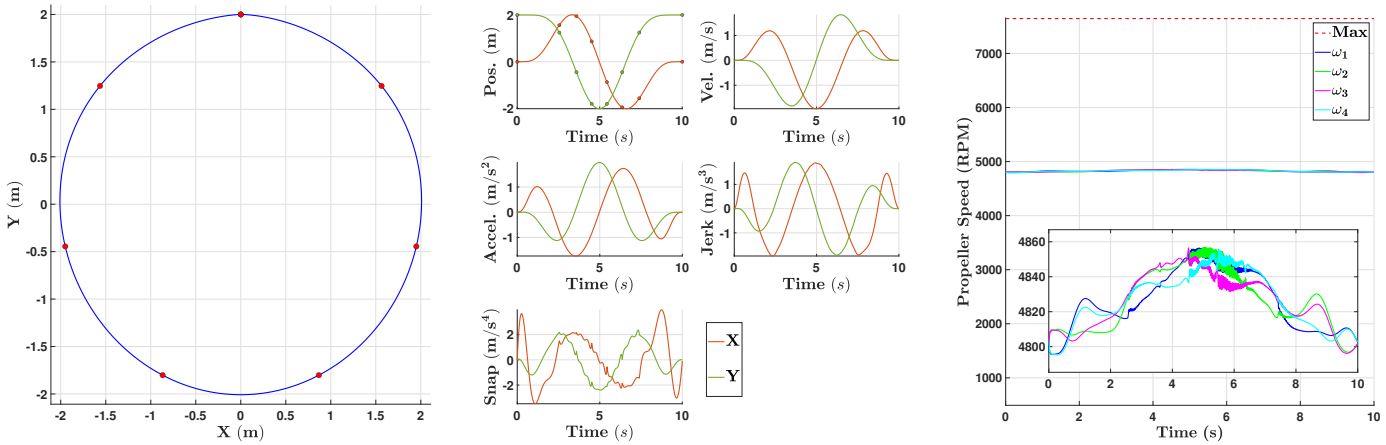
These high-frequency oscillations, although minor relative to the global motor profile scale, are amplified by the control system and are particularly noticeable in the motor RPM profiles of the FFS solution (Figure 16c). Specifically, these oscillations are concentrated between 4.5 and 6.5 s, while similar but less pronounced oscillations are observed for the polynomial solution at approximately 3 and 7.5 s. This behavior is attributed to slight oscillations in the snap of the reference trajectory, which propagate through the control system.

It is important to note that this coupling between reference snap oscillations and control system performance arises from the idealized nature of the simulation. With no actuator delays or disturbances, the simulation exaggerates the sensitivity to high-order derivatives. A more robust controller, as required for real-world scenarios, would reject such oscillations effectively. The controller gains were intentionally selected to be sensitive in order to highlight that even the highest derivative of position (snap) can influence the control system's performance. Additionally, these oscillations become more pronounced when the time of flight is further reduced, as tighter time constraints effectively compress the trajectory along the time axis. One potential solution to mitigate these effects is to use Chebyshev or other non-uniform waypoint distributions, which can help minimize artifacts resembling the Runge phenomenon.



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 15. Simulated results for the time-allocated "Circle" trajectory with polynomial parameterization.



(**a**) 2D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 16. Simulated results for the time-allocated "Circle" trajectory with FFS parameterization.

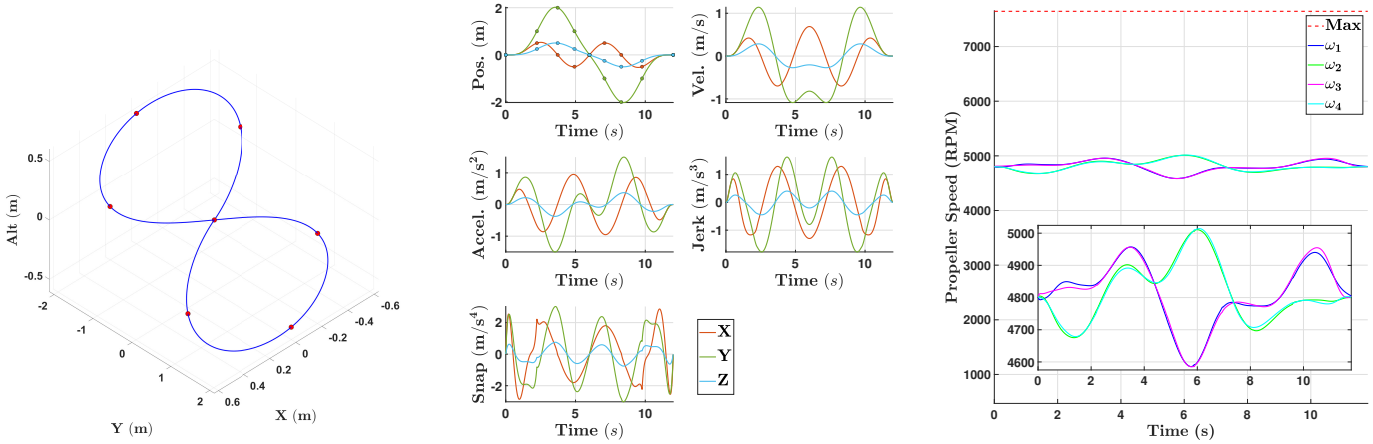
3.2.4. "Eight" Trajectory

In this case, the shape became more circularized, perfectly tracing the "Eight" (Figures 17a and 18a). Time factors were also chosen such that the global time for the flight is decreased by more than half of the original guess. For this reason, all the ranges for derivatives and motor speeds became somewhat higher than for the fixed-time solution. However, it should be obvious that, without a proper time allocation, performing the exact same maneuver, but more than twice as fast, would significantly increase all the requirements. No notable or new differences between the two solutions are observed neither on the level of derivatives (Figures 17b and 18b) nor at the level of control system outputs (Figures 17c and 18c).

One important point worth mentioning is that, even though we have seen some minor issues on the snap level for both parameterizations, they do not appear to show a distinctive pattern or correlation. In summary, it should be clear that the motion planning of quadrotors can be performed using either of the parameterizations and the FFS shows close convergence to the polynomial solution even when time allocation is performed. In addition to the graphical examples discussed earlier in this section, the solutions for the equivalent time factor k_T are summarized in Table 2. Although the shapes of all the solutions for any k_T values remained the same, these results were included to provide a one-to-one comparison of the two parameterizations when every single input is identical. As it was shown by the fixed-time solution, the minimum-snap cost of the FFS parameterization is higher than the polynomial cost. For this reason, the FFS method requires a higher gain k_T to match the same global time. Thus, the total flight time should always be higher for the FFS solution than for an equivalent polynomial solution.

