sensors

MDPI

Article

Optimal Motion Planning in GPS-Denied Environments Using Nonlinear Model Predictive Horizon

Younes Al Younes and Martin Barczyk *

Department of Mechanical Engineering, University of Alberta, Edmonton, AB T6G 1H9, Canada; alyounes@ualberta.ca
* Correspondence: mbarczyk@ualberta.ca

Abstract: Navigating robotic systems autonomously through unknown, dynamic and GPS-denied environments is a challenging task. One requirement of this is a path planner which provides safe trajectories in real-world conditions such as nonlinear vehicle dynamics, real-time computation requirements, complex 3D environments, and moving obstacles. This paper presents a methodological motion planning approach which integrates a novel local path planning approach with a graph-based planner to enable an autonomous vehicle (here a drone) to navigate through GPS-denied subterranean environments. The local path planning approach is based on a recently proposed method by the authors called Nonlinear Model Predictive Horizon (NMPH). The NMPH formulation employs a copy of the plant dynamics model (here a nonlinear system model of the drone) plus a feedback linearization control law to generate feasible, optimal, smooth and collision-free paths while respecting the dynamics of the vehicle, supporting dynamic obstacles and operating in real time. This design is augmented with computationally efficient algorithms for global path planning and dynamic obstacle mapping and avoidance. The overall design is tested in several simulations and a preliminary real flight test in unexplored GPS-denied environments to demonstrate its capabilities and evaluate its performance.

Keywords: motion planner; path planning; nonlinear model predictive approach; feedback linearization; dynamic obstacle avoidance; drone vehicle

Citation: Al Younes, Y.; Barczyk, M. Optimal Motion Planning in GPS-Denied Environments Using Nonlinear Model Predictive Horizon. *Sensors* **2021**, *21*, 5547. https://doi.org/10.3390/s21165547

Academic Editor: Carlos Silvestre

Received: 29 June 2021
Accepted: 16 August 2021
Published: 18 August 2021

1. Introduction

Throughout the last century, injuries and fatalities in subterranean environments have remained a major concern around the world. For example, mine workers are vulnerable to hazards such as cave-ins, underground flooding, and gas explosions. Unmanned vehicles can play a key role in performing both tedious and dangerous tasks, for instance air quality sampling, tunnel inspections, and search-and-rescue missions. A flying drone is a particularly attractive platform for underground operations due to its abilities to move quickly, traverse any terrain, navigate through tight spaces, and capture data from any angle. Recent advances in robotics have motivated research into designing novel path planning approaches, allowing the vehicle to plan safe paths and navigate through previously unknown environments.

Path planning is a computational problem to generate and follow a collision-free trajectory from one point to another [1]. It has many applications, such as robotic surgery [2], driverless cars [3], automation [4], and mining [5]. An extensive amount of research has been conducted in the field of path planning for autonomous vehicles [3,6]. However, most of the presented approaches provide non- or sub-optimal solutions and do not account for the dynamics of the vehicle, instead treating it as a kinematic model with velocity inputs [1], for instance a unicycle or kinematic car [7]. Moreover, navigating through dynamic and unknown environments is a challenging task as it requires safe navigation around both static and dynamic obstacles, which adds computational load for the onboard computer of the autonomous vehicle. Nonlinear Model Predictive Control (NMPC [8]) is

an attractive methodology to address the above-named challenges, since it is capable of predicting optimal trajectories, accounts for the dynamics of the plant, and supports hard state constraints which can be used to model either static or dynamic obstacles.

The goals of this paper are twofold. The first goal is presenting the design of a local path planning approach and applying it to a multi-rotor drone. The nonlinear dynamics of a drone makes it an excellent test candidate for this work. The path planning approach is based on recent work by the authors [9]. It combines a variant of NMPC, named Nonlinear Model Predictive Horizon (NMPH), which employs the Feedback Linearization (FBL) technique [10,11] to reduce the non-convexity of the optimization problem and thus provide faster solutions for the path planning problem. NMPH provides feasible solutions, generates smooth and collision-free paths, supports moving obstacles, is able to run in real-time, and reduces battery draw by minimizing abrupt drone motions. The second goal is developing a global motion planner for the drone to explore a subterranean environment. This operates by building a map of the environment and guiding the vehicle to unexplored areas within this map using a graph-based planner. The global motion planner is a design that integrates the local path planner design from the first goal with a graph-based planner named GBPlanner [5]. We propose a choice of computationally efficient algorithms for obstacle mapping and avoidance, plus robust path guidance. A block diagram of the proposed global motion planner design is shown in Figure 1.

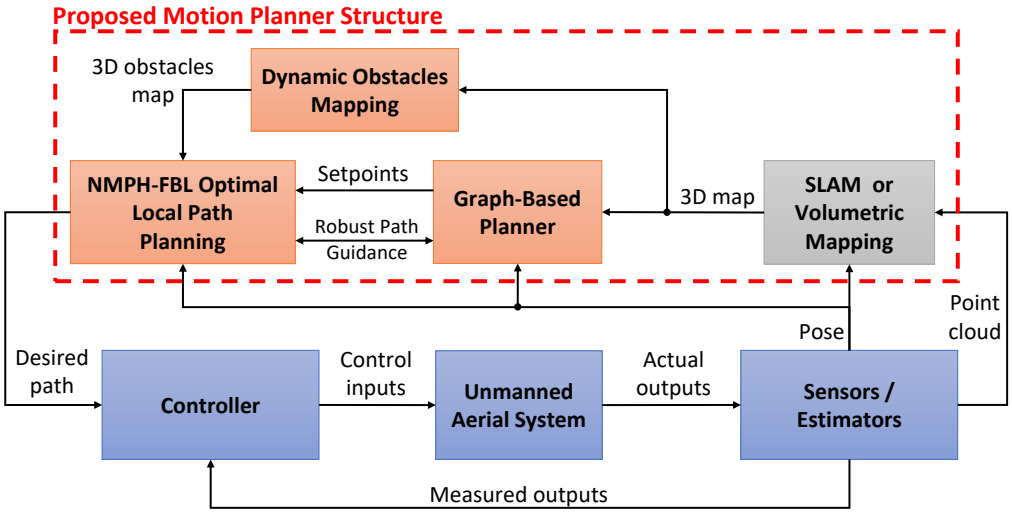


Figure 1. Block diagram of the proposed global motion planner.

The contributions of this work are as follows:

- A recently proposed trajectory generation algorithm (NMPH) is used for local path planning of the drone. The NMPH is integrated inside the global motion planner and produces optimal local trajectories for the drone vehicle in real-time.
- A methodological two-layer global motion planner design is proposed. The first layer utilizes a graph-based planner to generate terminal setpoints for the second layer, which uses the NMPH design to generate continuous optimal paths from the vehicle's current pose to the terminal setpoint in real time.
- Efficient algorithms for obstacle mapping and avoidance are proposed which produce models of static and dynamic obstacles used by the NMPH to generate safe and collision-free paths in a dynamically changing environment.
- A robust path guidance algorithm is implemented to avoid the risk of NMPH getting trapped into a local minima.
- The overall design is implemented using quadcopter and hexacopter drone dynamics, enabling navigation through unknown, dynamic and GPS-denied environments.
- Several simulation results and a preliminary experiment are presented in this work to validate the proposed approach.

The remainder of this paper is organised as follows. Section 2 surveys literature related to our work. Section 3 presents the problem formulation of NMPH and its integration with feedback linearization. The two-layer global motion planner design and choices of algorithms to provide robust path planning and obstacle avoidance are discussed in Section 4. System dynamic models of and implementation of our design on rotary-wing drones are presented in Section 5. In Section 6, various simulation and experimental results are presented to evaluate and validate the proposed approach. Finally, concluding remarks are given in Section 7.

2. Related Work

Path planning for an autonomous robot in an unknown, GPS-denied and dynamically changing environment is a challenging task, since the robot needs to plan trajectories that consider the vehicle's relative motion with respect to the surrounding obstacles. The path planning problem itself has been thoroughly studied in the literature and can be classified into three main categories: search-based, sampling-based, and optimization-based methods.

The search-based methods, a.k.a. grid-based, discretize the environment map into a graph of grids and use a search algorithm to find a collision-free path through these grids [6]. The two fundamental graph search algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS) [12]. BFS is based on a first-in-first-out queue and can produce an optimal solution if the graph is uniformly weighted. Meanwhile, a last-in-first-out stack is used in DFS until the goal is reached, but no optimality is guaranteed.

One of the most widely used optimal searching algorithms for quickly finding the shortest path is the Dijkstra algorithm [13]. It directs the search towards unvisited nodes, then calculates and updates the shortest distances to the neighbor nodes from the root node. It keeps doing this until all the nodes are visited. Meanwhile, A* [14] is a commonly used algorithm for path planning. A* is an extension to Dijkstra algorithm, where it combines the cost search with heuristics that guide the search towards the goal point to achieve quicker searching performance. Many extensions of A* have been proposed, for instance Lifelong Planning A* (LPA*) [15] was developed to support changes in the environment without recalculating the entire graph, D* Lite [16] extends LPA* to re-plan the path while the robot is moving, Anytime Repairing A* (ARA*) [17] improves the optimality of the path by reusing suboptimal solutions from previous executions, and Hybrid-state A* [18] generates the graph based on the robot dynamics and thus searches for a dynamically feasible path.

The sampling-based methods are considered one of the main motion planning methods for robots with a high number of Degrees-Of-Freedom (DOF) [19]. In these methods, feasible robot poses are randomly sampled to form admissible paths. Probabilistic Road-Map (PRM) [20] and Rapidly-Exploring Random Tree (RRT) [21] are the fundamental sampling-based methods for motion planning. In PRM, a graph is built from random configurations and connected using a local planner (for instance Dijkstra's searching algorithm for the shortest path between two configuration). PRM is complete but does not necessarily provide an optimal path solution. The RRT method is designed to randomly build a space-filling tree of vertices and edges inside a complex environment to find a feasible path to the goal node. However, the RRT-generated paths are not optimal [22]. Asymptotic optimality of paths can be achieved by employing various extensions of RRT, such as RRT* and Rapidly-Exploring Random Graph (RRG) methods [22]. RRG constructs a graph by connecting new samples with all nodes within a specified distance, then finding the shortest path using a local planner such as the Dijkstra algorithm. Meanwhile, RRT* searches the local nodes and finds the shortest path from the start to end nodes.

In general, there are three main limitations of the search- and sampling-based motion planning approaches. First, they do not account for the constraints imposed by the robot dynamics, even if some support kinematic and/or dynamic constraints (e.g., velocity and/or acceleration limits, respectively) [23]. A second limitation is consistency, since for

several executions the algorithms may not produce identical trajectories between a start and goal configuration in the very same environment. Third, the computational load of these methods generally prevents them from being able to actively regenerate paths while moving between the start and goal configuration, which makes motion planning a difficult task in dynamic environments. However, some optimization-based methods can overcome these limitations, and the present work is directed at using optimization for planning safe, consistent, and time-efficient paths which also respect the dynamics of the vehicle. This last feature allows generating smooth trajectories for the robot vehicle, avoding the jerky motions and rapidly changing trajectories often generated by other planning methods [1].

The optimization-based approaches solve a constrained non-convex optimization problem to smooth the trajectory generated by other methods. Some optimization-based methods use cost-gradient information of a trajectory's waypoints for refinement purposes, for instance CHOMP [24], Trajopt [25], and STOMP [26]. Other optimization-based methods are more closely related to optimal control, which focuses on system dynamics more than collision prevention. Examples include dynamic programming [27], LQR-based [28], and Model Predictive Control (MPC) [29].

One of the challenges of using an optimization-based path planning approach is accounting for obstacle constraints at each time instant the optimization problem is solved, especially for real-time implementations [30]. For a small number of obstacles, it has been demonstrated that finding local optimal trajectories is possible with MPC in outdoor environments [31]. Conversely, increasing the number of obstacles and considering 3D and dynamic environments makes the optimization problem much more computationally expensive to find feasible paths in real-time.

