drones

MDPI

Article

Suboptimal Trajectory Planning Technique in Real UAV Scenarios with Partial Knowledge of the Environment

Matilde Gelli †, Luca Bigazzi †, Enrico Boni * and Michele Basso

Department of Information Engineering, University of Florence, Via Santa Marta 3, 50139 Firenze, Italy; matilde.gelli@unifi.it (M.G.); luca.bigazzi@unifi.it (L.B.); michele.basso@unifi.it (M.B.)

* Correspondence: enrico.boni@unifi.it

† These authors contributed equally to this work.

Abstract: In recent years, the issue of trajectory planning for autonomous unmanned aerial vehicles (UAVs) has received significant attention due to the rising demand for these vehicles across various applications. Despite advancements, real-time trajectory planning remains computationally demanding, particularly with the inclusion of 3D localization using computer vision or advanced sensors. Consequently, much of the existing research focuses on semi-autonomous systems, which rely on ground assistance through the use of external sensors (motion capture systems) and remote computing power. This study addresses the challenge by proposing a fully autonomous trajectory planning solution. By introducing a real-time path planning algorithm based on the minimization of the snap, the optimal trajectory is dynamically recalculated as needed. Evaluation of the algorithm's performance is conducted in an unknown real-world scenario, utilizing both simulations and experimental data. The algorithm was implemented in MATLAB and subsequently translated to C++ for onboard execution on the drone.

Keywords: trajectory planning; real time; onboard calculation

Citation: Gelli, M.; Bigazzi, L.; Boni, E.; Basso, M. Suboptimal Trajectory Planning Technique in Real UAV Scenarios with Partial Knowledge of the Environment. *Drones* **2024**, *8*, 211. https://doi.org/10.3390/drones8060211

Academic Editor: Abdessattar Abdelkefi

Received: 30 March 2024
Revised: 12 May 2024
Accepted: 18 May 2024
Published: 21 May 2024

1. Introduction

Trajectory planning for multicopters has been studied extensively in recent years due to the increasing demand of drones in many fields, from entertainment and daily life to sophisticated professional applications [1]. Therefore, it is a very important task to plan a feasible trajectory along which a drone can fly safely while executing a mission. The problem of trajectory planning can be stated as an optimization problem whose solution is a minimum-cost and collision-free path. Common criteria used in the optimization for this purpose are, for example, the minimization of the whole path length [2], the total flight time [3], the flight altitude [4], and the snap (fourth derivative of position). This last approach was first suggested in [5,6] and has become rapidly popular because, by leveraging the differential flatness of a quadrotor [7], a relatively simple and effective planning algorithm that can minimize the maximum force required throughout the whole path is provided.

Path planners can be classified into two main categories in relation to the way the computations are carried out. In particular, they can be divided into *offline* and *online* algorithms [8]. The offline algorithms plan the trajectory before the takeoff; hence, they need information about the environment (such as obstacles and fly-zones) in advance [9,10]. Even though they always guarantee a feasible trajectory, these algorithms find a limited range of application, since they cannot be applied in partially unknown and/or dynamic environments where moving vehicles or other obstacles can cause a change to the planned trajectory. On the other hand, online (real-time) algorithms manage to deal with such types of environments. In fact, by exploiting the incoming information from sensors, they can adapt the trajectory to changes in the surroundings. Clearly, the computational cost

becomes very important for online algorithms, since all the calculations must truly be completed in real time. For this reason, even though approaches such as the probabilistic ones and those based on machine learning have been spreading more and more because of their promising performances in complex environments, the deterministic counterpart is still the best compromise. In fact, the others are more computational demanding, which makes them impractical for real-time applications [11]. This aspect is even more important when considering where these calculations must take place. As a matter of fact, the majority of the current literature addresses the problem of trajectory planning for *semi-autonomous* systems, i.e., all computations are not carried out on the onboard electronics but heavily exploit external workstations and motion capture systems (*mocap*) [12–15]. The latter approach has two main drawbacks: (i) it is not reproducible in an open environment, since the acquisition relies on a system of cameras and sensors properly set to track the motion; and (ii) it does not take into account the real computing power available onboard a vehicle, which is limited. As a result, real-time planners that work fine for mocap systems are still too demanding for autonomous vehicles in which every computation is accomplished using the onboard electronics.

In this paper, we aim to provide a real-time planning algorithm that exploits just the electronics available onboard the drone to compute a feasible trajectory that fulfills all the constraints. Note that, even though the resulting trajectory is obtained by solving an optimization problem, it is a nearly optimal path. Having only partial knowledge of the environment means that the generated trajectory cannot be truly optimal.

The proposed algorithm has a complexity compatible with real-time execution on embedded boards, and it was first written in the MATLAB environment and then exported to C++ using the MATLAB Coder toolbox [16]. Flight tests in an unknown environment were conducted to validate the algorithm. In these tests, the UAV first needed to detect the targets, which were updated regularly, and then compute a feasible and safe trajectory. Furthermore, whenever new targets were seen or their positions corrected, a new trajectory had to be evaluated.

The paper is organized as follows. In Section 2, some details of the hardware platform are given, along with the mathematical formulation of trajectory generation and constraint definitions. Section 3 contains the results obtained from the experiments carried out in a real scenario. These experiments show that the drone can successfully replan the trajectory several times during the mission. A comparison between the computed trajectory and the truly optimal one (obtained with a priori knowledge of all the waypoints) is provided. Finally, in Section 4, a brief recap of the main results and a summary of future developments are presented.