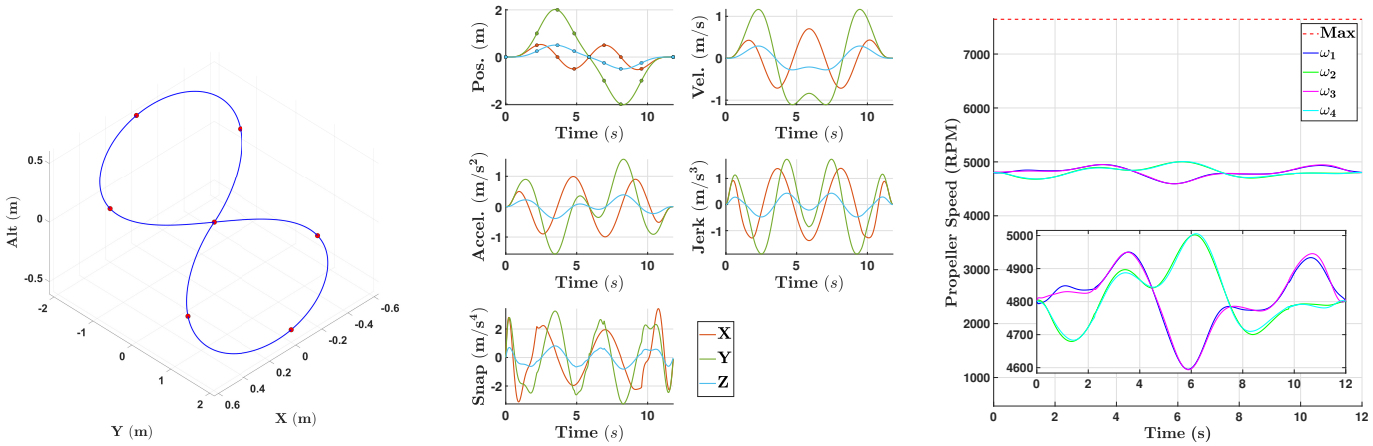
Although the theoretical assumptions and fixed-time solutions show that FFS-based parameterization is slower than the polynomial parameterization, the time-allocation problem may not follow the same trends. In fact, FFS seems to often converge in fewer iterations than an equivalent polynomial method. This is related to the time factor, k_T, but for the combination of inputs used in this work, some solutions have a low enough number of iterations such that the total computational time for the time-allocation problem offsets the longer evaluation times of each of the fixed-time solutions. This is clearly shown for the "3 Blocks" trajectory in Table 2, where FFS was able to achieve an optimal solution faster than the equivalent formulation using the polynomial ("3 Blocks" problem). The fact that the time-allocation problem shows that the FFS parameterization may often converge to the optimal (with a near-equivalent minimum-snap solution in fewer iterations than an identical algorithm with a polynomial formulation) is intriguing. Some features, specific

to the trigonometric functions, allow for controlling the rate of convergence and can be further explored.



(**a**) 3D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 17. Simulated results for the time-allocated "Eight" trajectory with polynomial parameterization.



(**a**) 3D view of the path. (**b**) Single axis information. (**c**) Motor speed.

Figure 18. Simulated results for the time-allocated "Eight" trajectory with FFS parameterization.

3.3. Numerical Results Summary in Practical Context

The numerical results presented in this work demonstrate the exceptional precision and smoothness achievable using the proposed trajectory optimization methods. Both the polynomial- and FFS-based parameterizations operate at a level of detail that exceeds the typical requirements of real-world quadrotor operations.

For instance, simulations show a close-to-hover propeller speed of approximately 4825 RPM for a 450 mm quadrotor, with deviations averaging less than 100 RPM per motor. This corresponds to less than a 1.3% throttle difference on individual motors, highlighting the finely tuned nature of the optimized trajectories. Such minimal deviations are unlikely to have a noticeable impact in real-world scenarios, where factors such as sensor noise, environmental disturbances, and manual control inputs typically dominate. In practical applications, these small control efforts would be nearly indistinguishable from background variability and would not pose a significant challenge to the UAV's control system.

In more demanding scenarios, such as the yaw-intensive "Eight" trajectory requiring two revolutions while maintaining translational motion, differential RPM changes of approximately 400 RPM are observed. These deviations, however, represent synchronized adjustments between motor pairs to meet specific maneuver requirements and are well

within the capabilities of standard quadrotor hardware. This particular scenario was selected for its complexity and successfully validated through experimental testing, further confirming the robustness of the proposed methods.

It is important to note that the optimized trajectories inherently minimize energy consumption by design, as the minimum-snap cost function directly correlates with reduced control effort. Any suboptimalities observed in fixed-time solutions are alleviated when time-allocation optimization is considered. Even without further optimization, the resulting trajectories are well within the agility capabilities of modern quadrotors, and the overall energy expenditure is primarily influenced by hover time rather than maneuvering effort.

In practical terms, the fine differences observed in control effort between the methods are unlikely to translate into measurable performance gains during real-world deployment. Sensor noise, environmental disturbances, and the inherent imprecision of real-world conditions will overshadow these theoretical differences. Additionally, the smoothness of the reference trajectories generated by the proposed methods is expected to reduce unnecessary wear and tear on motors and electronic speed controllers (ESCs), potentially improving hardware longevity under aggressive maneuvers.

In summary, while the numerical results highlight the fine-grained performance of the optimization methods, the real-world implications affirm that these trajectories are robust, efficient, and practical for UAV operations. Nitpicking minute differences in theoretical results is less relevant when the real-world performance is governed by external factors that dwarf these small variations.

4. Experimental Results

We consider a relatively small flying arena, and we are limited to trajectories confined to a medium-size room (about $4 \times 6 \times 3$ m tall). Five different trajectories are considered to compare the usefulness of the trajectories in practice. All time-optimal trajectories discussed in the previous sections have been successfully flown. Both polynomial and FFS parameterizations have shown identical performance. Theoretical and computational results have shown that the differences between the two methods are negligible, even in an ideal environment. Since there was little to no difference between the two methods, the results are reported only for the "3 Blocks", "Square", and "Eight" trajectories.

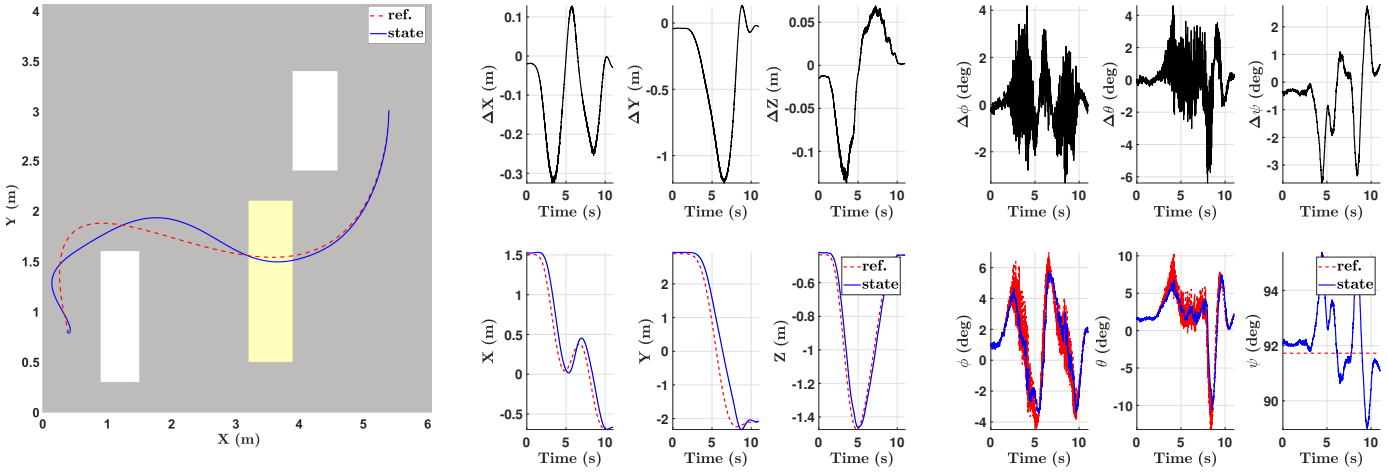
The trajectory optimization algorithm presented in this paper does not depend on any of the vehicle's parameters and the results of the time allocation have not been modified or recomputed onboard in any way; therefore, Table 2 also summarizes experimental results. The only difference would be in the actual power consumed (since the $P(W)$ in Table 2 uses simulation results). Unfortunately, the current hardware framework does not allow for accurate in-flight power consumption estimation. But, with an RPM measurement sensor, it will be possible to have an accurate estimate of the actual power consumption.