Our proposed formulation addresses the above challenges in two ways. First, it reduces the non-convexity of the optimization problem by combining the nonlinear plant model with a feedback linearization. Second, it maps obstacles as possibly moving volumes, and adaptively introduces state constraints modeling them in order to efficiently find local solutions. Our proposed design is integrated with a graph-based exploration planner [5] in order to provide global motion planning capabilities.

3. Nonlinear Model Predictive Horizon for Path Planning

The Nonlinear Model Predictive Horizon is a recently proposed methodology by the authors [9]. The purpose of NMPH is to generate optimal reference trajectories for closed-loop systems, and it will be implemented here for path planning. An overview of the NMPH algorithm is presented next.

3.1. NMPH Algorithm for Optimal Trajectory Generation

NMPH is an optimization-based reference trajectory generation technique for nonlinear closed-loop systems. Using a model of the nonlinear plant dynamics plus a feedback linearization control law, the NMPH creates optimal reference trajectories for a closed-loop system as shown in Figure 2. The generated trajectory is continuously updated by accounting for the current state of the system and path constraints within the optimization problem.

The purpose of the nonlinear control law within the NMPH is to reduce the nonlinearity of the system model, and consequently the non-convexity of the optimization problem. This leads to better performance in terms of reduced computational time and better convergence properties, enabling motion planning to be repeatedly computed in real time.

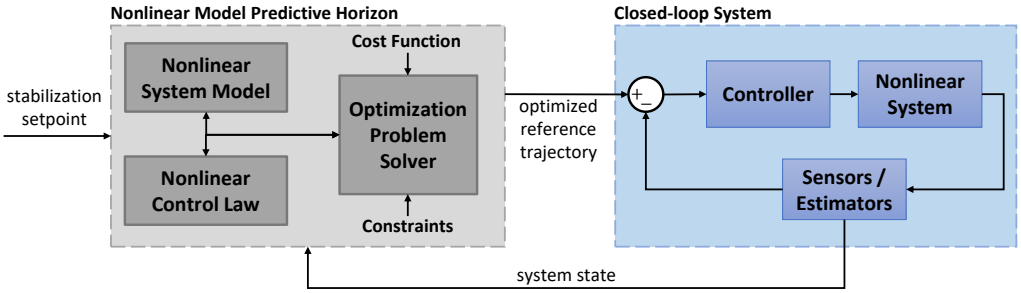


Figure 2. NMPH architecture. A model of the nonlinear system dynamics is used to perform the optimization process within NMPH (gray box). The resulting optimized reference trajectory is passed to the actual closed-loop system (blue box) for tracking purposes.

Consider a continuous-time nonlinear system controlled by a nonlinear control law,

$$\dot{x} = f(x, u) \tag{1}$$
$$\xi = h(x) \tag{2}$$
$$u = g(x, \xi_{ref}) \tag{3}$$

where $x \in X \subseteq \mathbb{R}^{n_x}$ are the system states, $u \in U \subseteq \mathbb{R}^{n_u}$ are the system inputs, and $\xi \in \Xi \subseteq \mathbb{R}^{n_\xi}$ are the system outputs (here 3D position and heading, such that $\Xi \subseteq X$). The reference trajectory is denoted by $\xi_{ref} \in \Xi$. The map $f : X \times U \to X$ represents the system dynamics and $g : X \times \Xi \to U$ is the control law that is used to make the system output follow the reference trajectory.

The optimization within NMPH uses Equations (1)–(3) plus assigned constraints to solve for the predicted system state trajectory $\tilde{x}$ (which for us includes the predicted output trajectory $\tilde{\xi}$ as a subset) and the estimated reference trajectory for the closed-loop system $\hat{\xi}_{ref}$, all while $\tilde{x}$ converges to a desired stabilization setpoint x_{ss}. The optimization calculations are summarized in Algorithm 1.

Within Algorithm 1, the optimization problem is solved over the finite time horizon $\tau \in [t_n, t_n + T]$. The cost function is chosen to penalize the deviation of the predicted system state trajectory from the stabilization setpoint x_{ss} , as well as the deviation of the predicted output trajectory $\tilde{\xi}$ from the estimated reference trajectory $\hat{\xi}_{ref}$. The weighting matrices W_x, W_ξ, and W_T are chosen by the user to balance the effects of the optimization. The cost function consists of the stage cost function $L\big(\tilde{x}(\tau), \hat{\xi}_{ref}(\tau)\big)$, which represents the cost of the problem over the time horizon, and the terminal cost function $E\big(\tilde{x}(t_n + T)\big)$, which represents the cost of steady-state error. $\mathcal{X} \subseteq X$, $\mathcal{U} \subseteq U$, and $\mathcal{Z} \subseteq X$ are the state, control, and trajectory constraint sets, respectively. The inequality constraint $\mathcal{O}_i(\tilde{x}) \leq 0$ is used to model p dynamic obstacles in the environment.

In Algorithm 1, the current system state $x(t_n)$ of the actual system is first measured or estimated, then a prediction of the reference trajectory $\hat{\xi}_{ref}$ for an admissible control is obtained by minimizing the cost function over the prediction horizon while respecting the system model, control law, and other constraints. Finally, the predicted reference trajectory is sent to the closed-loop system for tracking, and the above process is repeated at the next sampling time instant, possibly with an updated terminal setpoint x_{ss}.

Algorithm 1 (NMPH algorithm with stabilizing terminal condition x_{ss}).

1: Let t_n, $n = 0, 1, 2, \cdots$ represent successive sampling times; set $n = 0$
2: while $\|x_{ss} - x(t_n)\| \geq \delta$ **do**

$$\min_{\tilde{x}, \hat{\tilde{\xi}}_{ref}} \left(\int_{t_n}^{t_n+T} L\big(\tilde{x}(\tau), \hat{\tilde{\xi}}_{ref}(\tau)\big)\, d\tau \; + \; E(\tilde{x}(t_n + T)) \right) =$$

$$\min_{\tilde{x}, \hat{\tilde{\xi}}_{ref}} \left(\int_{t_n}^{t_n+T} \left(\|\tilde{x}(\tau) - x_{ss}\|^2_{W_x} + \|\tilde{\tilde{\xi}}(\tau) - \hat{\tilde{\xi}}_{ref}(\tau)\|^2_{W_{\tilde{\xi}}} \right) d\tau \right. \tag{4}$$

$$\left. + \|\tilde{x}(t_n + T) - x_{ss}\|^2_{W_T} \right)$$

$$\text{subject to} \qquad \tilde{x}(t_n) = x(t_n), \tag{5}$$

$$\dot{\tilde{x}}(\tau) = f(\tilde{x}(\tau), \tilde{u}(\tau)), \tag{6}$$

$$\tilde{u}(\tau) = g\big(\tilde{x}(\tau), \hat{\tilde{\xi}}_{ref}(\tau)\big), \tag{7}$$

$$\tilde{x}(\tau) \in \mathcal{X}, \;\; \tilde{u}(\tau) \in \mathcal{U}, \;\; \hat{\tilde{\xi}}_{ref}(\tau) \in \mathcal{Z}, \tag{8}$$

$$\mathcal{O}_i(\tilde{x}) \leq 0\,, \quad i = 1,\, 2,\, \ldots,\, p. \tag{9}$$

if $\tilde{x} \to x_{ss}$ **then** (estimated trajectory converging towards terminal setpoint)
$n \leftarrow n + 1$;
else
break;

In this paper, $\tilde{\xi}$ is defined as the predicted output trajectory which in our case (drone's position and yaw angle) represents a subset of the predicted state trajectory $\tilde{x}$, while $\hat{\tilde{\xi}}_{ref}$ is the estimated reference trajectory which is computed by minimizing the cost function Equation (4) within the NMPH. $\hat{\tilde{\xi}}_{ref}$ is used as the reference trajectory for the actual closed-loop system, providing smooth trajectories which respect the dynamics of the vehicle. Please note that $\xi_{ref}(n)$ in Equation (3) is taken as $\hat{\tilde{\xi}}_{ref}$, where the latter is actively updated by the NMPH algorithm in response to events such as new dynamic obstacles. Also, please note that $\hat{\tilde{\xi}}_{ref}$ and $\tilde{\tilde{\xi}}$ converge to each other, and both can be used as a reference trajectory for the actual closed-loop system (here the drone). Further details about the NMPH algorithm can be found in the authors' recent work [9].

3.2. Feedback Linearization Control Law

This section reviews the feedback linearization (FBL) control law used within the NMPH structure to improve the prediction performance of the reference trajectory. In this work, a Multi-Input Multi-Output (MIMO) control-affine system (here a drone) is used. The FBL design for this class of systems is summarized below.

Consider a MIMO nonlinear control-affine system of the form,

$$\dot{x} = f(x) + \sum_{i=1}^{n_u} \mathfrak{g}_i(x)\, u_i \triangleq f(x) + G(x)u \tag{10}$$

where f, $\mathfrak{g}_1, \ldots, \mathfrak{g}_{n_u}$ are smooth vector fields in $\mathbb{R}^{n_x}$. $G(x)$ is a $n_x \times n_u$ matrix with rank $G(0) = n_u$.

The objective of FBL is to transform the nonlinear system (10) into a linear canonical form with a non-singular state feedback [11], which is defined as

$$u = \beta(x) + D(x)^{-1} v \tag{11}$$

where $\beta(x)$ is a smooth function, and $\beta(0) = 0$. $D(x)^{-1}$ is the inverse of a non-singular $n_u \times n_u$ matrix. $D(x)$ and the non-singular state feedback transformation are

$$D(x) = \begin{bmatrix} \langle d\varphi_1, ad_f^{r_1-1}\mathfrak{g}_1\rangle & \dots & \langle d\varphi_1, ad_f^{r_1-1}\mathfrak{g}_{n_u}\rangle \\ \vdots & \ddots & \vdots \\ \langle d\varphi_{n_u}, ad_f^{r_{n_u}-1}\mathfrak{g}_1\rangle & \dots & \langle d\varphi_{n_u}, ad_f^{r_{n_u}-1}\mathfrak{g}_{n_u}\rangle \end{bmatrix} \tag{12}$$

$$v = \begin{bmatrix} L_f^{r_1}\varphi_1 \\ \vdots \\ L_f^{r_{n_u}}\varphi_{n_u} \end{bmatrix} + \begin{bmatrix} L_{\mathfrak{g}_1}L_f^{r_1-1}\varphi_1 & \dots & L_{\mathfrak{g}_{n_u}}L_f^{r_1-1}\varphi_1 \\ \vdots & \ddots & \vdots \\ L_{\mathfrak{g}_1}L_f^{r_{n_u}-1}\varphi_{n_u} & \dots & L_{\mathfrak{g}_{n_u}}L_f^{r_{n_u}-1}\varphi_{n_u} \end{bmatrix} u\,, \quad z = \begin{bmatrix} \varphi_1 \\ \vdots \\ L_f^{r_1-1}\varphi_1 \\ \vdots \\ \varphi_{n_u} \\ \vdots \\ L_f^{r_{n_u}-1}\varphi_{n_u} \end{bmatrix} \tag{13}$$

where $\{\varphi_i(x) : \langle d\varphi_i, \mathcal{G}_{r_i-2}\rangle = 0,\ j \geq i,\ i = 1, \dots, n_u\}$ are smooth functions that form the non-singular matrix $D(x)$. r_i are the controllability indices and $\mathcal{G}$ is a distribution of vector fields. $L_f\varphi = \langle d\varphi, f\rangle$ is the Lie derivative, L_f^r means the Lie derivative is applied r times, and $ad_f^r\mathfrak{g} = [f, ad_f^{r-1}\mathfrak{g}]$ is the Lie bracket between the vector fields f and $\mathfrak{g}$. Further details about the non-singular state feedback transformation for MIMO control-affine systems can be found in Theorem 2.7.3 of reference [11].

4. Motion Planning in GPS-Denied Environments

4.1. Motion Planner Architecture

Our proposed motion planning design aims to produce optimal vehicle paths while navigating in unexplored, dynamic and GPS-denied environments. We combine a graph-based exploration technique with a Nonlinear Model Predictive Horizon-based approach based on optimization which respects the vehicle's dynamics and supports dynamic obstacles. This integration yields feasible, optimal, and robust paths while exploring challenging environments.