2. Materials and Methods

2.1. Hardware Platform

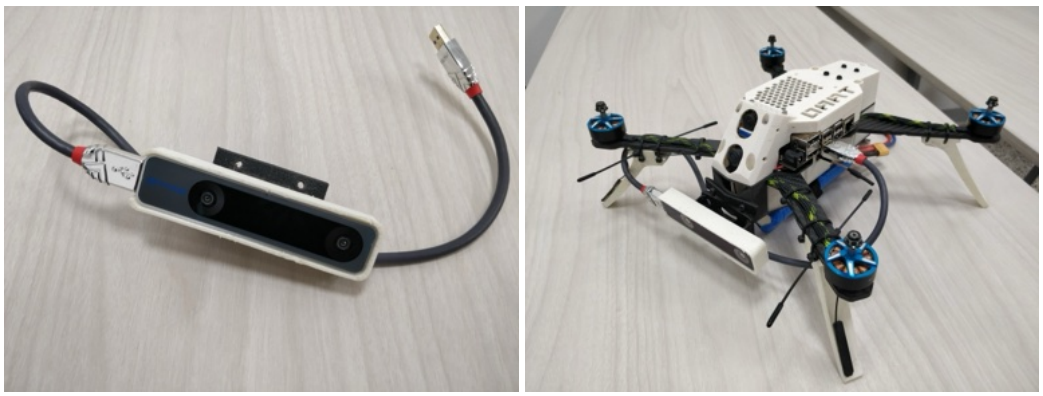
All the experiments carried out in this work were performed on the DART platform [17–20] produced by Florence Robotics; see Figure 1. The system consists of a quadcopter with a multilevel architecture able to process computationally expensive algorithms on board without the use of mocap techniques. In particular, the architecture is organized into two levels: (i) a low-level one that consists of the standard electronics available on a multicopter (flight controller and electronic speed controller); and (ii) a high-level one that is characterized by the NVIDIA Jetson Xavier NX board on which the embedded Linux *Operating System* (OS) runs. The latter is then interfaced via a custom *Application Programming Interface* (API) with the low-level electronics.

Thanks to the DART platform APIs, it is possible to receive real-time telemetry from all sensors on board the drone at a frequency of 200 Hz. Furthermore, it is possible to send autonomous commands from the high-level system to the flight controller (still with a frequency of 200 Hz), replacing those normally issued by the pilot during manual flight.

Figure 1. DART multicopter produced by Florence Robotics [20].

Since all signals passing through the DART platform are managed by specific intermediate hardware, which always ensures the correct fail-safe mode handling, the use of this drone significantly simplifies the experimentation of new algorithms and software for autonomous navigation. In fact, in the event of a failure in the software under test, the intermediate hardware can detect it and return the control to the pilot or, if configured, it automatically manages the switch between available flight modes on the flight controller, which may include auto-landing and return to home.

All of this allowed the abstraction of the software implementation of the trajectory planning. In fact, thanks to this approach, it was possible to test the trajectory planning algorithm in a more realistic way compared to the usual mocap techniques, which can only work in controlled environments. Obviously, since the present work is focused on testing a trajectory optimization algorithm on board an autonomous vehicle in a real scenario, the DART platform was equipped with an advanced optical sensor to accurately detect the targets. In particular, the Intel RealSense T265 camera was used for this purpose. This camera can provide a faithful 3D estimate of position and online speed. In Figure 2b, it is possible to see it mounted on the DART platform.



(**a**) (**b**)

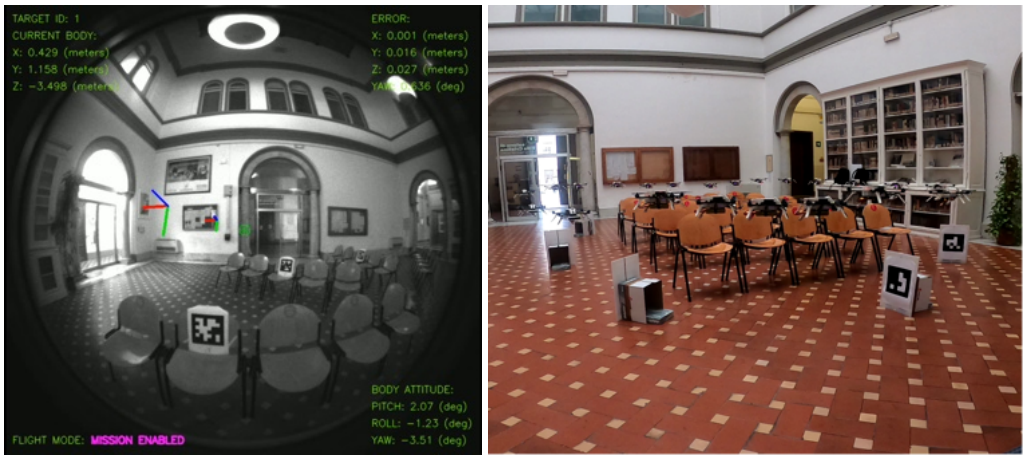
Figure 2. On the left (**a**) is the Intel RealSense T265 camera that was used in the experiments, while on the right (**b**) is the final system with all the sensors on.