4.1. "3 Blocks" Trajectory

Using the optimal solution for the "3 Blocks" trajectory with time allocation, the control system on the quadrotor had to follow the required position references (red) generated using the two parameterizations. The resulting 2D path (blue) is shown in Figures 19a and 20a. Although not as perfect as in the simulator, the two trajectories have been closely tracked and no significant deviation occurred for any of the two methods.

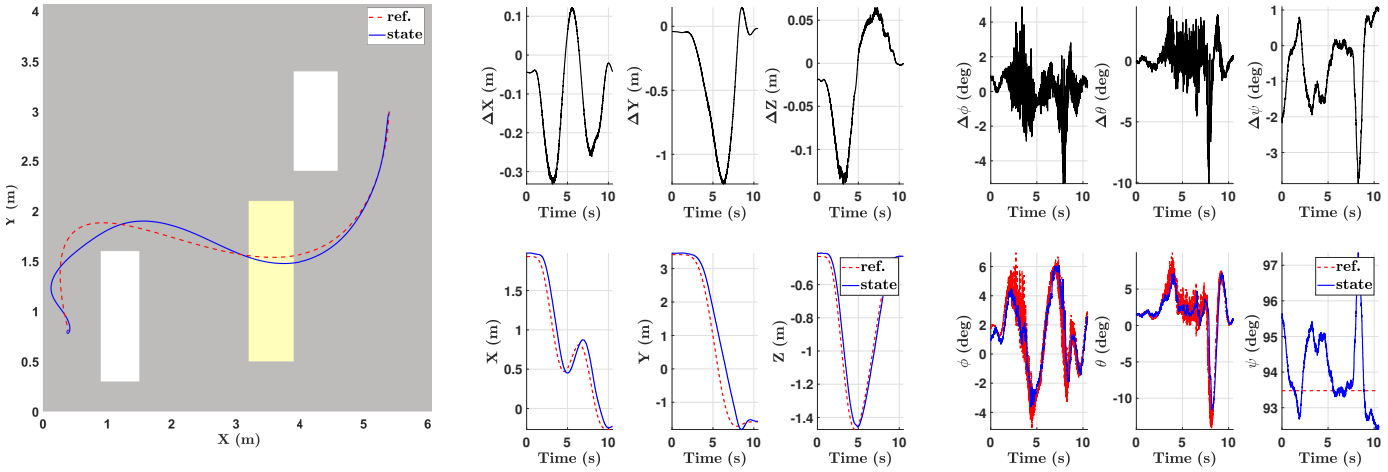
The performance of the position control tracking is shown on Figures 19b and 20b. Figures 19c and 20c show attitude control system tracking for the references generated by the higher-level velocity and position control (details of the flight control system are given in Ref. [62]). It is worth noting that the quadrotor is relatively heavy and more conservative control system gains were chosen. For this reason, the quadrotor noticeably lags behind,

with position-level errors growing in time as the trajectory references move away faster than the quadrotor is capable of catching up. However, the commanded trajectory is still followed with acceptable accuracy.



(**a**) 2D view of the flight path. (**b**) Tracking of position components. (**c**) Tracking of attitude components.

Figure 19. Experimental results for the time-allocated "3 Blocks" trajectory with polynomial parameterization.



(**a**) 2D view of the flight path. (**b**) Tracking of position components. (**c**) Tracking of attitude components.

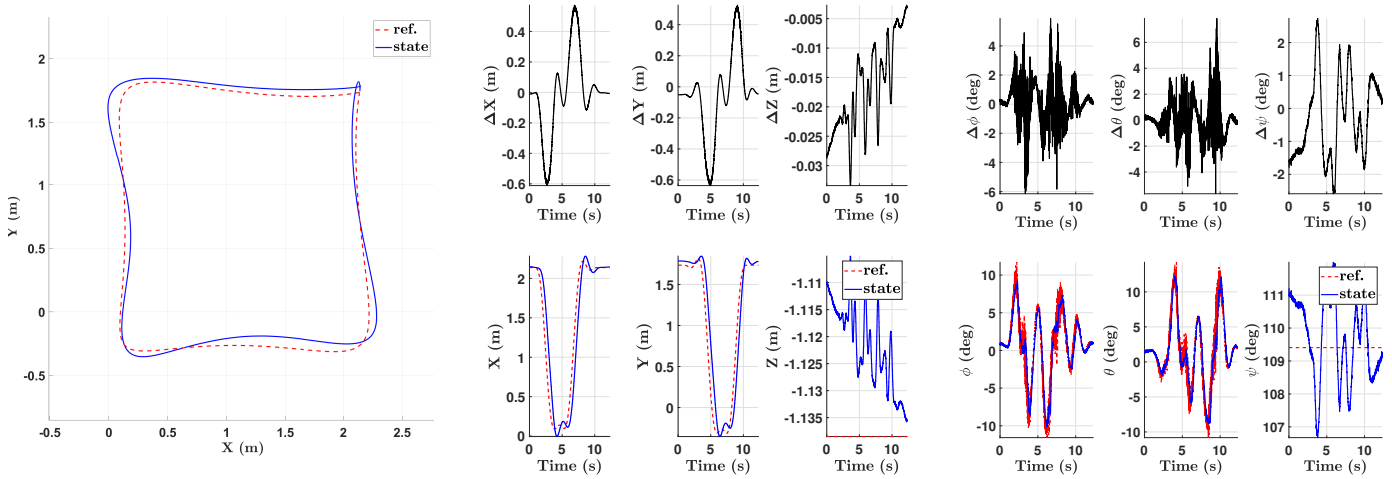
Figure 20. Experimental flight results for the time-allocated "3 Blocks" trajectory with FFS parameterization.

As a side note, the path does not collide at any point with any of the obstacles; in fact, there is enough margin for the quadrotor (in any orientation). Figures 19a and 20a show the same trajectory as in Figures 5a and 6a (but with optimal time allocation, which does not noticeably change the shape for such a short and simple trajectory). Due to the choice of the obstacles, 2D view and 3D view have to be combined to visualize the actual path of the quadrotor.