Figure 3 describes the overall architecture of our motion planner. The design is composed of three successive stages. The first stage acquires sensor data to build a physical representation of the environment which contains both static and dynamic objects in it. Volumetric mapping is used for this stage since it is computationally efficient, easy to visualize, can be incrementally constructed and reconstructed online, and provides the voxel grid structure needed for the next stage. Details about the volumetric mapping and its layers will be discussed in Section 4.2.

Section 4.3 discusses the second stage of the motion planner, which is built around a graph-based planning approach. It consists of the sampling-based RRG algorithm, which builds a connected roadmap graph, and the Dijkstra searching algorithm to extract the best path from within the graph. The main purpose of the graph-based approach is to guide the vehicle towards unexplored areas within the environment and provide terminal vertices, a.k.a. stabilization or terminal setpoints, to the local path planner.

The third stage of the motion planner uses the NMPH-FBL local path planning method. Fusing this method with the earlier stages improves the robustness of generating optimal paths and avoiding static and dynamic obstacles. The considerations involved in finding a feasible path are shown in Figure 3. Further details are provided in Section 4.4.

The reference trajectory computed by the path planner is fed to the control system of the vehicle for tracking purposes. As the drone vehicle moves, the NMPH continues to update its reference trajectory in response to feedback of the vehicle's state and new obstacles. Once the vehicle reaches a setpoint, the motion planning process is repeated,

which continues until the environment is fully explored or the mission is interrupted by the operator.

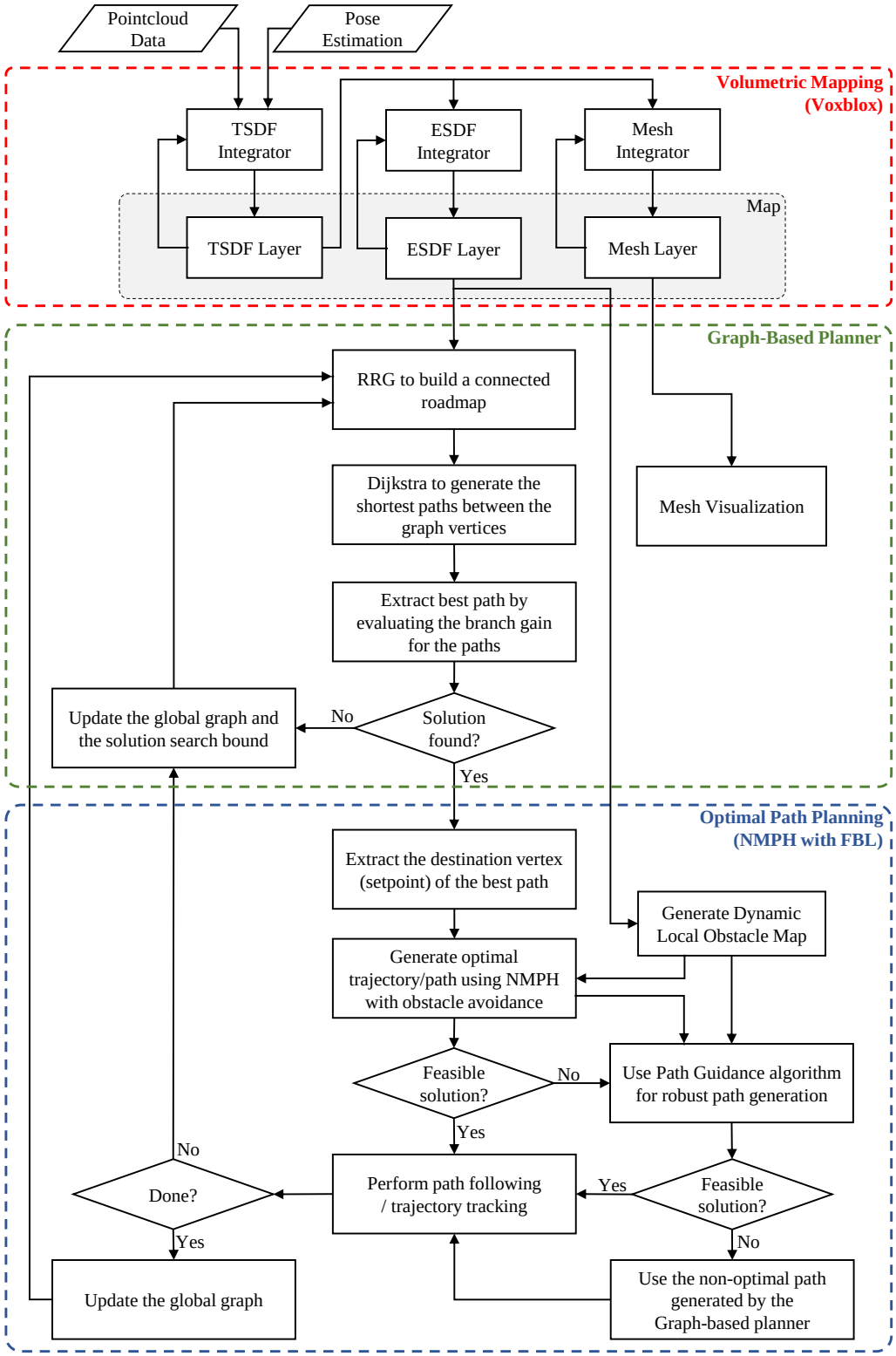


Figure 3. Motion Planner Architecture.

4.2. Volumetric Mapping

Volumetric mapping is the foundation of motion planning and navigation strategies in 3D environments. The volumetric mapping algorithm named Voxblox [32] is used in our work. In this technique, the map of the environment is represented volumetrically using the signed distance field to distinguish between known, unknown, free, and occupied spaces [33]. The grid consists of voxels with a corresponding type. Groups of occupied voxels represent surfaces of an object, walls, and so on. The main advantage of volumetric mapping is its real-time capability for incrementally representing unstructured and unexplored environments, which makes it a suitable solution for online planning and exploration. The Truncated Signed Distance Field (TSDF [34]) is one of the most efficient methods of representing volumetric maps. TSDF is an implicit surface representation that consists of a 3D voxel array. Each voxel is indexed by the distance of the ray between the sensor and the surface, and is truncated near the surface to decrease the errors that are caused by sensor noise. TSDFs are computationally efficient and can be constructed online. Also, they are capable of filtering out sensor noise and create meshes with voxel resolution for visualization purposes.

In contrast to TSDF, the Euclidean Signed Distance Field (ESDF) uses the Euclidean distance to the nearest occupied cell in labeling the voxel grid [32]. ESDFs are directly built out of existing TSDFs to make use of the distance information in determining the obstacle surface location for planning purposes. In other words, TSDF is for mapping and ESDF for planning, and the main difference between them is the way that distances are computed [35].

As presented in Figure 3, the volumetric mapping process consists of three layers. The sensor data is processed to build the TSDF layer, then the voxels are integrated into the ESDF and mesh layers as presented in [32]. The ESDF voxels and mesh blocks are updated incrementally allowing real-time map generation for planning and online visualization of the environment. To reduce the complexity of calculating the layers data, a voxel hashing approach [36] is used to store the information of each layer in a hash table, which results in $O(1)$ complexity for adding or retrieving the data.

4.3. Graph-Based Path Planning

In this section, we summarize the graph-based planner presented in [5], which is used to help the vehicle navigate through unknown GPS-denied environments.

Assume that $\mathbb{M}_G$ is a global 3D voxel-based map, which consists of voxels $m \in \mathbb{M}_G$. The map is incrementally built by a depth sensor $\mathbb{S}$ plus the vehicle's pose estimation using the volumetric mapping approach previously discussed in Section 4.2. The map is categorized into three spaces, free space $\mathbb{M}_G^f$, occupied space $\mathbb{M}_G^o$, and unknown space $\mathbb{M}_G^u$. The map has a global volume V_G and is incrementally constructed within a local map sub-space $\mathbb{M}_L$ of volume V_L centered around the current vehicle's output (here 3D position and heading) $\xi_0 = \begin{bmatrix} x_0 & y_0 & z_0 & \psi_0 \end{bmatrix}^T$.

The graph-based planner [5] performs a local search towards unknown areas of $\mathbb{M}_G$. It is based on the sampling-based RRG algorithm [37] which builds a connected roadmap graph $\mathbb{G}_G$ composed of collision-free vertices ν and edges e stored in vertex set $\mathbb{V}$ and edge set $\mathbb{E}$, respectively. The edges are straight paths connecting the vertices using the nearest neighbor search [38]. The global graph $\mathbb{G}_G$ is continuously constructed from the undirected local graph $\mathbb{G}_L$ within the local space $\mathbb{M}_L$. The local search within the bounded volume V_L considers the physical size of the vehicle V_R and bounds it within a sub-space $\mathbb{M}_R$. Collision detection is performed to ensure collision-free paths σ_L, where $\mathbb{M}_R \in \mathbb{M}_G^f$ for all randomly generated vertices and edges.

The set of all shortest paths Σ_L, starting from the initial or current vertex ξ_0 to all destination vertices ξ_ν, is found using the Dijkstra algorithm [12]. After that, the best path is evaluated by calculating the Volumetric Gain $\mathcal{V}$ for each vertex. The Volumetric Gain of a vertex is a measure of the unmapped volume based on the depth reading around that vertex. The weight functions related to distance and direction combined with $\mathcal{V}$ are used

to compute the Exploration Gain $\mathcal{E}(\sigma_i)$ for all $\sigma_i \in \Sigma_L$, $i = 1, \ldots, n$. The vertices of these paths are ν_j^i , $j = 1, \ldots, m_i$, and ν_0^i is the initial vertex along path σ_i. The Exploration Gain for a path is calculated as [5]

$$\mathcal{E}(\sigma_i) = e^{-\lambda_{\mathcal{S}}\, \mathcal{S}(\sigma_i, \sigma_{sp})} \sum_{j=1}^{m_i} \mathcal{V}\left(\nu_j^i\right) e^{-\lambda_{\mathcal{D}}\, \mathcal{D}(\nu_0^i, \nu_j^i)} \tag{14}$$

where $\mathcal{S}(\sigma_i, \sigma_{sp})$ is a distance factor between a path σ_i and its corresponding straight path σ_{sp} which has the same length along the estimated exploration direction. This factor prevents the vehicle from sudden changes in its exploration direction which might happen in branched environments within $\mathbb{M}_L$. $\mathcal{D}(\sigma_i, \sigma_{sp})$ is the distance between ν_j^i and ν_0^i of the path σ_i, which penalizes longer paths for a higher exploration rate. The tunable gains $\lambda_{\mathcal{S}}$ and $\lambda_{\mathcal{D}}$ are positive gains.

Subsequently, the best path σ_{best} that maximizes the Exploration Gain is selected and sent to the NMPH-FBL local motion planner to find the optimal path that the vehicle will follow. The whole procedure is repeated once the vehicle reaches the destination vertex. The detailed algorithm for building the map and planning the best path is presented in Algorithm 2.

If all vertices within $\mathbb{G}_L$ are explored, the search will be expanded to the unexplored vertices of $\mathbb{G}_G$. This will guide the vehicle to another location on the global map and continue the exploration mission. For the return-to-home feature, the Dijkstra algorithm is also used to find the shortest path between the vehicle's current output ξ_0 and the homing vertex on $\mathbb{G}_G$. This feature can be invoked once the exploration mission is completed, the battery level is low, or by intervention from the operator.

Algorithm 2 Graph-based Planner.