2.2. Scenario

To validate the effectiveness of the trajectory planner, which will be discussed in detail in Section 2.3, it is crucial to conduct a flight demonstration serving as a practical assessment for the planner. Specifically, the drone is assigned the task of reaching specific points within the space. Aruco environmental markers [21–23], in conjunction with the associated OpenCV libraries, are used for this purpose.

The markers are strategically positioned in the environment, with their locations intentionally undisclosed to the drone. Leveraging optical flow data from the stereoscopic camera, the drone must adeptly identify the markers and accurately estimate the 3D position of each.

At the mission's initiation, the drone possesses only the identification (ID) codes of the markers, devoid of any knowledge about their physical locations in the environment. Therefore, to fulfill the mission objectives, the drone must first locate the markers and then autonomously generate a trajectory so that it traverses over each marker in the precise order dictated by the provided sequence of IDs. This emphasizes the drone's reliance on its onboard capabilities, particularly in terms of the optical flow processed from the stereoscopic camera, which is needed for detecting and navigating to each marker without any external information about the marker positions. An example of a frame obtained from the on-board optical flow during a test mission is shown in Figure 3. Note that, in this position, just two waypoints are visible and then are passed together with the drone's current position as inputs to the planning module. In fact, the trajectory will be calculated starting from the drone position.



(**a**) (**b**)

Figure 3. On the left (**a**) is a frame processed on board the drone during the execution of an autonomous mission necessary for testing the planning algorithm. On the right (**b**), however, it is possible to observe an image that shows the trajectory traveled by the drone during one of the experiments performed.

For the sake of completeness, we report in Figure 4 the block diagram of the embedded software designed for the execution of the test missions. Within block (a), the module responsible for estimating the position of Aruco markers is depicted. This module takes the optical flow data from the stereoscopic camera as its input. Once new markers are detected, the mission supervisor module (block (b)), housing the optimal trajectory generation algorithm, is activated, and the desired trajectory is provided as a sequence of points.

Subsequently, the computed trajectory is relayed point by point to the module responsible for implementing the control strategy, block (c). This module also takes as input the estimated acceleration, speed, and position data from the Intel T265 stereoscopic camera, which are essential for accurate trajectory tracking.

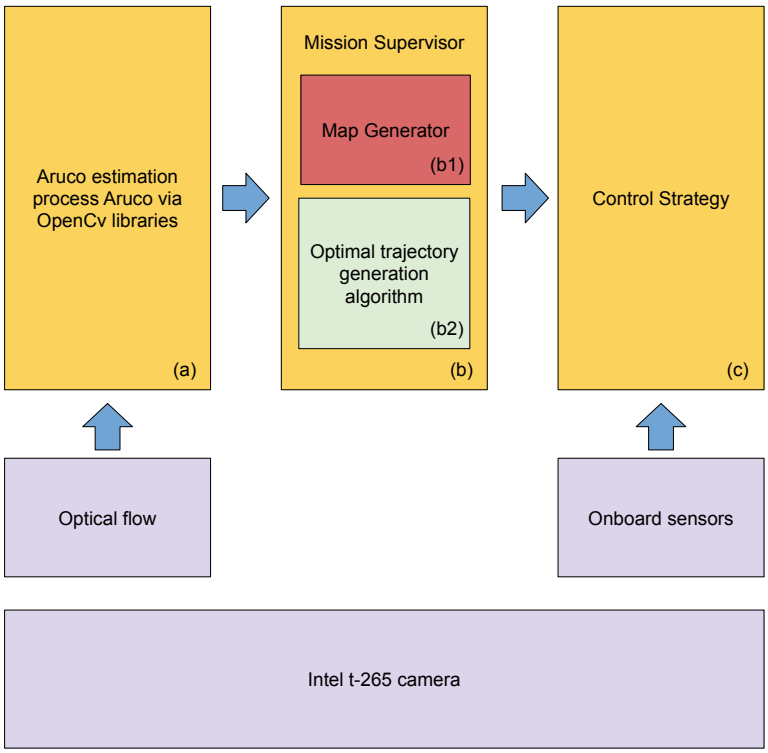


Figure 4. The block diagram representation of the framework developed for testing the trajectory generation algorithm. In yellow, from left to right, we have the marker position estimation module (**a**), followed by the mission supervisor module (**b**), and finally the module (**c**) that implements the control strategy. The mission supervisor module is composed by a Map Generator (**b1**) and the Trajectory planning module (**b2**). The purple blocks represent the onboard sensors.

2.3. Trajectory Generation

Let us state the problem for trajectory planning as follows. Assume we have a set of waypoints that each defines the desired position in the world frame, where the drone must be at specific time instants. Having this set, we want to establish the optimal trajectory with respect to some functional cost, so that all the constraints (such as smoothness and safety passages) are satisfied. Of course, it is important to guarantee the feasibility of the computed trajectory. In other words, during the flight, we must ensure that both velocities and accelerations required for carrying out the mission never exceed the maximum allowable values. If this happens, then we must relax some constraints (like time allocation with which the drone must move from a waypoint to the subsequent one) and recompute the trajectory to have a less aggressive behavior.

In the following, the bold font notation will be used for both vectors and matrices of matrices.