4.2. "Square" Trajectory

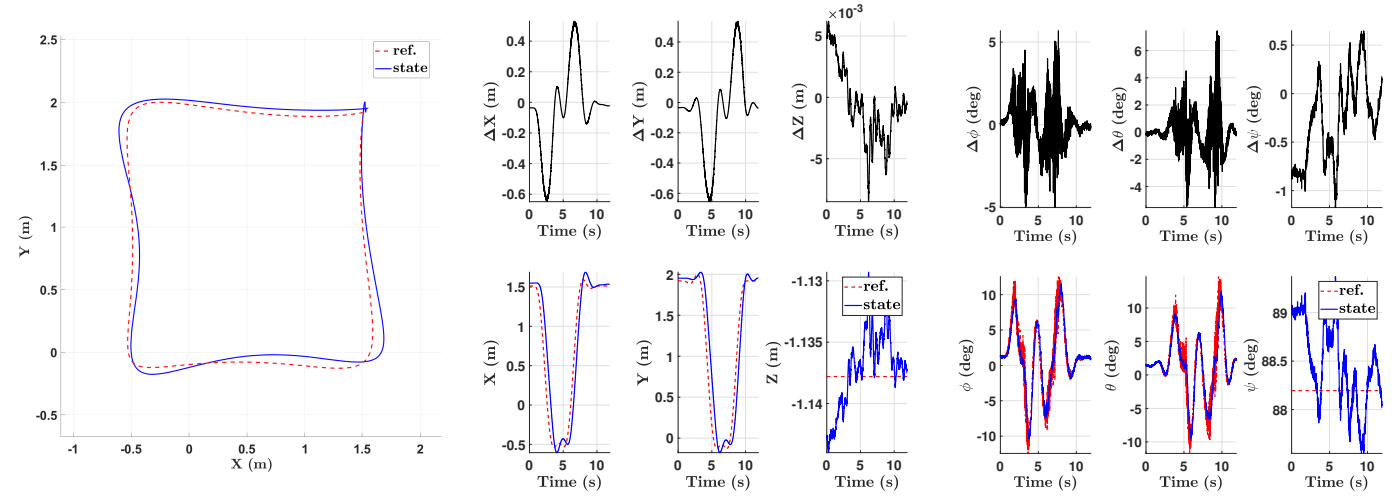
Similarly, taking the optimal solution for the square-shaped trajectory with time allocation, the quadrotor had to follow position-level setpoints. The resulting path is shown in Figures 21a and 22a. Again, the 2D view is shown instead of the 3D view because the motion is completely planar. In fact, the vertical position errors in Figures 21b and 22b clearly show that the altitude is kept within a few millimeters of the initialization vertical

setpoint. The performance of the attitude control tracking is shown in Figures 21c and 22c. Since the implemented control system has a cascaded PID control structure, the references for each control layer are generated by the control layer above, in a cascaded manner. For this reason and due to the significant numerical noise of the estimated states, the control system references for the attitude layer show noticeable bands of noise.



(**a**) 2D view of the flight path. (**b**) Tracking of position components. (**c**) Tracking of attitude components.

Figure 21. Experimental results for the time-allocated "Square" trajectory with polynomial parameterization.



(**a**) 2D view of the flight path. (**b**) Tracking of position components. (**c**) Tracking of attitude components.

Figure 22. Experimental flight results for the time-allocated "Square" trajectory with FFS parameterization.

4.3. Notes on Numerical Convergence

It was previously stated (in Section 2.1.2) that, when higher-order derivatives are considered, Equation (12) can become numerically unstable and lead to inaccurate solutions. Our numerical experiments indicate that both polynomial- and FFS-based parameterizations suffer from this issue; however, the convergence problems arise more quickly for FFS due to its structure ("Sooner" is defined as the derivative order is increased). The current remedy is to limit the order of the derivative considered, which is only a concern when constraints (or cost functions) involve derivatives beyond snap. In the case of minimum-snap optimization (where the highest derivative is of order 4), the convergence instability

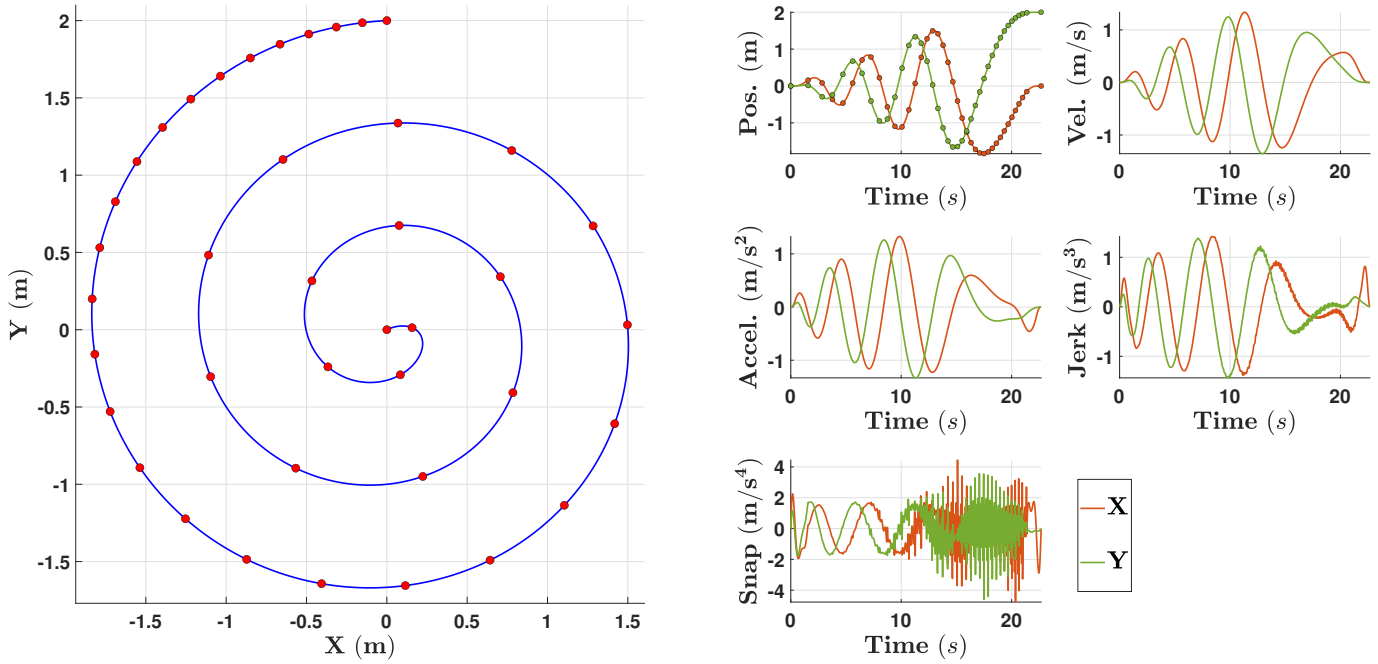
is minimal and mainly mitigated at the waypoint selection stage, before the algorithm described in this paper is executed.

The time scaling factor, t/T, used for both parameterizations, can negatively affect results for trajectories with closely spaced waypoints. Here, "closeness" refers not to the physical distance but to the ratio of distances between each of the points relative to each other. For example, Figure 23 demonstrates this effect by purposefully packing more waypoints than necessary near the end of a spiral trajectory (the start is at the center).

Although the solution may appear feasible in Figure 23a, significant oscillations are present in the fourth and even third derivatives (Figure 23b). If this reference were used, the quadrotor would exhibit similar high-frequency oscillations in its motor RPM profiles. This example highlights the need for pruning or a mesh refinement step when waypoints are selected automatically (rather than being hand-picked) to avoid such extreme cases. In this paper, it is assumed that the waypoints have already been appropriately selected.