$\xi_0 \leftarrow$ **CurrentMeasurement**();
$\mathbb{M}_G \leftarrow$ **VolumetricMapping**($\mathbb{S}$);
$\mathbb{V} \leftarrow \{\xi_0\}$; $\mathbb{E} \leftarrow \emptyset$; $\mathbb{G}_L = (\mathbb{V}, \mathbb{E})$;
$\mathbb{M}_L \leftarrow$ **LocalBoundSpace**($\xi_0, \mathbb{M}_G$);
for $i = 1, \ldots, n$ **do** ▷ RRG to build the local graph $\mathbb{G}_L$
  $\xi_{rand} \leftarrow$ **SampleFree**$_i$($\mathbb{M}_L$);
  $\xi_{nearest} \leftarrow$ **NearestVertex**($\mathbb{G}_L, \xi_{rand}$);
  if CollisionFree($\xi_{rand}, \xi_{nearest}$) **then**
    $\mathbb{V} \leftarrow \mathbb{V} \cup \{\xi_{rand}\}$;
    $\mathbb{E} \leftarrow \mathbb{E} \cup \{\xi_{rand}, \xi_{nearest}\}$;
    $\mathbb{N}_{near} \leftarrow$ **NearVertices**($\mathbb{G}_L, \xi_{rand}$);
    for each $\xi_{nearest} \in \mathbb{N}_{near}$ **do**
      if CollisionFree(ξ_{rand}, ξ_{near}) **then**
        $\mathbb{E} \leftarrow \mathbb{E} \cup \{\xi_{rand}, \xi_{near}\}$;
      end if
    end for
  end if
end for
$\Sigma_L \leftarrow$ **DijkstraAlgorithm**($\mathbb{G}_L, \xi_\nu$); ▷ Find the shortest paths
$\sigma_{best} \leftarrow \emptyset$; $\mathcal{E}_{best} \leftarrow 0$;
for each $\sigma \in \Sigma_L$ **do** ▷ Find the best path
  $\mathcal{E}_\sigma \leftarrow$ **ExplorationGain(**σ, **VolumetricGain**($\mathbb{V}$)**)**;
  if $\mathcal{E}_\sigma > \mathcal{E}_{best}$ **then**
    $\sigma_{best} \leftarrow \sigma$; $\mathcal{E}_{best} \leftarrow \mathcal{E}_\sigma$;
  end if
end for
$\mathbb{G}_G \leftarrow \mathbb{G}_G \cup \mathbb{G}_L$; ▷ Update the global graph
if *ReturnHome* = *true* **then**
  $\sigma_{best} \leftarrow$ **DijkstraAlgorithm**($\mathbb{G}_G, \xi_{home}$); ▷ Find the shortest path to home
end if

4.4. Nmph for Local Path Planning

The graph-based planner in Section 4.3 generates non-optimal or sub-optimal paths because the vertices are created randomly within V_L. In addition, the straight edges connecting vertices cause jerky motions for a drone following the path. Finally, the path generated by the graph-based planner does not respect the vehicle's dynamics. The NMPH-equipped path planning approach presented in Algorithm 3 overcomes these issues by generating a path which respects the system's dynamics and provides a smooth and optimal path which also avoids obstacles. From Figure 3, the NMPH path planning stage includes

- Dynamic Local Obstacle Mapping (c.f. Section 4.4.1), a technique which utilizes the continously updated volumetric map of the environment to generate a dynamically changing map of obstacles which are used as constraints for the optimization within the NMPH algorithm.
- Obstacle Avoidance (c.f. Section 4.4.2), an algorithm which allows the optimization problem solver to select constraints which correspond to obstacles in the path of the vehicle.
- Path Guidance (c.f. Section 4.4.3), an algorithm which enhances the robustness of path generation to infeasible situations by making use of all the vertices of the graph-based planner-generated path, not just the terminal vertex. This allows the generation of multiple consecutive and feasible paths, leading to an overall path to the terminal vertex.

Algorithm 3 Local Optimal Path Planning using NMPH.

$\sigma_{best} \leftarrow$ **GraphBasedPlanner**$(\xi_0, \mathbb{M}_G)$;
$v_{term} \leftarrow$ **ExtractVertex**$_{terminal}(\sigma_{best})$;
$\mathbb{M}_{obs} \leftarrow$ **LocalObstacleBound**$(\xi_0, \mathbb{M}_G)$; ▷ Consider certain voxels around ξ_0
for $i = k, \ldots, n$ **do** ▷ Remove extra voxels
  for $j = i - k, \ldots, i$ **do**
    if $\|m_i - m_j\| < \delta$ **then**
      $\mathbb{M}_{obs} \leftarrow \mathbb{M}_{obs} \backslash m_i$; ▷ Remove v_i from the obstacles map
    end if
  end for
end for
$\mathbb{C}_L \leftarrow$ **ObstacleConstraint**$(v_{term}, \xi_0, \mathbb{M}_{obs})$; ▷ Find the obstacles constraints
$\sigma_{opt} \leftarrow$ **NMPH_Planning**$(v_{term}, \xi_0, \mathbb{C}_L)$;
if σ_{opt} **not** *feasible* **then**
  for $i = 1, \ldots, n$ **do** ▷ Path Guidance Algorithm
    $v_i \leftarrow$ **ExtractVertex**$_i(\sigma_{best})$;
    $\sigma_{opt} \leftarrow$ **NMPH_Planning**$(v_i, \xi_0, \mathbb{C}_L)$;
  end for
  if σ_{opt} **not** *feasible* **then**
    $\sigma_{opt} = \sigma_{best}$;
  end if
end if
PathFollowing(σ_{opt}); ▷ Follow the optimal path

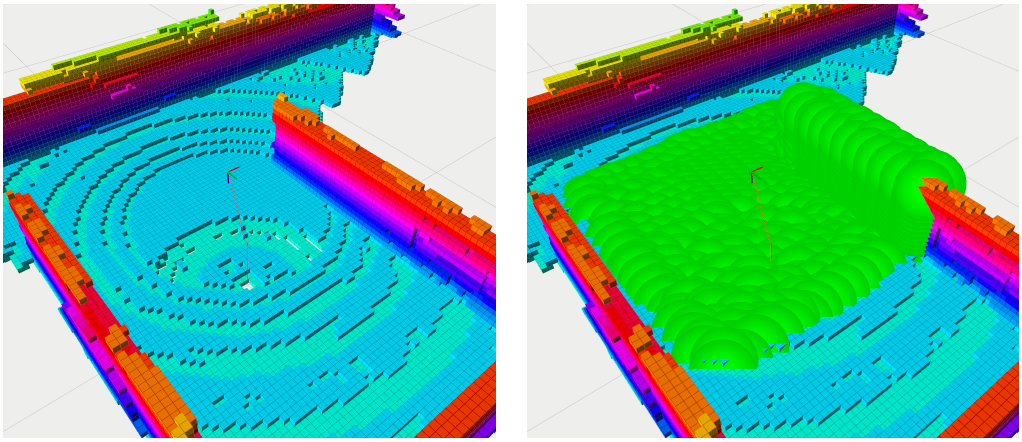
4.4.1. Dynamic Local Obstacle Mapping

Transforming physical obstacles within the volumetric map to optimization constraints is a challenging task. These obstacles need to be represented by a cluster of constraints while respecting the limitations of the optimization process, specifically a limit on the number of inequality constraints that the optimization problem can handle.

In this Section, we will present a strategy that maps obstacles in the environment into a dynamically moving space around the vehicle. This facilitates representing the obstacles as inequality constraints for optimization. This mapping technique, called Dynamic Local Obstacle Mapping (DLOM), generates a continuously changing map $\mathbb{M}_{obs}$.

Based on the occupied voxel in $\mathbb{M}_G^o$, the DLOM strategy generates virtual spheres centered on occupied voxels within a certain space surrounding the vehicle (e.g., a box

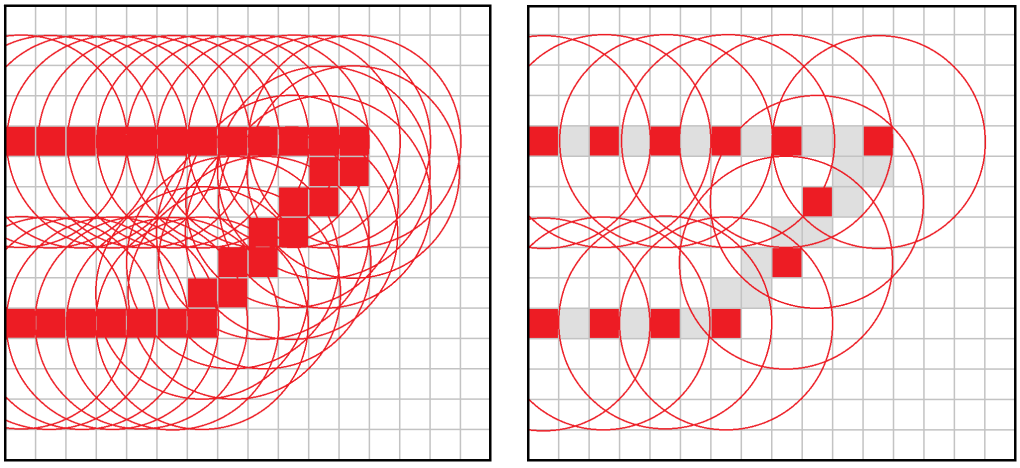
of dimensions D_{obs}). These virtual spheres have a radius which ensures a safe clearance between the vehicle sides and the occupied voxel. Figure 4 shows the volumetric map without and with DLOM. One advantage of using a sphere is for modeling the obstacle as a state constraint. This inequality constraint requires r^i_{obs}, the distance between the vehicle and the center of the i^{th} sphere, to be larger than a specific threshold r_{thld} representing the radius of the virtual sphere as $\left(r^i_{obs} \geq r_{thld}\right)$. The solution of the optimization problem within NMPH will thus generate a path that doesn't pass through the virtual spheres and hence avoids the obstacles in the environment.



(**a**) Without DLOM (**b**) With DLOM

Figure 4. Dynamic Local Obstacle Mapping (DLOM).

Modeling all occupied voxels in D_{obs} as obstacles would result in an excessively large computational burden to continuously generate $\mathbb{M}_{obs}$ and solve the optimization problem. Instead, any voxels inside the ith sphere are excluded from $\mathbb{M}_{obs}$. Lines (3–10) of Algorithm 3 employ a simple running window strategy to remove extra voxels, and those remaining are represented as virtual spheres which provide constraints to the optimization problem. Figure 5 shows how the extra spheres are removed to reduce the computational load involved in producing the obstacles map. The exact time needed to build the dynamic obstacles map depends on the number of occupied voxels within D_{obs}. Experimentally, we found that the time required to build such a map on a desktop-class machine with a modern GPU (detailed specifications are given in Section 6.1) takes approximately 3 ms.



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

Figure 5. Dynamic Local Obstacle Mapping. (**a**) All voxels are used to map the obstacle's surface. (**b**) A subset of voxels (highlighted in red) is selected to represent the obstacle's surface, and their neighbouring voxels are excluded.

4.4.2. Obstacle Avoidance Algorithm

As soon as the obstacle map is created, the NMPH creates an optimal local path respecting the constraints acquired from $\mathbb{M}_{obs}$. The optimization solver is limited in the number of inequality constraints it can handle, making it impossible to include all the mapped obstacles in $\mathbb{M}_{obs}$ within the optimization problem. Hence, a dynamic method for selecting a specific number of constraints is described next and included in Algorithm 4.

Algorithm 4 Obstacle Constraints.

function ObstacleConstraint($\nu_{term}, \xi_0, \mathbb{M}_{obs}$)
  $\sigma_{opt} \leftarrow$ **NMPH_Planning**(ν_{term}, ξ_0);
  $k = 1$;
  $\mathbb{C}_D \leftarrow \emptyset$; $\mathbb{C}_T^k \leftarrow \emptyset$; ▷ Dynamic and Temporary Constraint Arrays
  for $j = 1, \ldots, N$ **do** ▷ N is the number of the horizon points
    for $i = 1, \ldots, n_{obs}$ **do** ▷ n_{obs} is the number of obstacles
      if $r^i_{obs,\nu_j} < r_{thld}$ **then**
        $\mathbb{C}_D \leftarrow \mathbf{s}_i^j$; ▷ Store ith obstacle position which is indexed by j
        $\mathbb{C}_T^k \leftarrow \mathbf{s}_i$; ▷ Store ith obstacle position in the kth temp constraint
        $k = k + 1$;
        if k **is** n_T **then**
          $k = 1$;
        end if
      end if
    end for
  end for
  $\mathbb{C}_L = (\mathbb{C}_D, \mathbb{C}_T^k)$;
return $\mathbb{C}_L$

Our chosen solver provides a solution to the optimization problem every ∼4 ms (running on the computer described in Section 6.1), which makes it possible to solve the problem several times before sending the optimum reference path to the vehicle's flight control system. The Obstacle Avoidance algorithm takes advantage of this by first solving the optimization problem without considering obstacle constraints, then running a collision check on the generated path to find whether it crosses any virtual spheres in $\mathbb{M}_{obs}$. It is important to mention that the collision check is performed over the entire optimization horizon $[t_n, t_n + T]$ in Algorithm 1, which is discretized into N points for numerical computation.