The position of the quadcopter's center of mass between one waypoint and the subsequent one can be described by three independent N-order polynomials (one for each $x(t)$, $y(t)$, and $z(t)$), as follows:

$$P(t) = p_0 + p_1 t + \ldots + p_N t^N = \sum_{n=0}^{N} p_n t^n. \tag{1}$$

Therefore, computing an optimal trajectory means finding the optimal coefficients of these polynomials along the whole path. Consider that here, unlike [6,24], the yaw angle $\psi(t)$ is not optimized along with the trajectory. In fact, thanks to the differential flatness property of a multicopter, it can be demonstrated that, in principle, the vehicle is capable of following a desired trajectory regardless of the heading angle. Therefore, we prefer to

compute it afterwards, once the optimal trajectory is available, so that the drone is always forced to head forward.

Assuming a one-dimensional single-segment trajectory, the objective function is simply

$$J(T) = \int_0^T c_0 P(t)^2 + c_1 P'(t)^2 + c_2 P''(t)^2 + \cdots + c_N P^{(N)}(t)^2 dt, \tag{2}$$

with T being the flight time, while the weights c_r are penalties chosen by the user. However, Equation (2) can be rewritten in matrix fashion, which is more convenient for implementation, as

$$J(T) = \boldsymbol{p}^T Q(T) \boldsymbol{p}, \tag{3}$$

where $\boldsymbol{p}$ is the vector of the N coefficients, and where matrix $Q(T) \in \mathbb{R}^{(N+1)\times(N+1)}$ (see [24]) is defined as

$$Q = \sum_{r=0}^{N} c_r Q_r \qquad \text{for } r = 0, 1, \ldots, N,\ i = 0, 1, \ldots, N,\ l = 0, 1, \ldots, N \tag{4}$$

with

$$Q_r(i,l) = \begin{cases} \prod_{m=0}^{r-1} (i-m)(l-m) \frac{T^{i+l-2r+1}}{i+l-2r+1} & \text{if } i \geq r \wedge l \geq r \\ 0 & \text{if } i < r \vee l < r. \end{cases} \tag{5}$$

As stated previously, penalties c_r can be arbitrarily chosen; nevertheless, in [25], it is proved that the minimization with respect to the snap ($P^{(4)}(t)$) leads to minor stress on the rotors. Therefore, in the continuation, we will assume $c_4 = 1$, while all the other weights will be set to zero.

Given Equations (2)–(5), we can easily extend the problem from the case of a one-dimensional single-segment to a three-dimensional multi-segment path. In this case, the joint functional cost becomes

$$\boldsymbol{J_{joint}}(\boldsymbol{T}) = \begin{bmatrix} \boldsymbol{p_1} \\ \vdots \\ \boldsymbol{p_M} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{Q_1(T_1)} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{Q_M(T_M)} \end{bmatrix} \begin{bmatrix} \boldsymbol{p_1} \\ \vdots \\ \boldsymbol{p_M} \end{bmatrix}. \tag{6}$$

In Equation (6), every T_i is the flight time for moving from the i-th waypoint to the $(i+1)$-th waypoint, and $\boldsymbol{p}$ is the vector of the (unknown) coefficients of each polynomial composing the trajectory, whose structure is as follows:

$$\boldsymbol{p} = \left[[\boldsymbol{p_{x1}}, \boldsymbol{p_{y1}}, \boldsymbol{p_{z1}}]^T, \ldots, [\boldsymbol{p_{xi}}, \boldsymbol{p_{yi}}, \boldsymbol{p_{zi}}]^T, \ldots, [\boldsymbol{p_{xM}}, \boldsymbol{p_{yM}}, \boldsymbol{p_{zM}}]^T \right]^T. \tag{7}$$

Again, the three states $x(t), y(t)$, and $z(t)$ are actually independent, and thus three independent cost functions can be written and solved separately [6]. However, we prefer to use a joint cost function since requirements on passages through oriented windows couple together more state variables [24]. Therefore, the current formulation can be more easily extended in the future for allowing acrobatic maneuvers.

Given the cost function of Equation (6), the Quadratic Problem (QP) can be finally formulated as follows:

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \boldsymbol{J_{joint}}(\boldsymbol{T}) = \boldsymbol{p}^T \boldsymbol{Q} \boldsymbol{p} \\ \text{s.t.} \quad & \boldsymbol{A_{tot}} \boldsymbol{p} \leq \boldsymbol{B_{tot}} \end{aligned}, \tag{8}$$

where $\boldsymbol{A_{tot}}$ and $\boldsymbol{B_{tot}}$ are matrices defining the constraint set over the trajectory.

Equation (8) is particularly useful for the implementation, since standard functions available on MATLAB (such as `quadprog`) can be used for solving the optimization problem. Furthermore, since MATLAB 2021, the C++ Coder toolbox allows for the export of C++ code, simplifying notably the work [16].

Finally, we can address the problem of constraint formulation. They can be sorted into three categories:

- *Position constraints;*
- *Derivative constraints;*
- *Corridor constraints.*

2.4. Position Constraints

These are imposed at the endpoints of each segment to assure the passage in every desired waypoint at given time instants. Specifically, given $\boldsymbol{T}$, position constraints can be easily written as a matrix in the following way:

$$\boldsymbol{A}\boldsymbol{p} = \boldsymbol{B}. \tag{9}$$

Here, $\boldsymbol{A}$ is a block-diagonal matrix in which every component $\boldsymbol{A}(i,i)$ has been computed by evaluating $P_i(t)$ at $t = 0$ and $t = T_i$, and $\boldsymbol{B}$ is a vector containing the coordinates of each waypoint.