Moreover, it is important to note that the value of the time allocation weight coefficient, k_T, does not substantially affect the convergence or solution quality. While this may seem counterintuitive, the value of k_T does not alter the time allocation per segment because the ratio of time per segment to the total flight time remains constant for any converged solution, assuming only k_T is modified. This relationship is explained in Section 2.1.3 and derived in Equation (23). Hence, k_T is not crucial for convergence or solution quality; it is more of a personal preference in formulation. An alternative explanation has been added at the end of Section 2.2.

The convergence and solution quality are primarily determined by the number of waypoints and boundary conditions, as these elements control the time allocation per segment. As mentioned earlier, both parameterizations experience numerical challenges when time distribution is uneven. Despite the scaling and mapping techniques discussed, care must be taken when generating the initial set of waypoints, as improper distribution can cause numerical issues (see Section 4, “Notes on Numerical Convergence”).



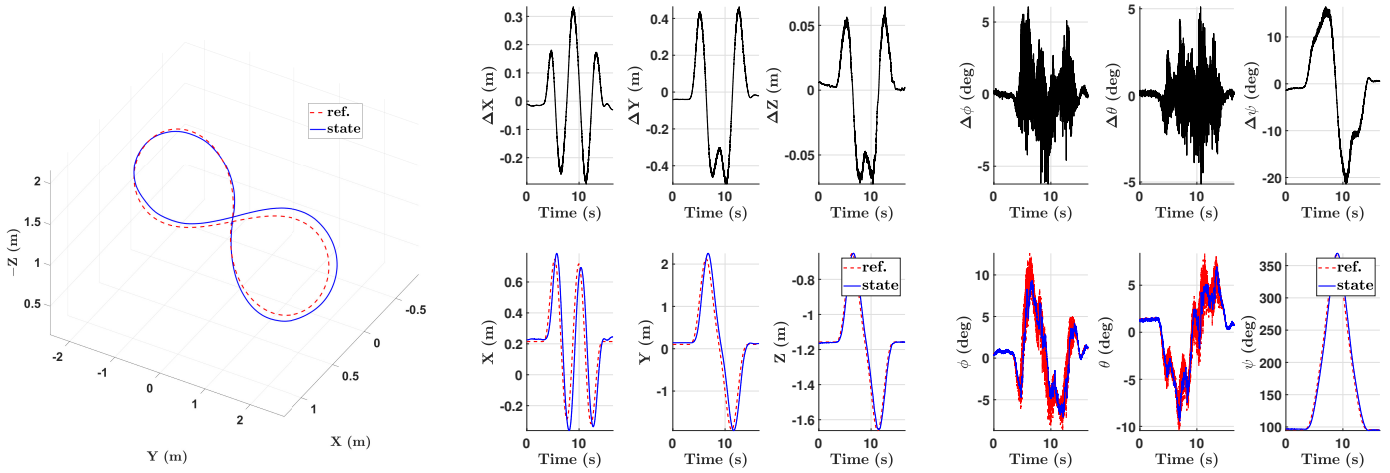
(**a**) 2D view with waypoints highlighted in red.

(**b**) Position and higher-order derivatives.

Figure 23. Trajectory optimization results using FFS method with non-uniformly spaced waypoints.

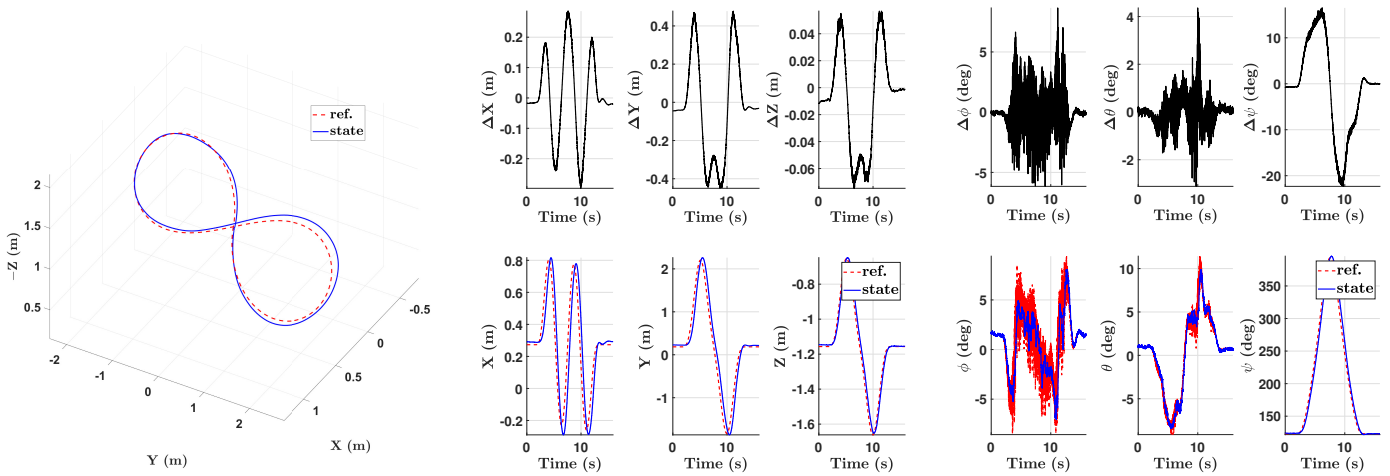
4.4. "Eight" Trajectory

The experimental results for the fast and complex "Eight" trajectory are also as expected (Figures 24 and 25). Although the uncertainties due to the environment allow for the use of experimental results as a validation, they limit the acquisition of any additional insight into the solutions. No unexpected behavior was observed, and it is clear that the two methods generated trajectories that can be followed by a quadrotor in our laboratory.



(**a**) 3D view of the flight path. (**b**) Tracking of position components. (**c**) Tracking of attitude components.

Figure 24. Experimental results for the time-allocated "Eight" trajectory with polynomial parameterization.



(**a**) 3D view of the flight path. (**b**) Tracking of position components. (**c**) Tracking of attitude components.

Figure 25. Experimental flight results for the time-allocated "Eight" trajectory with FFS parameterization.

5. Conclusions

This work developed a quadrotor motion-planning algorithm based on the finite Fourier series (FFS) parameterization for minimum-snap trajectory optimization. By leveraging the inherent mathematical properties of the FFS method, the trajectory optimization problem was reformulated as an unconstrained quadratic programming problem, offering an alternative to the well-established polynomial parameterization.

The comparative analysis demonstrated that the FFS method produces trajectories comparable to those generated by polynomial parameterization across various scenarios, including "simple", "circular", "square", "3 Blocks", and "Eight" trajectories. Experimental results show less than 0.1% relative difference in power consumption between

the two methods, validating the suitability of FFS parameterization for quadrotor motion planning. While FFS parameterization occasionally converged faster for the time-allocation problem under certain conditions, the polynomial parameterization method generally exhibited better computational performance and robustness, making it preferable for most minimum-snap trajectory optimization tasks.

One of the significant contributions of this work is the introduction of an alternative formulation for the time-allocation problem. Our investigation revealed that time allocation can be expressed as a set of optimal ratios for each trajectory segment, which remain invariant under different total flight times, provided the boundary conditions are unchanged. This property allows for post-processing adjustments to the total flight time without the need to resolve the optimization problem, ensuring flexibility and efficiency in practical applications. Additionally, the equivalence of the minimum-snap method to minimum-energy optimization further underscores its utility for power-constrained missions where minimizing time-of-flight and energy consumption is critical.