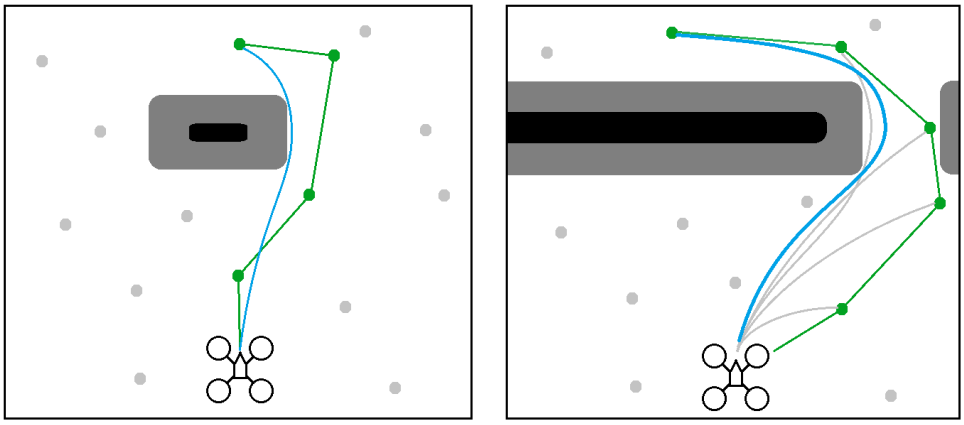
If a collision is detected at some points within the optimization horizon, a *Dynamic Constraint Array* registers the center of a sphere $\mathbf{s} \in \mathbb{R}^3$ that contains these collision points, and passes them to the solver as inequalities used to compute a new solution which avoids them. The *Dynamic Constraint Array* has dimensions of $N \times 3$ and can register up to N different inequality constraints for the next run of the optimization problem. For example, assume that a collision is detected on horizon points 3, 4 and 5, and each of the collision points are located within the 40th virtual sphere. In this case, the coordinates of the center of this sphere are registered in the *Dynamic Constraint Array* at indices 3, 4 and 5, while the rest of the array entries are kept Null. In the next iteration of the solver, a new constraint representing the cloned entries of the Dynamic Constraint Array will yield a path which avoids the region where the collisions previously occured.

To enhance the reliability of the Obstacle Avoidance algorithm while the vehicle is in motion, a specific number of Temporary Constraint Arrays (labeled by k in Algorithm 4) store the information from the Dynamic Constraint Array and are used in the optimization solution as well. The Temporary Constraint Arrays are static, which means that each registers only one virtual sphere over all its N indices.

4.4.3. Robust Path Guidance Algorithm

The initial state of the vehicle, the nature of the environment (e.g., branched narrow passages), and the terminal condition location may all affect the feasibility of the optimization problem solution. Figure 6 depicts two different path planning scenarios. In Figure 6a, the obstacle is small and NMPH can easily find a feasible solution. In Figure 6b, the obstacles almost block the way to the destination point. In this situation, the NMPH solver risks producing infeasible solutions by getting trapped in local minima.

As mentioned in Section 4.3, the graph-based path planning yields multiple vertices, which are used by the NMPH approach to generate multiple feasible paths, ranging from the nearest to the most distal (terminal) vertex. The small obstacle depicted in Figure 6a does not cause any issues for the NMPH in generating a feasible path directly to the terminal vertex. However, Figure 6b illustrates how the NMPH algorithm uses multiple consecutive paths (gray lines) generated to the intermediate vertices of the path generated by the graph-based planner (green line) to eventually find the resulting optimal path (blue line). Lines 12–21 in Algorithm 3 demonstrate the Path Guidance algorithm that adds robustness to the NMPH approach in finding a feasible solution. Note in case the Path Guidance algorithm is unable to help NMPH find a feasible path to the terminal vertex, the system can still use the path generated by the graph-based planner.



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

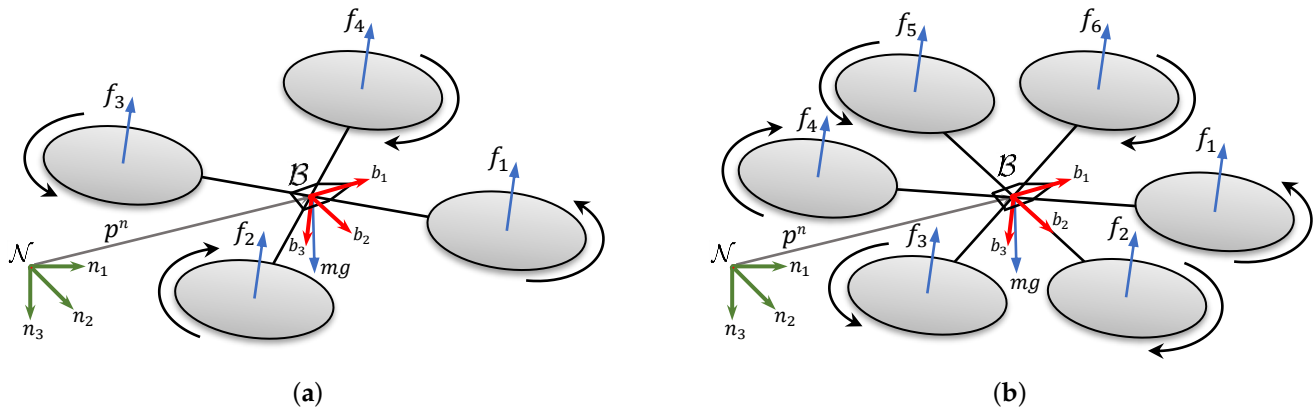
Figure 6. Graph-based vs NMPH path planning. (**a**) The terminal vertex of the green path (from graph-based planner) is sufficient to generate the optimum blue path by NMPH. (**b**) All the vertices of the green path are used successively to guide the solutions of NMPH to the final optimal trajectory (blue path).

5. Application of Motion Planner to A Drone

5.1. System Model

In this section, both a quadcopter and a hexacopter system are modeled as rigid bodies with lumped force and torque inputs at each rotor. For simplicity, drag forces, rotor gyroscopic effects, and propeller dynamics are not included in the models. The rigid-body dynamics are formulated using the Newton-Euler equations [39].

A fixed navigation frame $\mathcal{N}$ and a moving body-fixed frame $\mathcal{B}$ are the two reference frames used in this work and their basis are selected to follow the North, East, and Down (NED) aerospace convention. Schematics of the drones with their body-fixed reference frames and basis are depicted in Figure 7.



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

Figure 7. Schematics of (**a**) quadcopter and (**b**) hexacopter vehicles.

The dynamics of a rigid body moving through 3D space is represented by rigid-body kinematics and the Newton-Euler equations as shown below [39],

$$\begin{aligned} \dot{p}^n &= v^n \\ m\dot{v}^n &= -\bar{u}Rn_3 + gn_3 \\ \dot{R} &= R\,S(\omega^b) \\ J\dot{\omega}^b &= -S(\omega^b)J\omega^b + \tau^b \end{aligned} \tag{15}$$

where $p^n \in \mathbb{R}^3$ is the vehicle's position and $v^n \in \mathbb{R}^3$ is its velocity, both in coordinates of the inertial navigation frame $\mathcal{N}$. The mass moment of inertia matrix J is assumed to be diagonal as $J = \text{diag}([J_1, J_2, J_3])$, and m is the total mass of the drone. $\omega^b, \dot{\omega}^b$ are the angular velocity and acceleration vectors, respectively, in coordinates of the body-fixed frame. The rotation matrix of $\mathcal{B}$ with respect to $\mathcal{N}$ is $R = R_{nb} \in \text{SO(3)}$. $S(\cdot) \in \mathbb{R}^{3\times3}$ is a skew-symmetric matrix such that $S(x)y = x \times y$, $x, y \in \mathbb{R}^3$. The system input vector is $[\bar{u}, \tau^b]^T$, where $\bar{u} > 0$ is the net thrust from all rotors in the direction of $-b_3$, and $\tau^b = [\tau^{b1}, \tau^{b2}, \tau^{b3}]^T$ are the torques created by the rotors about the three body frame axes.

It is important to mention that each of the vehicle configurations (quadcopter and hexacopter) transforms the rotors' thrusts and torques to the system input vector $[\bar{u}, \tau^b]^T$ differently. These transformations are assumed to be performed in the onboard flight controller, and consequently both configurations are represented by the same dynamics Equation (15). Hence, the proposed algorithm development is the same for both configurations.

5.2. Development of NMPH on a Drone Vehicle

The state and input vectors of the rigid-body dynamics presented in Equation (10) are

$$\begin{aligned} x &= \left[(p^n)^T, (v^n)^T, (\eta)^T, (\omega^b)^T\right]^T \in \mathbb{R}^{12}, \\ u &= \left[\bar{u}, (\tau^b)^T\right]^T \in \mathbb{R}^4. \end{aligned} \tag{16}$$

Using the roll-pitch-yaw (ϕ, θ, ψ) Euler angle parameterization of $R \in SO(3)$, the nonlinear control-affine representation of Equation (10) can be expressed as

$$\dot{x} = f(x) + \sum_{i=1}^{4} \mathfrak{g}_i(x)u_i \triangleq f(x) + G(x)u \tag{17}$$

where

$$f(x) = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ 0 \\ 0 \\ g \\ x_{10} + s_{x_7} t_{x_8} x_{11} + c_{x_7} t_{x_8} x_{12} \\ c_{x_7} x_{11} - s_{x_7} x_{12} \\ \frac{s_{x_7}}{c_{x_8}} x_{11} + \frac{c_{x_7}}{c_{x_8}} x_{12} \\ \left(\frac{J_2 - J_3}{J_1}\right) x_{11} x_{12} \\ \left(\frac{J_3 - J_1}{J_2}\right) x_{10} x_{12} \\ \left(\frac{J_1 - J_2}{J_3}\right) x_{10} x_{11} \end{bmatrix}, G(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -\frac{1}{m}(c_{x_7} s_{x_8} c_{x_9} + s_{x_7} s_{x_9}) & 0 & 0 & 0 \\ -\frac{1}{m}(c_{x_7} s_{x_8} s_{x_9} - s_{x_7} c_{x_9}) & 0 & 0 & 0 \\ -\frac{1}{m} c_{x_7} c_{x_8} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{J_1} & 0 & 0 \\ 0 & 0 & \frac{1}{J_2} & 0 \\ 0 & 0 & 0 & \frac{1}{J_3} \end{bmatrix}$$

where $s_{(\cdot)} = \sin(\cdot)$, $c_{(\cdot)} = \cos(\cdot)$, and $t_{(\cdot)} = \tan(\cdot)$.