2.5. Derivative Constraints

Derivative constraints are necessary for defining the dynamics of the drone, for example, if the mission is started/finished from rest or not, but also for guaranteeing a smooth enough trajectory. We define derivative constraints up to the second-order derivative, since it is a good compromise between accuracy and computation time for our purposes. In fact, higher-order derivative constraints improve the smoothness, but the computation time for solving the optimization problem increases, and hence it is not suitable for real-time applications.

If velocities and accelerations are known at every endpoint, these constraints can then be written similarly to Equation (9), with appropriate substitution in $\boldsymbol{A}$ and $\boldsymbol{B}$.

In our application, we assume that just the derivatives of the first and last detected waypoints were known, while the others were left free. However, some continuity constraints over those halfway points have to be imposed to guarantee a smooth trajectory. This problem can be overcome by requiring that the derivatives at the end of the i-th segment match the derivatives at the beginning of the $(i+1)$-th segment, i.e.,:

$$\boldsymbol{A}_{T_i,i}\boldsymbol{p_i} - \boldsymbol{A}_{0,i+1}\boldsymbol{p_{i+1}} = 0, \tag{10}$$

which can be stored as a matrix as follows:

$$\underbrace{\begin{bmatrix} \boldsymbol{A}_{T_1,1} & -\boldsymbol{A}_{0,2} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_{T_2,2} & -\boldsymbol{A}_{0,3} & \dots & \boldsymbol{0} & \boldsymbol{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{A}_{T_M,M-1} & -\boldsymbol{A}_{0,M} \end{bmatrix}}_{\boldsymbol{A}_{HW}} \begin{bmatrix} \boldsymbol{p_1} \\ \boldsymbol{p_2} \\ \vdots \\ \boldsymbol{p_{M-1}} \\ \boldsymbol{p_M} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{B}_{HW}}. \tag{11}$$

2.6. Corridor Constraints

Corridor constraints were first introduced in [6] because, sometimes, especially in indoor environments, the available space for a drone's maneuvers is very tight. In these situations, additional constraints are needed for avoiding possible collisions with the surroundings. Let $\boldsymbol{u}_i$ be the unit vector along the direction of the i-th segment and $\boldsymbol{r} = [x(t), y(t), z(t)]^T$ be the position of the drone center of mass expressed in the world frame. The perpendicular distance vector $\boldsymbol{d_i}(t)$ from segment i is therefore

$$\boldsymbol{d_i}(t) = (\boldsymbol{r}(t) - \boldsymbol{r}_i) - ((\boldsymbol{r}(t) - \boldsymbol{r}_i) \cdot \boldsymbol{u}_i)\boldsymbol{u}_i. \tag{12}$$

A corridor constraint on segment i can be translated as requiring that $\boldsymbol{d_i}(t)$ does not exceed a maximum value δ_i, i.e.,:

$$\|\boldsymbol{d}_i(t)\|_\infty \leq \delta_i \quad \text{with} \quad t \in [0, T_i]. \tag{13}$$

Then, Equation (13) must be rearranged in a matrix structure to be inserted in the QP, Equation (8). By introducing n_c additional intermediate points and projecting vector $\boldsymbol{d}_i(t)$ on the three-world axis $\boldsymbol{x}_W$, $\boldsymbol{y}_W$, and $\boldsymbol{z}_W$, Equation (13) becomes

$$\left| \boldsymbol{x}_W \cdot \boldsymbol{d}_i\left(\frac{k}{1+n_c} T_i\right) \right| \leq \delta_i \quad \text{for} \quad k = 1, \ldots, n_c. \tag{14}$$

Equivalent expressions can be derived for $\boldsymbol{y}_W$ and $\boldsymbol{z}_W$ as well. Finally, Equation (14) can be split into two linear constraints by removing the absolute value. This last step yields an equation that can be finally added as a constraint in the original problem.

2.7. Time Allocation

The reader should have noticed that, in the formulation of the optimization problem described by Equation (8), both the cost function and the constraints depend explicitly on vector $\boldsymbol{T}$. Hence, $\boldsymbol{T}$ must be fixed prior to the optimization.

A common approach is to use the steepest descent method, which allows for smart allocation of the times for every segment [5,24]. Concerning our algorithm, we proceeded differently, since the steepest descent method remarkably increases the computation time. The algorithm that we derived simply checks that, for every segment, both velocity and acceleration are kept within the desired ranges for each component. In the eventuality that the requirements are not satisfied, a new augmented time is imposed over just those segments violating the constraints. Then, a new optimal trajectory using Equation (8) is computed. Therefore, our method does not aim to find an optimal time allocation; rather, it finds the first feasible configuration. It is also clear that the speed with which the algorithm converges to a feasible solution depends on how much each violating segment time is increased. However, this approach guarantees that it always converges to a solution, since all maneuvers are possible for a long enough travel time. Concerning the experimental results and our bounds—$v_{max} = 0.35$ m/s and $a_{max} = 2.5$ m/s^2—an increase of $+0.5$ s is enough for converging to a solution within 10 iterations.