The practical considerations explored in this study highlight the method's applicability in real-world scenarios. The hardware experiments validated the proposed approach, particularly its effectiveness in achieving smooth trajectory tracking. These results demonstrate the potential for deploying the method in dynamic environments requiring rapid replanning and obstacle avoidance, as evidenced by real-time applications demonstrated in our recent work [64]. The method is also suitable for trajectory tracking using a data-driven $\mathcal{H}_\infty$ controller [65].

While this study focused on an alternative parametrization for the fixed-time optimization framework, future work will explore extending the methodology to address additional constraints and complex dynamic environments. Incorporating aerodynamic effects, such as wind disturbances, can enhance the robustness of trajectory planning. These considerations can be incorporated at the trajectory reconstruction step through differential flatness, enabling accurate modeling of real-world flight dynamics without altering the core optimization routine.

Furthermore, addressing limited battery life within the trajectory optimization framework provides another avenue for development. The existing cost function inherently prioritizes energy efficiency, but integrating explicit constraints on remaining battery life and recharge cycles could improve its utility for extended missions.

The modularity of the proposed framework also makes it a candidate for extension to more complex UAV platforms, such as tilt-wing or tilt-rotor aircraft. These systems require reformulating the dynamic equations and trajectory generation process, and their applicability would depend on proving differential flatness properties for such configurations. However, the current framework's reliance on minimal path constraints and fixed-time formulations enables faster computational performance, a trade-off that may need to be revisited for systems requiring more complex constraints.

In conclusion, this study demonstrates the practical utility and theoretical soundness of the proposed FFS-based method for quadrotor motion planning. By combining rigorous theoretical development with experimental validation, the work provides a robust foundation for future research in energy-efficient, flexible trajectory optimization. Potential extensions to more complex systems and dynamic environments, coupled with further computational improvements, promise to enhance the method's scalability and applicability in diverse UAV missions.

Author Contributions: Conceptualization, E.T.; Data curation, E.T.; Formal analysis, Y.K. and E.T.; Investigation, Y.K.; Methodology, Y.K. and E.T.; Project administration, E.T.; Resources, E.T.; Software, Y.K.; Supervision, E.T.; Validation, Y.K.; Visualization, Y.K.; Writing—original draft, Y.K.; Writing—review & editing, Y.K. and E.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: We have open sourced the autopilot code (developed in the Aero-Astro Computational and Experimental (ACE) Lab) accessible through the following GitHub repository https://github.com/YevheniiKovryzhenko/ACEPilot.git (accessed on 16 January 2025). The original contributions presented in the study are included in the article, and further inquiries can be directed to the corresponding author.

Acknowledgments: The autopilot firmware, *ACEPilot* https://github.com/YevheniiKovryzhenko/ACEPilot.git (accessed on 16 January 2025), which is used in this work, is a heavily modified version of *rc_pilot* https://github.com/StrawsonDesign/rc_pilot (accessed on 16 January 2025) firmware that was originally developed by James Strawson. We thank Ella Atkins for providing us with the *rc_pilot* firmware and Matthew Romano for explaining the firmware. We would also like to thank Behdad Davoudi for his help regarding the overall structure of the *rc_pilot* firmware.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Waypoints for all the cases are summarized. All dimensions are specified in meters or radians. Time per segment T_{i} notation is re-used to indicate a time-stamp of when a specific point i has to be reached. The exact same set of inputs was used to solve the fixed-time problems and are use as initial guesses for the time-allocation problems.

"Simple" trajectory:
$T_1 = 0, X_1 = 1, T_2 = 3, X_2 = 3.$

"Eight" trajectory:
$T_1 = 0, X_1 = 0, Y_1 = 0, Z_1 = 0, \psi_1 = 0.79;$
$T_2 = 3.75, X_2 = 0.5, Y_2 = 1, Z_2 = 0.25, \psi_2 = 1.57;$
$T_3 = 7.5, X_3 = 0, Y_3 = 2, Z_3 = 0.5, \psi_3 = 3.14;$
$T_4 = 11.25, X_4 = -0.5, Y_4 = 1, Z_4 = 0.25, \psi_4 = 4.71;$
$T_5 = 15, X_5 = 0, Y_5 = 0, Z_5 = 0, \psi_5 = 5.5;$
$T_6 = 18.75, X_6 = 0.5, Y_6 = -1, Z_6 = -0.25, \psi_6 = 4.71;$
$T_7 = 22.50, X_7 = 0, Y_7 = -2, Z_7 = -0.50, \psi_7 = 3.14;$
$T_8 = 26.25, X_8 = -0.5, Y_8 = -1, Z_8 = -0.25, \psi_8 = 1.57;$
$T_9 = 30, X_9 = 0, Y_9 = 0, Z_9 = 0, \psi_9 = 0.79.$

"Square" trajectory:
$T_1 = 0, X_1 = 1, Y_1 = 1;$
$T_2 = 1.25, X_2 = 0, Y_2 = 1;$
$T_3 = 2.5, X_3 = -1, Y_3 = 1;$
$T_4 = 3.75, X_4 = -1, Y_4 = 0;$
$T_5 = 5, X_5 = -1, Y_5 = -1;$
$T_6 = 6.25, X_6 = 0, Y_6 = -1;$
$T_7 = 7.5, X_7 = 1, Y_7 = -1;$
$T_8 = 8.75, X_8 = 1, Y_8 = 0;$
$T_9 = 10, X_9 = 1, Y_9 = 1.$

"Circle" trajectory:
$T_1 = 0, X_1 = 0, Y_0 = 2;$
$T_2 = 1.43, X_2 = 1.56, Y_2 = 1.25;$
$T_3 = 2.86, X_3 = 1.95, Y_3 = -0.45;$
$T_4 = 4.29, X_4 = 0.87, Y_4 = -1.8;$
$T_5 = 5.71, X_5 = -0.87, Y_5 = -1.8;$

$T_6 = 7.14, X_6 = -1.95, Y_6 = -0.45;$
$T_7 = 8.57, X_7 = -1.56, Y_7 = 1.25;$
$T_8 = 10, X_8 = 0, Y_8 = 2.$

"3 Blocks" trajectory:
$T_1 = 0, X_1 = 5.4, Y_1 = 3, Z_1 = -0.4;$
$T_2 = 2.25, X_2 = 4.2, Y_2 = 1.6, Z_2 = -1.4;$
$T_3 = 4.5, X_3 = 2.8, Y_3 = 1.6, Z_3 = -1.4;$
$T_4 = 6.75, X_4 = 0.5, Y_4 = 1.8, Z_4 = -0.9;$
$T_5 = 9, X_5 = 0.4, Y_5 = 0.8, Z_5 = -0.4.$