In order to make the system in Equation (17) state feedback linearizable, the state vector is augmented with two additional states, which are the thrust $x_{13} = \bar{u}$ and its rate $x_{14} = \dot{\bar{u}}$, and the thrust is replaced by $\ddot{\bar{u}}$ in the input vector [9]. Moreover, integral states ζ are added to the system dynamics to compensate for unmodeled external disturbances which affect the control system and NMPH performances. The proposed extension of the system is presented in Equations (18) and (19).

$$\begin{aligned} x &= \left[(p^n_{3\times1})^T, (v^n_{3\times1})^T, (\eta_{3\times1})^T, (\omega^b_{3\times1})^T, \bar{u}, \dot{\bar{u}}, (\zeta^{p^n}_{3\times1})^T, \zeta^{\psi} \right]^T \in \mathbb{R}^{18} \\ u &= \left[\ddot{\bar{u}}, (\tau^b_{3\times1})^T \right]^T \in \mathbb{R}^4 \end{aligned} \tag{18}$$

$$\dot{x} = \bar{f}(x) + \bar{G}(x) u \tag{19}$$

where,

$$\bar{f}(x) = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ -\frac{1}{m}(c_{x_7} s_{x_8} c_{x_9} + s_{x_7} s_{x_9}) x_{13} \\ -\frac{1}{m}(c_{x_7} s_{x_8} s_{x_9} - s_{x_7} c_{x_9}) x_{13} \\ g - \frac{1}{m} c_{x_7} c_{x_8} x_{13} \\ x_{10} + s_{x_7} t_{x_8} x_{11} + c_{x_7} t_{x_8} x_{12} \\ c_{x_7} x_{11} - s_{x_7} x_{12} \\ \frac{s_{x_7}}{c_{x_8}} x_{11} + \frac{c_{x_7}}{c_{x_8}} x_{12} \\ \left(\frac{J_2 - J_3}{J_1}\right) x_{11} x_{12} \\ \left(\frac{J_3 - J_1}{J_2}\right) x_{10} x_{12} \\ \left(\frac{J_1 - J_2}{J_3}\right) x_{10} x_{11} \\ x_{14} \\ 0 \\ x_1 \\ x_2 \\ x_3 \\ x_9 \end{bmatrix}, \quad \bar{G}(x) = \begin{bmatrix} & \mathbf{0}_{9\times4} & \\ 0 & \frac{1}{J_1} & 0 & 0 \\ 0 & 0 & \frac{1}{J_2} & 0 \\ 0 & 0 & 0 & \frac{1}{J_3} \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ & \mathbf{0}_{4\times4} & \end{bmatrix}$$

Based on the transformation presented in Section 3.2, the four smooth functions, which represent the linearizing outputs, are $\{\varphi_1(x) = x_1,\ \varphi_2(x) = x_2,\ \varphi_3(x) = x_3,\ \varphi_4(x) = x_9\}$, and the system transformation into linear canonical form can be written as

$$\begin{aligned} \dot{z} &= A_c\, z + B_c\, v\,, \quad & z \in \mathbb{R}^{18},\ v \in \mathbb{R}^4 \\ \xi &= C_c\, z\,, & \xi \in \mathbb{R}^4 \end{aligned} \tag{20}$$

where,

$$z = \left[\varphi_1, \ldots, L_f^{r_1-1}\varphi_1, \ \ldots \ , \varphi_m, \ldots, L_f^{r_m-1}\varphi_m\right]^T$$
$$= \left[x_{15}, x_1, x_4, \dot{x}_4, \ddot{x}_4, \ x_{16}, x_2, x_5, \dot{x}_5, \ddot{x}_5, \ x_{17}, x_3, x_6, \dot{x}_6, \ddot{x}_6, \ x_{18}, x_9, \dot{x}_9\right]^T$$

$$\dot{z} = \left[z_2, z_3, z_4, z_5, v_1, z_7, z_8, z_9, z_{10}, v_2, z_{12}, z_{13}, z_{14}, z_{15}, v_3, z_{17}, z_{18}, v_4\right]^T$$

The domain for a non-singular solution is $\{\bar{u} \neq 0, \ -\frac{\pi}{2} < \phi < \frac{\pi}{2}, \ -\frac{\pi}{2} < \theta < \frac{\pi}{2}\}$, which is found by the determinant of matrix $D(x)$ being $\det D(x) = -\frac{\bar{u}^2 \cos(\phi)}{m^3 J_1 J_2 J_3 \cos(\theta)}$. The domain limits shown above are included within the constraints of the optimization problem in NMPH (8).

Finally, the feedback linearization control inputs are found using Equation (13), which are

$$\begin{aligned}
\ddot{u} =& m\, c_{x_9} s_{x_7} v_2 - m\, s_{x_7} s_{x_9} v_1 - m\, c_{x_7}(s_{x_8} c_{x_9} v_1 + s_{x_8} s_{x_9} v_2 + c_{x_8} v_3) \\
&+ x_{13}\left(x_{10}^2 + \ x_{11}^2\right) \\
\tau^{b1} =& \frac{mJ_1}{x_{13}} s_{x_7}(s_{x_8} c_{x_9} v_1 + s_{x_8} s_{x_9} v_2 + c_{x_8} v_3) + \frac{mJ_1}{x_{13}} c_{x_7}(c_{x_9} v_2 - s_{x_9} v_1) - (J_2 - J_3) x_{12} x_{11} \\
&+ \frac{J_1}{x_{13}}(x_{11} x_{12} x_{13} - \ 2x_{10} x_{14}) \\
\tau^{b2} =& -\frac{mJ_2}{x_{13}}\, c_{x_8}(c_{x_9} v_1 + s_{x_9} v_2) + \frac{mJ_2}{x_{13}} s_{x_8} v_3 - \frac{J_2}{x_{13}}(x_{10} x_{12} x_{13} + \ 2x_{11} x_{14}) \\
&+ (J_1 - J_3) x_{12} x_{10} \\
\tau^{b3} =& -\frac{1}{c_{x_7} c_{x_8} x_{13}}\Big[2J_3\left(2x_{12} x_{11} c_{x_7}^2 + s_{x_7} x_{11}^2 - x_{12}^2 c_{x_7} - x_{12} x_{11}\right) x_{13} s_{x_8} \\
&+ \left((J_1 - J_2 + J_3) x_{10} x_{11} x_{13} c_{x_7} + J_3 s_{x_7}(m s_{x_8} v_3 - 2x_{10} x_{12} x_{13} - 2x_{11} x_{14})\right) c_{x_8} \\
&- J_3\left(m\, s_{x_7}(c_{x_9} v_1 + s_{x_9} v_2) + \ x_{13} v_4\right) c_{x_8}^2\Big]
\end{aligned} \tag{21}$$

where the feedback inputs are selected as follows,

$$v := \left\{ v_4 = \sum_{i=16}^{18} k_i e_{z_i} \ , \ \ v_j = \sum_{i=5j-4}^{5j} k_i e_{z_i} \ : \ j = 1, 2, 3 \right\} \tag{22}$$

and the error e_{z_i} is defined as the difference between the desired and the actual feedback state $e_{z_i} = z_i^{ref} - z_i \ , \ i = 1, \ldots, n$.

As presented in Equations (15) and (21), the drone's behavior is described by its nonlinear system dynamics and the feedback linearization control. The optimization within NMPH exploits their integration to enhance the performance of generating the reference trajectory. The continuous-time NMPH presented in Algorithm 1 is solved using a multiple shooting optimization technique. The solver used in our work, ACADO [40], discretizes the system dynamics, control law, and inequality constraints over the prediction horizon at each time instant t_n. Figure 8 shows the optimization process from the problem formulation to the trajectory generation.

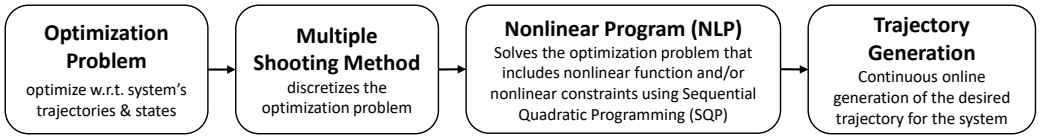


Figure 8. NMPH Optimization Process. The non-convex optimization problem can be solved iteratively using Sequential Quadratic Programming (SQP) [41]. In SQP, the problem is divided into a sequence of subproblems, each of which solves a quadratic objective function subject to linearized constraints [42].

The integration of the feedback linearization within the NMPH aims to reduce the non-convexity of the optimization problem. A perfect model of the system dynamics would allow the closed-loop form to be represented by a linear canonical form as shown in Equation (20), for which the feasibility and stability of the optimized solution are guaranteed and the computational power needed to solve the optimization problem is greatly reduced over the non-convex case. However, even an imperfect model still reduces the non-convexity of the optimization problem as compared to working directly with the (nonlinear) plant dynamics as in standard NMPC.

6. Experimental Results

In this section, simulation and a preliminary real-time hardware flight test are presented to evaluate and validate the proposed design on quadcopter and hexacopter vehicles while operating in GPS-denied environments.

The algorithms are implemented within the Robot Operating System (ROS) [43], a Linux-based system that handles communication between the individual subsystems and the vehicle. The ACADO Toolkit [40] is used for optimization. The optimization problem is programmed in a self-contained C++ environment within this toolkit, then a real-time nonlinear solver is generated to run the optimizations online. The resulting code can be compiled and run within ROS, which also handles the communication between the solver and the vehicle, either simulated or real [44]. The NMPH optimization problem (4)–(9) was written in C++ code using ACADO, then automatically converted into a highly efficient C code that is able to solve the optimization problem in real-time.

6.1. Simulation Results

In order to test the proposed approach before implementing it on a real drone, the quadcopter drone vehicle is simulated within AirSim [45]. AirSim is an open-source simulator that provides photo-realistic environments plus a physics engine to enable performing lifelike simulations.

All frameworks and the AirSim simulator are run in ROS on an Intel Core i7-10750H CPU @ 2.60–5.00 GHz equipped with the Nvidia GeForce RTX 2080 Super Max-Q GPU. The prediction horizon for NMPH was set to $T = 8$ s using a sampling time of 0.2 s, which was found satisfactory for trajectory generation. The cost function weights W_x, $W_{\tilde{\xi}}$, and W_T were adjusted heuristically to ensure a balanced trajectory generation performance towards the terminal setpoint.

The drone's measured pose and pointcloud information are obtained from the AirSim simulator and sent to the motion planner. The global graph-based and local NMPH planners generate reference trajectories for the vehicle, which are forwarded to AirSim for trajectory tracking purposes. RViz, the 3D visualization tool for ROS, is used to monitor and visualize the simulation process. Figure 9 shows the ROS network architecture of the nodes and topics employed in performing our simulation.

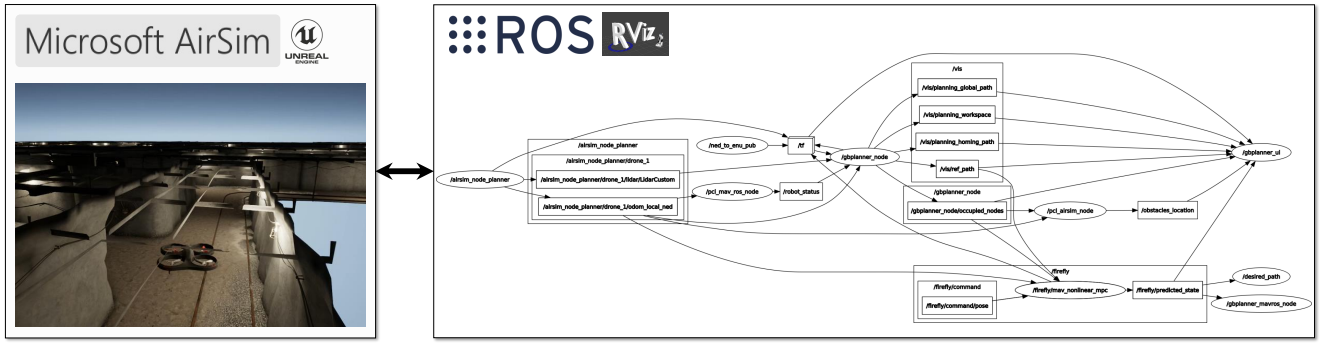
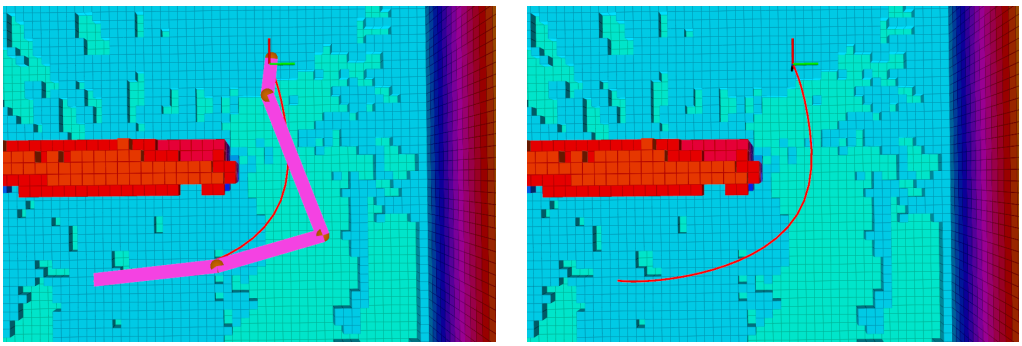


Figure 9. Simulation Architecture.