2.8. Experimental Setup

Our experiments have been conducted in an indoor environment where we defined two different possible scenarios: (i) a known environment with a priori knowledge of the mission, and (ii) an unknown environment where the location of each marker must be uncovered on the go. The former scenario is less interesting from a practical point of view, since it does not reflect the common conditions in which a UAV must work. On the contrary, the latter requires efficient algorithms so that a trajectory can be computed in real time using the current available information. Therefore, the last scenario is important for testing the goodness of our proposed algorithm, while those tests in a known environment are relevant just for comparing the discrepancies between the predicted trajectories in the two scenarios. In the following, we refer to the known environment as the "one-step" approach, since all the computations are done altogether, while with "iterative", we refer to the second scenario, in which the trajectory is iteratively computed as new data become available.

Before moving to the next section, we would like to give further details about the experimental setup and implementation for the iterative approach. At every iteration, module (a) in Figure 4 outputs a vector containing the coordinates of those markers detected in the current frame. The dimension of this vector is dynamic since the number of markers seen may differ from frame to frame. The vector is then fed to the Mission Supervisor, module (b), in which a refinement of the markers' locations and trajectory planning are carried out. Specifically, whenever a new coordinate vector is provided by

(a), the Map Generator (b1) checks whether those coordinates refer to new markers or not. If a new marker is seen, then it is saved and added to the map (memory); otherwise, the Map Generator decides whether to update the marker's estimated position or not given the following Equation (15):

$$\boldsymbol{d}_l = \left|\hat{\boldsymbol{p}}_{c,l} - \hat{\boldsymbol{p}}_{o,l}\right| < \boldsymbol{\eta}_l, \tag{15}$$

where $\boldsymbol{d}_l$ represents the difference between the current $\hat{\boldsymbol{p}}_{c,l}$ and the old $\hat{\boldsymbol{p}}_{o,l}$ estimated position vectors of marker l, respectively. Therefore, the position of marker l in the map is updated if and only if $d_l \leq \eta_l$, with η_l defined by the user. This logic has been implemented to avoid excessive calls to the trajectory planning algorithm (b2). In fact, every time new estimated positions are available, the trajectory must be recomputed, but this requires a lot of computational effort.

Finally, we would like to make the readers aware that our markers have to be imagined at the center of gates, and the drone must cross these passages as perpendicular as possible for safety reasons. Hence, in addition to corridor constraints, for every detected marker, two additional points are generated along the perpendicular line of the gate plane, one before and the other after the passage; see Figure 5. Therefore, the actual waypoints with which the new trajectory is computed are the last point of the current trajectory and the pair of additional points for each detected marker.

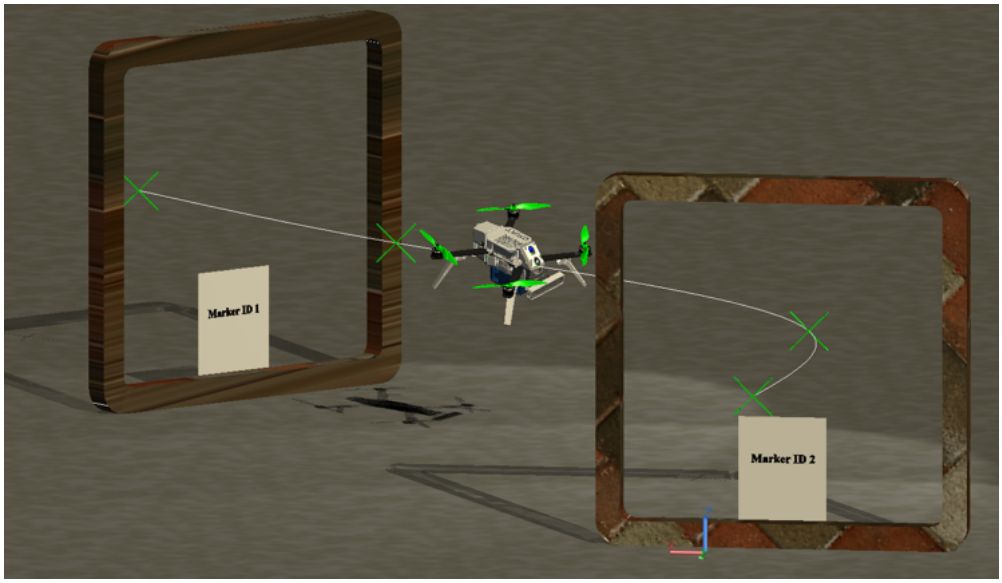


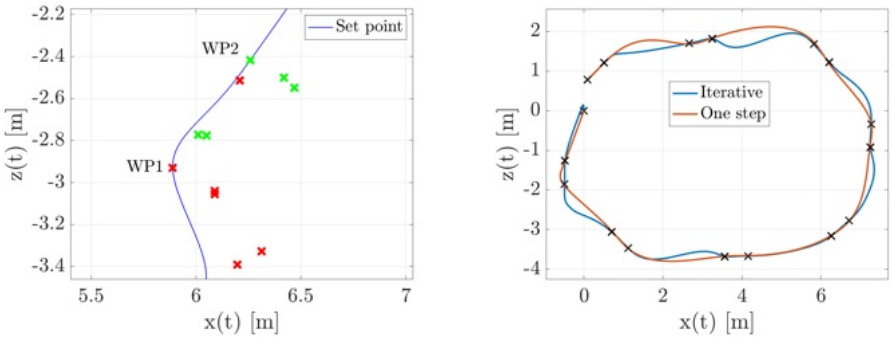
Figure 5. Ideally, the markers represent the center of gate passages through which the drone must fly. For each marker, two additional points are generated along the line perpendicular to the gate plane.