References

1. Goerzen, C.; Kong, Z.; Mettler, B. A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance. *J. Intell. Robot. Syst.* **2010**, *57*, 65–100. [CrossRef]
2. Kawamura, E.; Azimov, D. Integrated Extremal Control and Explicit Guidance for Quadcopters. *J. Intell. Robot. Syst.* **2020**, *100*, 1583–1613. [CrossRef]
3. Paredes, J.; Sharma, P.; Ha, B.; Lanchares, M.; Atkins, E.; Gaskell, P.; Kolmanovsky, I. Development, implementation, and experimental outdoor evaluation of quadcopter controllers for computationally limited embedded systems. *Annu. Rev. Control* **2021**, *52*, 372–389. [CrossRef]
4. Vargas-Ramírez, N.; Paneque-Gálvez, J. The Global Emergence of Community Drones (2012–2017). *Drones* **2019**, *3*, 76. [CrossRef]
5. Trélat, E. Optimal Control and Applications to Aerospace: Some Results and Challenges. *J. Optim. Theory Appl.* **2012**, *154*, 713–758. [CrossRef]
6. Williams, P. Three-Dimensional Aircraft Terrain-Following via Real-Time Optimal Control. *J. Guid. Control Dyn.* **2007**, *30*, 1201–1206. [CrossRef]
7. Quan, L.; Han, L.; Zhou, B.; Shen, S.; Gao, F. Survey of UAV motion planning. *IET Cyber-Syst. Robot.* **2020**, *2*, 14–21. [CrossRef]
8. Karelahti, J.; Virtanen, K.; Öström, J. Automated Generation of Realistic Near-Optimal Aircraft Trajectories. *J. Guid. Control Dyn.* **2008**, *31*, 674–688. [CrossRef]
9. Hii, M.S.Y.; Courtney, P.; Royall, P.G. An Evaluation of the Delivery of Medicines Using Drones. *Drones* **2019**, *3*, 52. [CrossRef]
10. Benarbia, T.; Kyamakya, K. A Literature Review of Drone-Based Package Delivery Logistics Systems and Their Implementation Feasibility. *Sustainability* **2022**, *14*, 360. [CrossRef]
11. Kotlinski, M.; Calkowska, J.K. U-Space and UTM Deployment as an Opportunity for More Complex UAV Operations Including UAV Medical Transport. *J. Intell. Robot. Syst.* **2022**, *106*, 12. [CrossRef] [PubMed]
12. Huang, H.; Savkin, A.V.; Huang, C. A New Parcel Delivery System with Drones and a Public Train. *J. Intell. Robot. Syst.* **2020**, *100*, 1341–1354. [CrossRef]
13. Munawar, H.S.; Ullah, F.; Heravi, A.; Thaheem, M.J.; Maqsoom, A. Inspecting Buildings Using Drones and Computer Vision: A Machine Learning Approach to Detect Cracks and Damages. *Drones* **2022**, *6*, 5. [CrossRef]
14. Detka, J.; Coyle, H.; Gomez, M.; Gilbert, G.S. A Drone-Powered Deep Learning Methodology for High Precision Remote Sensing in California's Coastal Shrubs. *Drones* **2023**, *7*, 421. [CrossRef]
15. Viegas, C.; Chehreh, B.; Andrade, J.; Lourenço, J. Tethered UAV with Combined Multi-rotor and Water Jet Propulsion for Forest Fire Fighting. *J. Intell. Robot. Syst.* **2022**, *104*, 21. [CrossRef]
16. Amponis, G.; Lagkas, T.; Zevgara, M.; Katsikas, G.; Xirofotos, T.; Moscholios, I.; Sarigiannidis, P. Drones in B5G/6G Networks as Flying Base Stations. *Drones* **2022**, *6*, 39. [CrossRef]
17. LaValle, S.M. *Planning Algorithms*, 1st ed.; Cambridge University Press: Cambridge, UK, 2006. [CrossRef]
18. Siméon, T.; Laumond, J.P.; Nissoux, C. Visibility-based probabilistic roadmaps for motion planning. *Adv. Robot.* **2000**, *14*, 477–493. [CrossRef]
19. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
20. Frazzoli, E.; Dahleh, M.A.; Feron, E. Real-Time Motion Planning for Agile Autonomous Vehicles. *J. Guid. Control Dyn.* **2002**, *25*, 116–129. [CrossRef]
21. Zinage, V.V.; Ghosh, S. Generalized Shape Expansion-Based Motion Planning in Three-Dimensional Obstacle-Cluttered Environment. *J. Guid. Control Dyn.* **2020**, *43*, 1781–1791. [CrossRef]
22. Liu, H.; Suzuki, S. Model-Free Guidance Method for Drones in Complex Environments Using Direct Policy Exploration and Optimization. *Drones* **2023**, *7*, 514. [CrossRef]
23. Elmokadem, T.; Savkin, A.V. Towards Fully Autonomous UAVs: A Survey. *Sensors* **2021**, *21*, 6223. [CrossRef] [PubMed]

24. Nguyen, K.; Schoedel, S.; Alavilli, A.; Plancher, B.; Manchester, Z. TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 1–7. [CrossRef]
25. Betts, J.T. Survey of Numerical Methods for Trajectory Optimization. *J. Guid. Control Dyn.* **1998**, *21*, 193–207. [CrossRef]
26. Faiz, N.; Agrawal, S.K.; Murray, R.M. Trajectory Planning of Differentially Flat Systems with Dynamics and Inequalities. *J. Guid. Control Dyn.* **2001**, *24*, 219–227. [CrossRef]
27. Faessler, M.; Franchi, A.; Scaramuzza, D. Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories. *IEEE Robot. Autom. Lett.* **2018**, *3*, 620–626. [CrossRef]
28. Mellinger, D.; Kumar, V. Minimum snap trajectory generation and control for quadrotors. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 2520–2525. [CrossRef]
29. Invernizzi, D.; Panza, S.; Lovera, M. Robust Tuning of Geometric Attitude Controllers for Multirotor Unmanned Aerial Vehicles. *J. Guid. Control Dyn.* **2020**, *43*, 1332–1343. [CrossRef]
30. Mueller, M.W.; Hehn, M.; D'Andrea, R. A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation. *IEEE Trans. Robot.* **2015**, *31*, 1294–1310. [CrossRef]
31. Betts, J.T.; Huffman, W.P. Path-constrained trajectory optimization using sparse sequential quadratic programming. *J. Guid. Control Dyn.* **1993**, *16*, 59–68. [CrossRef]
32. Du, R.; Cowlagi, R.V. Interactive Sensing and Planning for a Quadrotor Vehicle in Partially Known Environments. *J. Guid. Control Dyn.* **2019**, *42*, 1601–1611. [CrossRef]
33. Luo, Y.; Lu, J.; Zhang, Y.; Qin, Q.; Liu, Y. 3D JPS Path Optimization Algorithm and Dynamic-Obstacle Avoidance Design Based on Near-Ground Search Drone. *Appl. Sci.* **2022**, *12*, 7333. [CrossRef]
34. Aldao, E.; González-deSantos, L.M.; Michinel, H.; González-Jorge, H. UAV Obstacle Avoidance Algorithm to Navigate in Dynamic Building Environments. *Drones* **2022**, *6*, 16. [CrossRef]
35. Hong, Y.; Kim, S.; Kim, Y.; Cha, J. Quadrotor path planning using A* search algorithm and minimum snap trajectory generation. *ETRI J.* **2021**, *43*, 1013–1023. [CrossRef]
36. Zhou, B.; Gao, F.; Wang, L.; Liu, C.; Shen, S. Robust and Efficient Quadrotor Trajectory Generation for Fast Autonomous Flight. *IEEE Robot. Autom. Lett.* **2019**, *4*, 3529–3536. [CrossRef]
37. Robinson, D.R.; Mar, R.T.; Estabridis, K.; Hewer, G. An Efficient Algorithm for Optimal Trajectory Generation for Heterogeneous Multi-Agent Systems in Non-Convex Environments. *IEEE Robot. Autom. Lett.* **2018**, *3*, 1215–1222. [CrossRef]
38. Lee, E.M.; Choi, J.; Lim, H.; Myung, H. REAL: Rapid Exploration with Active Loop-Closing toward Large-Scale 3D Mapping using UAVs. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; pp. 4194–4198. [CrossRef]
39. Richter, C.; Bry, A.; Roy, N. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments. *Robot. Res.* **2016**, *114*, 649–666. [CrossRef]
40. Bry, A.; Richter, C.; Bachrach, A.; Roy, N. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *Int. J. Robot. Res.* **2015**, *34*, 969–1002. [CrossRef]
41. Sabetghadam, B.; Cunha, R.; Pascoal, A. A Distributed Algorithm for Real-Time Multi-Drone Collision-Free Trajectory Replanning. *Sensors* **2022**, *22*, 1855. [CrossRef]
42. Yakimenko, O.A. Direct Method for Rapid Prototyping of Near-Optimal Aircraft Trajectories. *J. Guid. Control Dyn.* **2000**, *23*, 865–875. [CrossRef]
43. Kreciglowa, N.; Karydis, K.; Kumar, V. Energy efficiency of trajectory generation methods for stop-and-go aerial robot navigation. In Proceedings of the 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 13–16 June 2017; pp. 656–662. [CrossRef]
44. Hehn, M.; D'Andrea, R. Real-Time Trajectory Generation for Quadrocopters. *IEEE Trans. Robot.* **2015**, *31*, 877–892. [CrossRef]
45. Almeida, M.M.D.; Moghe, R.; Akella, M. Real-Time Minimum Snap Trajectory Generation for Quadcopters: Algorithm Speed-up Through Machine Learning. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 683–689. [CrossRef]
46. Burke, D.; Chapman, A.; Shames, I. Generating Minimum-Snap Quadrotor Trajectories Really Fast. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October–24 January 2020; pp. 1487–1492. [CrossRef]
47. Wang, Z.; Ye, H.; Xu, C.; Gao, F. Generating Large-Scale Trajectories Efficiently using Double Descriptions of Polynomials. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; pp. 7436–7442, ISSN 2577-087X. [CrossRef]
48. Paris, A.; Lopez, B.T.; How, J.P. Dynamic Landing of an Autonomous Quadrotor on a Moving Platform in Turbulent Wind Conditions. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 9577–9583. [CrossRef]