Different simulation results are now presented to evaluate the performance of the proposed approach on a quadcopter drone navigating autonomously through a previously unexplored, GPS-denied underground environment available within the AirSim simulator. The motion planner design illustrated in Figure 3 is implemented for this purpose. Within AirSim, the virtual quadcopter is equipped with a customized 32-channel 360° scanning Lidar sensor with a 45° vertical field of view, 10 Hz rotation rate, and 50 m range. The pointcloud data plus the vehicle pose are acquired and used to build a volumetric map of the environment and locate the vehicle within it.

As discussed in Section 4.3, the graph-based planning algorithm guides the vehicle towards unexplored areas within the map and provides vertices as terminal setpoints x_{ss} for the NMPH local path planner. The design's robustness is increased by implementing the Obstacle Avoidance and Path Guidance algorithms proposed in Sections 4.4.2 and 4.4.3, respectively. Finally, the generated reference path from the motion planner is sent to the AirSim quadcopter for tracking. The NMPH continues updating the path during the vehicle's movement toward a setpoint. This allows to avoid dynamic obstacles and improves the tracking performance. Once the vehicle reaches a setpoint, the planning process is repeated until the environment is fully explored or the mission is interrupted by the operator.

Figure 10 shows the paths generated by the graph-based and the NMPH path planners. The NMPH is seen to provide a smooth and optimal path as compared to the sharp-corner path generated by the graph-based planner. Moreover, the NMPH keeps updating the path dynamically from the start to the terminal point at a rate of up to 100 Hz, while the graph-based planner generates only one path between the two points. To reduce computational load, the NMPH algorithm rate is set to 5 Hz, which was found to be suitable in generating continuous and smooth paths in the environment. This rate of path regeneration also provides good path following performance in the presence of static and dynamic obstacles.



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

Figure 10. Motion Planner. (**a**) Path planning using graph-based approach (pink) and NMPH algorithm (red). (**b**) Optimal path using NMPH algorithm.

A portion of the overall tracked trajectory between multiple vertices using the NMPH algorithm can be seen in Figure 11. Respecting the system dynamics provides smooth flight paths and thus reduces power consumption caused by abrupt changes in the trajectory.

The DLOM generates a continuously changing obstacle map modeled by virtual spheres as depicted in Figure 12. As discussed in Sections 4.4.1 and 4.4.2, mapped obstacles are represented by (continuously updated) inequality constraints within the optimization problem. The Obstacle Avoidance Algorithm helps in creating and updating a path that avoids the obstacles as shown in Figure 12.

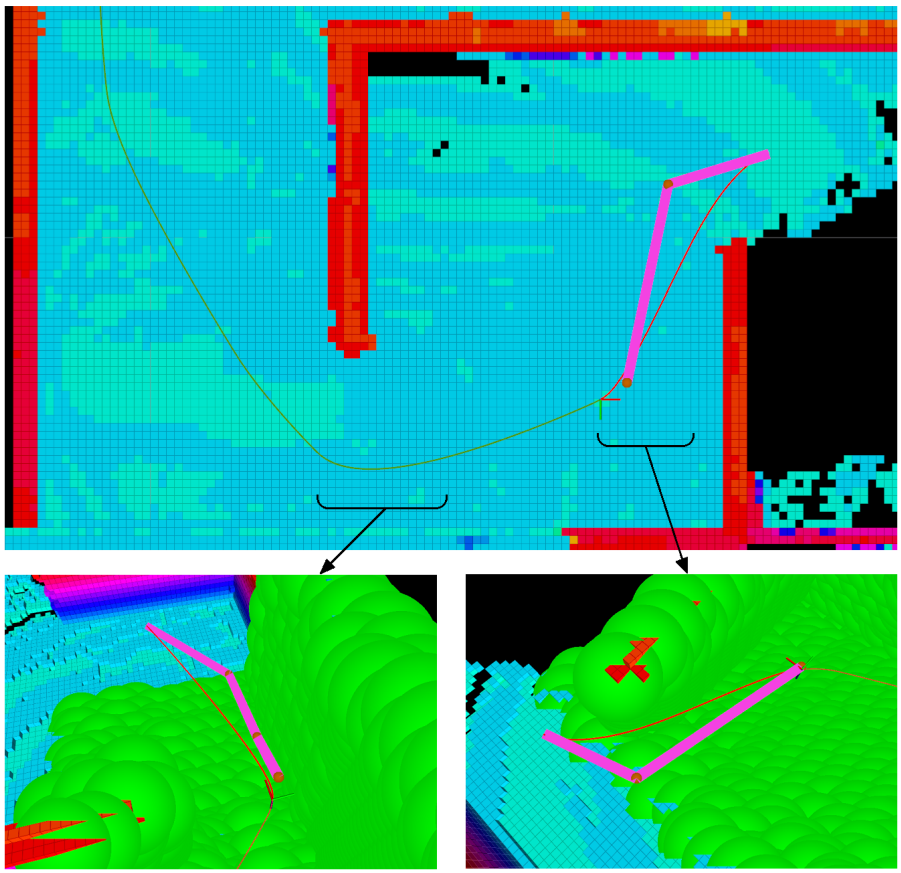
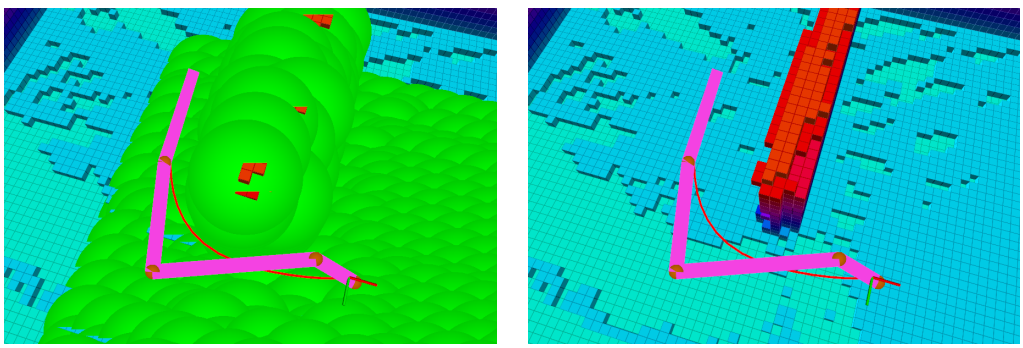


Figure 11. Illustration of trajectory generation and tracking. The green path is the trajectory of the vehicle, and the red path is the future reference path.



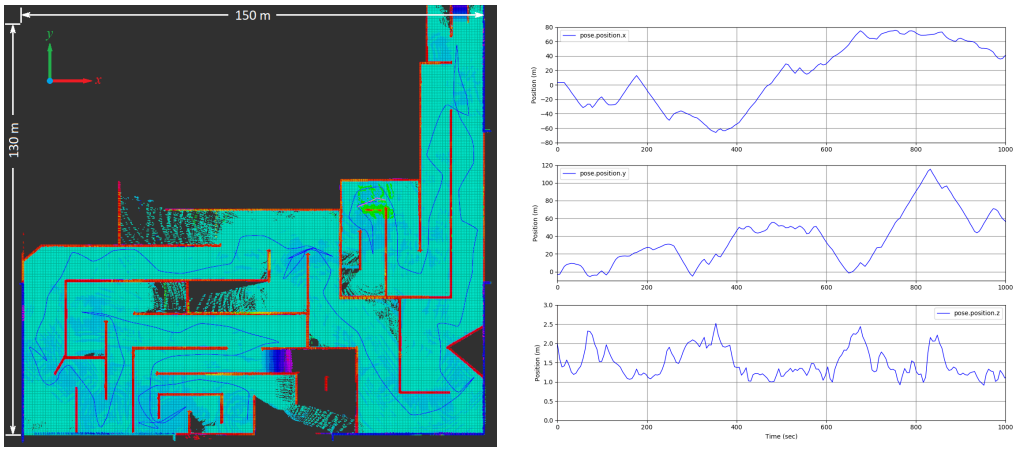
(**a**) (**b**)

Figure 12. Dynamic Local Obstacle Mapping and Avoidance. In (**a**) the DLOM is made visible while in (**b**) it is hidden.

In the next simulation test, the quadcopter autonomously navigates an unexplored GPS-denied environment. Figure 13 shows the exploration mission performed by the quadcopter. The drone travels a total distance of 774.5 m while following smooth trajectories generated by our proposed algorithm. Meanwhile, the graph-based planner generated paths with a total length of 993.1 m for the same exploration mission. Table 1 and Figure 14 offer a mission performance comparison between using the graph-based planner solo versus using the graph-based planner integrated with our NMPH approach in terms of exploration time, average computation time of the generated paths, path lengths between terminal vertices, and average and total length of the generated paths. This comparison shows the impact of using the NMPH algorithm for reducing power consumption, total mission time, and unwanted abrupt motions while following the generated reference paths.

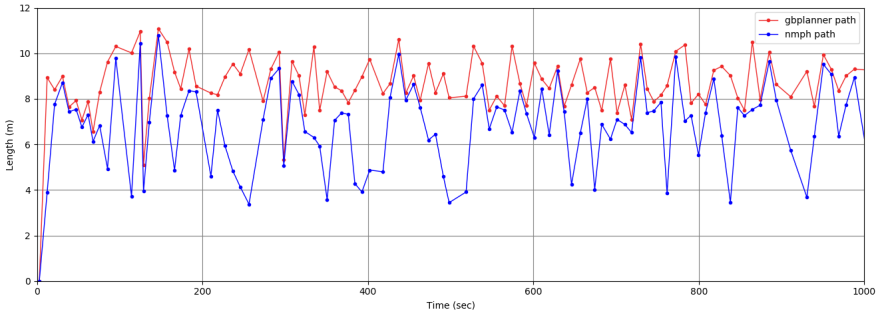
Table 1. Comparison between Graph-based and Graph-based-plus-NMPH approaches to path planning.

	Total Length of the Generated Paths	Average Path Length (between Terminal Points)	Average Path Computation Time	Exploration Time	Continuous Path Generation
Graph-based	993.1 m	8.79 m	733 ms	1327 s	No
Graph-based-plus-NMPH	774.5 m	6.86 m	4.34 ms	1035 s	Yes

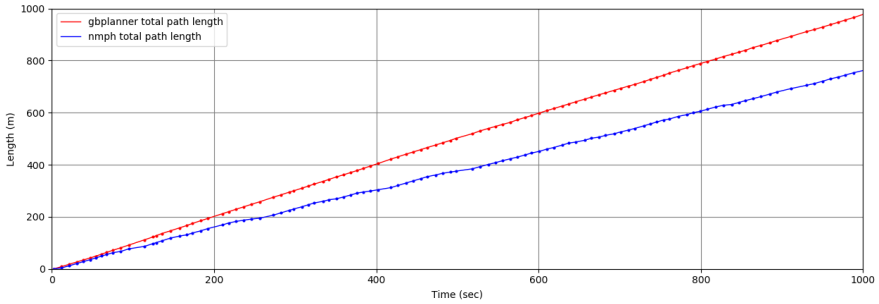


(**a**) (**b**)

Figure 13. Autonomous navigation and exploration in GPS-denied environment. The vehicle travelled a total distance of 774.5 m in about 1035 s. (**a**) Overhead visualization of exploration path through environment. (**b**) Plot of vehicle positions (x,y,z) versus time.



(**a**)



(**b**)

Figure 14. Comparison of path lengths between graph-based planner and our proposed NMPH path planner. (**a**) Path length between stabilization points. (**b**) Total length of generated paths.