3. Results & Discussion

In this section, we report the outcomes of the tests that we performed to validate our algorithm. In particular, as mentioned previously, we give more emphasis on those experiments done over an unknown environment; at first, the drone is not aware of the position of the markers, and therefore it needs to detect them during the flight. After comprehending the designated positions and establishing the map, the drone ideally possesses the ability to execute the mission seamlessly, obviating the necessity for trajectory recalculations. In Figure 6a, we report an example of such an estimation process. The reader can note two clusters of waypoints (red and green crosses) representing the consecutive estimated positions of points WP1 and WP2, respectively. For each new estimate, the Map Generator (b1) evaluates Equation (15) and updates the map if needed until the last iteration. Then, the ultimate corrected estimates are kept and sent to the planning module (b2), and a trajectory is computed so that it interpolates all those points. For the sake of clarity, in Figure 6b, we show the computed trajectory using our iterative algorithm (blue line), where the path planning module was recalled whenever new markers were available

and/or when Equation (15) was violated. This trajectory differs enough from the one-step path (red line) where markers are fed altogether to the planning module.

The one-step path is the real optimal trajectory, since it is evaluated by optimizing the whole mission. However, in a real scenario, such a trajectory could not be computed, since it requires perfect knowledge about markers at zero time. Instead, the iterative approach takes care of just the current information provided. Hence, the drone iteratively updates the trajectory with its available information and its current state, resulting in an optimal trajectory with respect to the actual scenario; however, it is suboptimal compared to the red line.



(**a**) (**b**)

Figure 6. In (**a**), we zoom in on an example of a cluster of estimated waypoints. Among all of them, just the last are kept and sent to the path planning module. In (**b**), a comparison between the trajectory generated using our iterative algorithm (blue line) and the traditional way of proceeding (red line) is shown.

To further validate our algorithm, we conducted additional flight tests. For example, in some missions, we required the drone to fly twice around the desired sequence of markers. Again, in the first lap, the drone is not aware of the surrounding environment, hence it must detect the markers and create the trajectory. Then, during the second lap, having already produced a map of the environment, the drone does not need to recompute a trajectory (unless deep changes in the positions of the markers are observed); therefore, it repeats the same path as before. An example of such a flight test is shown in Figure 7, where the top view of the desired and real trajectories are reported. The desired trajectory (dashed line) is computed on board by the path planning module during the first lap. However, for this specific test, it is recomputed again in the second lap, since the new waypoint positions differ from the old ones by an amount more than η. Instead, the continuous line represents the real trajectory performed by the drone, which is slightly different from the set point due to the control error.

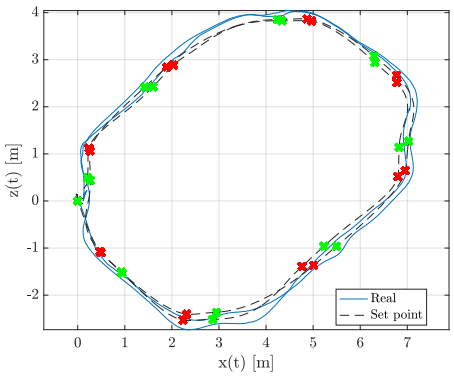


Figure 7. Top view of the x–z plane of the desired (dashed line) and actual (continuous) trajectories. Furthermore, the red x's identify the waypoints preceding a marker, while the green x's identify the subsequent waypoints.

4. Conclusions

In this paper, we implemented a real-time trajectory planning algorithm for an autonomous drone with the peculiarity that all the computations are carried out by exploiting the on-board sensors and computing resources. In particular, the suggested algorithm can find a trajectory that is optimal and feasible (with respect to the physical limits of the drone), but it can also replan a path whenever new incoming information becomes available.

To validate the efficiency of our algorithm, tests were conducted in an unknown environment with the positions of the targets not known a priori. Given this context, the drone had to detect the markers on the go and compute a desired trajectory solely based on the available data. Then, whenever new waypoints were discovered or updated, a replanning step was needed.

Those tests proved the readiness of the algorithm, since the drone was always able to recompute an optimal path in real time without stopping. Clearly, the computed trajectory appears to be optimal with respect to the current available information, but it is suboptimal compared to the one that would have been computed assuming a priori knowledge of the targets.

In the present work, we tackled just the trajectory planning task, but in the future, we aim to join this algorithm with some obstacle avoidance techniques that are essential when addressing trajectory planning problems in a dynamic environment.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/drones8060211/s1.

Author Contributions: Conceptualization, M.G., L.B. and E.B.; Methodology, M.G.; Software, L.B.; Formal analysis, M.G.; Investigation, M.G., L.B. and E.B.; Writing—original draft, M.G.; Writing—review and editing, M.G., L.B., E.B. and M.B.; Visualization, M.B.; Supervision, E.B. and M.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available in the Supplementary Material of the article.