49. Shi, D.; Shen, J.; Gao, M.; Yang, X. A Multi-Waypoint Motion Planning Framework for Quadrotor Drones in Cluttered Environments. *Drones* **2024**, *8*, 414. [CrossRef]
50. Penicka, R.; Scaramuzza, D. Minimum-Time Quadrotor Waypoint Flight in Cluttered Environments. *IEEE Robot. Autom. Lett.* **2022**, *7*, 5719–5726. [CrossRef]
51. Foehn, P.; Romero, A.; Scaramuzza, D. Time-optimal planning for quadrotor waypoint flight. *Sci. Robot.* **2021**, *6*, eabh1221. [CrossRef] [PubMed]
52. Liu, S.; Lin, Z.; Wang, Y.; Huang, W.; Yan, B.; Li, Y. Three-body cooperative active defense guidance law with overload constraints: A small speed ratio perspective. *Chin. J. Aeronaut.* **2025**, *38*, 103171. [CrossRef]
53. Taheri, E.; Abdelkhalik, O. Shape Based Approximation of Constrained Low-Thrust Space Trajectories using Fourier Series. *J. Spacecr. Rocket.* **2012**, *49*, 535–546. [CrossRef]
54. Taheri, E.; Abdelkhalik, O. Fast Initial Trajectory Design for Low-Thrust Restricted-Three-Body Problems. *J. Guid. Control Dyn.* **2015**, *38*, 2146–2160. [CrossRef]
55. Taheri, E.; Abdelkhalik, O. Initial three-dimensional low-thrust trajectory design. *Adv. Space Res.* **2016**, *57*, 889–903. [CrossRef]
56. Taheri, E.; Kolmanovsky, I.; Atkins, E. Shaping low-thrust trajectories with thrust-handling feature. *Adv. Space Res.* **2018**, *61*, 879–890. [CrossRef]
57. Huo, M.; Zhang, G.; Qi, N.; Liu, Y.; Shi, X. Initial Trajectory Design of Electric Solar Wind Sail Based on Finite Fourier Series Shape-Based Method. *IEEE Trans. Aerosp. Electron. Syst.* **2019**, *55*, 3674–3683. [CrossRef]
58. Fan, Z.; Huo, M.; Qi, N.; Xu, Y.; Song, Z. Fast preliminary design of low-thrust trajectories for multi-asteroid exploration. *Aerosp. Sci. Technol.* **2019**, *93*, 105295. [CrossRef]
59. Zhou, W.; Li, H.; Wang, H.; Ding, R. Low-Thrust Trajectory Design Using Finite Fourier Series Approximation of Pseudoequinoctial Elements. *Int. J. Aerosp. Eng.* **2019**, *2019*, 1–18. [CrossRef]
60. Caruso, A.; Bassetto, M.; Mengali, G.; Quarta, A.A. Optimal solar sail trajectory approximation with finite Fourier series. *Adv. Space Res.* **2021**, *67*, 2834–2843. [CrossRef]
61. Nurre, N.P.; Taheri, E. Multiple gravity-assist low-thrust trajectory design using finite Fourier series. In Proceedings of the AIAA/AAS Astrodynamics Specialist Conference, South Lake Tahoe, CA, USA, 9–13 August 2020; AAS 20-671, p. 21.
62. Kovryzhenko, Y.; Taheri, E. Comparison of minimum-snap and finite fourier series methods for multi-copter motion planning. In Proceedings of the AIAA SCITECH 2022 Forum, San Diego, CA, USA & Virtual, 3–7 January 2022. [CrossRef]
63. Kovryzhenko, Y. Application of the Finite Fourier Series for Smooth Motion Planning of Quadrotors. Master's Thesis, Auburn University, Auburn, AL, USA, 2023.
64. Kovryzhenko, Y.; Li, N.; Taheri, E. A Control System Design and Implementation for Autonomous Quadrotors with Real-Time Re-Planning Capability. *Robotics* **2024**, *13*, 136. [CrossRef]
65. Kovryzhenko, Y.; Li, N.; Taheri, E. Implementation of a Data-Driven Control Method for Unmanned Aerial Vehicles. In Proceedings of the AIAA Aviation Forum and ASCEND, Las Vegas, NV, USA, 29 July–2 August 2024; American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2024. [CrossRef]
66. Levine, J. *Analysis and Control of Nonlinear Systems: A Flatness-Based Approach*; Springer: Berlin/Heidelberg, Germany, 2009. [CrossRef]
67. Kyriakopoulos, K.; Saridis, G. Minimum jerk path generation. In Proceedings of the 1988 IEEE International Conference on Robotics and Automation, Philadelphia, PA, USA, 24–29 April 1988; Volume 1, pp. 364–369. [CrossRef]
68. Piazzi, A.; Visioli, A. Global minimum-jerk trajectory planning of robot manipulators. *IEEE Trans. Ind. Electron.* **2000**, *47*, 140–149. [CrossRef]