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

References

1. Raivi, A.M.; Huda, S.M.A.; Alam, M.M.; Moh, S. Drone Routing for Drone-Based Delivery Systems: A Review of Trajectory Planning, Charging, and Security. *Sensors* **2023**, *23*, 1463. [CrossRef] [PubMed]
2. Flores-Caballero, G.; Rodríguez-Molina, A.; Aldape-Pérez, M.; Villarreal-Cervantes, M.G. Optimized Path-Planning in Continuous Spaces for Unmanned Aerial Vehicles Using Meta-Heuristics. *IEEE Access* **2020**, *8*, 176774–176788. [CrossRef]
3. Morbidi, F.; Cano, R.; Lara Alabazares, D. Minimum-energy path generation for a quadrotor UAV. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 1492–1498. [CrossRef]
4. Zhang, X.; Duan, H. An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning. *Appl. Soft Comput.* **2015**, *26*, 270–284. [CrossRef]
5. 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]
6. 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]
7. Nieuwstadt, M.J.V.; Murray, R.M. Real-time trajectory generation for differentially flat systems. *Int. J. Robust Nonlinear Control* **1998**, *53*, 9405–9411. [CrossRef]
8. Gugan, G.; Haque, A. Path Planning for Autonomous Drones: Challenges and Future Directions. *Drones* **2023**, *7*, 169. [CrossRef]
9. Ubbink, J.B.; Engelbrecht, J.A.A. Sequence-Constrained Trajectory Planning and Execution for a Quadrotor UAV with Suspended Payload. *IFAC-PapersOnLine* **2020**, *53*, 9405–9411. [CrossRef]
10. Chen, P.; Jiang, Y.; Dang, Y.; Yu, T.; Liang, R. Real-Time Efficient Trajectory Planning for Quadrotor Based on Hard Constraints. *J. Intell. Robot. Syst.* **2022**, *105*, 52. [CrossRef]
11. Singh, R.; Ren, J.; Lin, X. A Review of Deep Reinforcement Learning Algorithms for Mobile Robot Path Planning. *Vehicles* **2023**, *5*, 1423–1451. [CrossRef]
12. Foehn, P.; Romero, A.; Scaramuzza, D. Time-optimal planning for quadrotor waypoint flight. *Sci. Robot.* **2021**, *6*, eabh1221. [CrossRef] [PubMed]

13. Romero, A.; Sun, S.; Foehn, P.; Scaramuzza, D. Model Predictive Contouring Control for Time-Optimal Quadrotor Flight. *IEEE Trans. Robot.* **2022**, *38*, 3340–3356. [CrossRef]
14. Zimmermann, M.; Vu, M.N.; Beck, F.; Nguyen, A.; Kugi, A. Two-Step Online Trajectory Planning of a Quadcopter in Indoor Environments with Obstacles. *arXiv* **2023**, arXiv:2211.06377.
15. Sabetghadam, B.; Alcántara, A.; Capitan, J.; Cunha, R.; Ollero, A.; Pascoal, A. Optimal Trajectory Planning for Autonomous Drone Cinematography. In Proceedings of the 2019 European Conference on Mobile Robots (ECMR), Prague, Czech Republic, 4–6 September 2019; pp. 1–7. [CrossRef]
16. MATLAB Coder. Available online: https://www.mathworks.com/help/coder/ (accessed on 10 May 2024).
17. Basso, M.; Bigazzi, L.; Innocenti, G. DART Project: A High Precision UAV Prototype Exploiting On-board Visual Sensing. In Proceedings of the ICAS 2019: Fifteenth International Conference on Autonomic and Autonomous Systems, Athens, Greece, 2–6 June 2019; pp. 27–31.
18. Bigazzi, L.; Gherardini, S.; Innocenti, G.; Basso, M. Development of non expensive technologies for precise maneuvering of completely autonomous unmanned aerial vehicles. *Sensors* **2021**, *21*, 391. [CrossRef] [PubMed]
19. Bigazzi, L.; Basso, M.; Boni, E.; Innocenti, G.; Pieraccini, M. A Multilevel Architecture for Autonomous UAVs. *Drones* **2021**, *5*, 55. [CrossRef]
20. Florence Robotics. Available online: https://www.florence-robotics.com/ (accessed on 10 May 2024).
21. Kalaitzakis, M.; Cain, B.; Carroll, S.; Ambrosi, A.; Whitehead, C.; Vitzilaios, N. Fiducial Markers for Pose Estimation. Overview, Applications and Experimental Comparison of the ARTag, AprilTag, ArUco and STag Markers. *J. Intell. Robot. Syst.* **2021**, *101*, 71. [CrossRef]
22. Sani, M.F.; Karimian, G. Automatic navigation and landing of an indoor AR. drone quadrotor using ArUco marker and inertial sensors. In Proceedings of the 2017 International Conference on Computer and Drone Applications (IConDA), Kuching, Malaysia, 9–11 November 2017.
23. Khazetdinov, A.; Zakiev, A.; Tsoy, T.; Svinin, M.; Magid, E. Embedded ArUco: A novel approach for high precision UAV landing. In Proceedings of the 2021 International Siberian Conference on Control and Communications (SIBCON), Kazan, Russia, 13–15 May 2021.
24. Liu, T.L.; Subbarao, K. Optimal Aggressive Constrained Trajectory Synthesis and Control for Multi-Copters. *Aerospace* **2022**, *9*, 281. [CrossRef]
25. Richter, C.; Bry, A.; Roy, N. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research, Proceedings of the 16th International Symposium ISRR, Singapore, 16–19 December 2013*; Springer: Cham, Switzerland, 2016; pp. 649–666.