In the final simulation test, obstacle avoidance for a moving object is tested while the drone navigates through the environment. This is shown in Figure 15, where the continuous regeneration of the path by the NMPH algorithm enables the drone to safely navigate to the stabilization point.

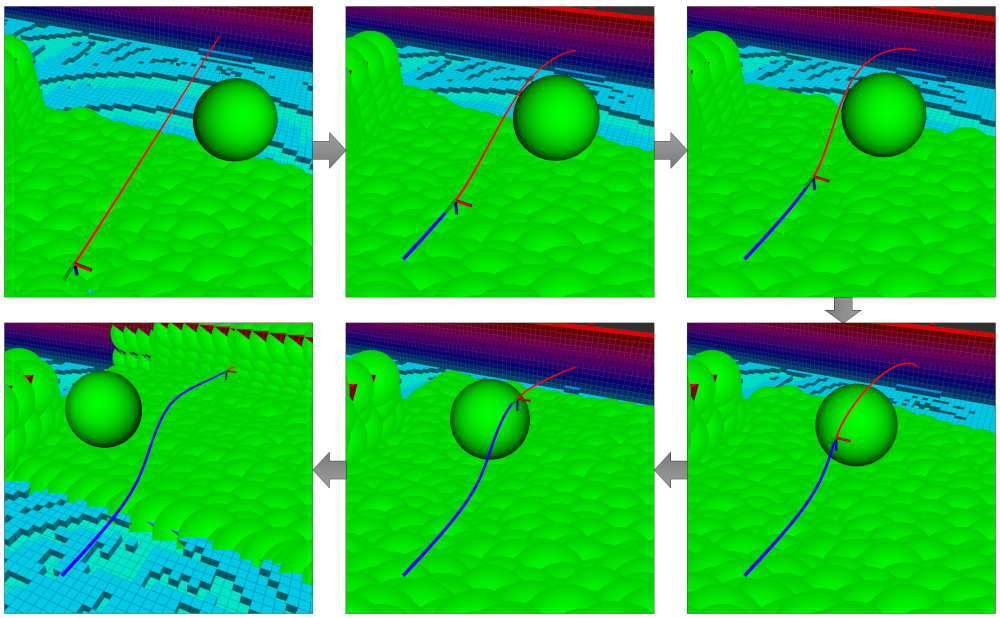


Figure 15. Obstacle avoidance for a moving object. The object (sphere) is moving to the left. The NMPH regenerates the red path continuously to avoid the object. The blue curve represents the flight trajectory of the drone.

6.2. Preliminary Real-Time Flight Test Results

For real-time hardware testing, a FlameWheel F550 hexacopter (DJI Technology, Shenzhen, China) was used. The vehicle is equipped with a Pixhawk 2.1 autopilot control board running the PX4 flight control software [46], and an onboard Jetson TX2 (NVidia, Santa Clara, CA, USA) computer running ROS. The communication between ROS and PX4 is established through MAVLink. A RealSense T265 stereo camera and a RealSense L515 LiDAR camera (Intel, Santa Clara, CA, USA) are mounted on the hexacopter to provide pose and RGB-D pointcloud data, respectively.

The preliminary flight test evaluates the path generation performance of NMPH running onboard a real drone, whose Jetson TX2 has lower computational power than the computer used in simulation. Also, local trajectory tracking and the functionality of the motion planner are tested in this experiment, as shown in Figure 16. Note that hardware flights in large-scale areas will be performed and evaluated in future work.

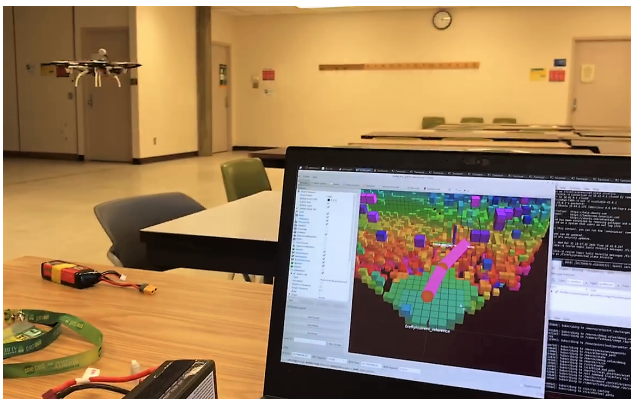
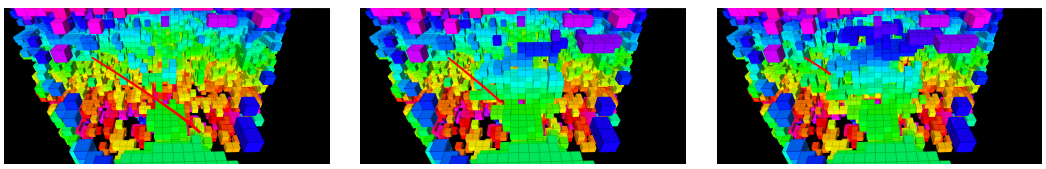


Figure 16. Preliminary flight test.

In the current setup, the optimization solver within NMPH was able to provide continuous regeneration of the reference trajectories at rates approaching 70 Hz. This rate was reduced to 5 Hz to minimize the computational load on the onboard system. The graph-based motion planner was also tested by generating terminal points for NMPH. The latter sent the predicted reference trajectories to the onboard PX4, and the hexacopter was able to follow them. Figure 17 shows the hexacopter following a continuously regenerated reference path to a terminal vertex.



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

Figure 17. Hardware flight testing: successive captures of screen (**a**–**c**) displaying sensed environment (blocks) and planned path (red line) of drone.

7. Conclusions

This paper presented a methodological motion planning approach for drone exploration in GPS-denied environments, which integrates our recently proposed NMPH path planning approach with a graph-based planner. The NMPH formulation employs the nonlinear system dynamics model with feedback linearization control inside an online optimization-based process to generate feasible, optimal and smooth reference trajectories for the vehicle. The performance of the overall motion planner is increased by introducing methods for robust path generation and dynamic obstacle mapping and avoidance.

The developed motion planner was evaluated through a series of simulation flights as well as a real-time hardware flight test to validate the performance of the proposed design on quadcopter and hexacopter drones navigating the environment. The results show the ability our algorithm to improve motion planning performance over conventional techniques and generate smooth and safe flight trajectories in a computationally efficient way.

Future work will include testing the proposed motion planning methodology inside large-scale GPS-denied environments such as subterranean mines.

Author Contributions: Conceptualization, Y.A.Y. and M.B.; methodology, Y.A.Y.; software, Y.A.Y.; validation, Y.A.Y.; formal analysis, Y.A.Y.; investigation,Y.A.Y.; resources, M.B.; data curation, Y.A.Y.; writing—original draft preparation, Y.A.Y.; writing—review and editing, M.B.; visualization, Y.A.Y.; supervision, M.B.; project administration, M.B.; funding acquisition, M.B. Both authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NSERC Alliance-AI Advance Program grant number 202102595. The APC was funded by NSERC.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Acknowledgments: The authors wish to thank Selina Leveugle and Ali Muneer for their extensive work on the hardware drone platform.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. LaValle, S.M. *Planning Algorithms*; Cambridge University Press: New York, USA, 2006.
2. Sozzi, A.; Bonfè, M.; Farsoni, S.; De Rossi, G.; Muradore, R. Dynamic motion planning for autonomous assistive surgical robots. *Electronics* **2019**, *8*, 957. [CrossRef]
3. Paden, B.; Čáp, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intell. Veh.* **2016**, *1*, 33–55. [CrossRef]
4. Ahmed, S.M.; Tan, Y.Z.; Lee, G.H.; Chew, C.M.; Pang, C.K. Object detection and motion planning for automated welding of tubular joints. In Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, 9–14 October 2016; pp. 2610–2615.
5. Dang, T.; Mascarich, F.; Khattak, S.; Papachristos, C.; Alexis, K. Graph-based path planning for autonomous robotic exploration in subterranean environments. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 4–8 November 2019; pp. 3105–3112.
6. Quan, L.; Han, L.; Zhou, B.; Shen, S.; Gao, F. Survey of UAV motion planning. *IET Cyber Syst. Robot.* **2020**, *2*, 14–21. [CrossRef]
7. Corke, P. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 2nd ed.; Springer Tracts in Advanced Robotics; Springer: Cham, Switzerland, 2017; Volume 118.
8. Grüne, L.; Pannek, J. *Nonlinear Model Predictive Control: Theory and Algorithms*; Springer: London, UK , 2017.
9. Al Younes, Y.; Barczyk, M. Nonlinear Model Predictive Horizon for Optimal Trajectory Generation. *Robotics* **2021**, *10*, 90. [CrossRef]
10. Isidori, A. *Nonlinear Control Systems*, 3rd ed.; Springer: London, UK , 1995.
11. Marino, R.; Tomei, P. *Nonlinear Control Design: Geometric, Adaptive, and Robust*; Prentice Hall: London, UK, 1995.
12. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Introduction to Algorithms*, 3rd ed.; MIT Press: Cambridge, MA, USA, 2009.
13. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
14. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
15. Koenig, S.; Likhachev, M. Fast replanning for navigation in unknown terrain. *IEEE Trans. Robot.* **2005**, *21*, 354–363. [CrossRef]
16. Al-Mutib, K.; AlSulaiman, M.; Emaduddin, M.; Ramdane, H.; Mattar, E. D* lite based real-time multi-agent path planning in dynamic environments. In Proceedings of the Third International Conference on Computational Intelligence, Modelling & Simulation, Langkawi, Malaysia, 20–22 September 2011; pp. 170–174.
17. Likhachev, M.; Gordon, G.J.; Thrun, S. ARA*: Anytime A* with provable bounds on sub-optimality. In Proceedings of the 16th International Conference on Neural Information Processing Systems, Whistler, BC, Canada, 9–11 December 2003; pp. 767–774.

18. Dolgov, D.; Thrun, S.; Montemerlo, M.; Diebel, J. Path planning for autonomous vehicles in unknown semi-structured environments. *Int. J. Robot. Res.* **2010**, *29*, 485–501. [CrossRef]
19. Yang, Y.; Pan, J.; Wan, W. Survey of optimal motion planning. *IET Cyber-Syst. Robot.* **2019**, *1*, 13–19. [CrossRef]
20. Siméon, T.; Laumond, J.P.; Nissoux, C. Visibility-based probabilistic roadmaps for motion planning. *Adv. Robot.* **2000**, *14*, 477–493. [CrossRef]
21. LaValle, S.M.; Kuffner, J.J. Rapidly-exploring random trees: Progress and Prospects. In *Algorithmic and Computational Robotics: New Directions*; Donald, B.R., Lynch, K.M., Rus, D., Eds.; CRC Press: Boca Raton, FL, USA, 2001; pp. 293–308.
22. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
23. Canny, J.; Rege, A.; Reif, J. An exact algorithm for kinodynamic planning in the plane. *Discret. Comput. Geom.* **1991**, *6*, 461–484. [CrossRef]
24. Ratliff, N.; Zucker, M.; Bagnell, J.A.; Srinivasa, S. CHOMP: Gradient optimization techniques for efficient motion planning. In Proceedings of the 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 12–17 May 2009; pp. 489–494.
25. Schulman, J.; Duan, Y.; Ho, J.; Lee, A.; Awwal, I.; Bradlow, H.; Pan, J.; Patil, S.; Goldberg, K.; Abbeel, P. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Robot. Res.* **2014**, *33*, 1251–1270. [CrossRef]
26. Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; Schaal, S. STOMP: Stochastic trajectory optimization for motion planning. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 4569–4574.
27. Atkeson, C.G. Using local trajectory optimizers to speed up global optimization in dynamic programming. In Proceedings of the 6th International Conference on Neural Information Processing Systems, Denver, CO, USA, 29 November–2 December 1993; pp. 663–670.
28. Perez, A.; Platt, R.; Konidaris, G.; Kaelbling, L.; Lozano-Perez, T. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA), Saint Paul, MN, USA, 14–18 May 2012; pp. 2537–2542.
29. Nolte, M.; Rose, M.; Stolte, T.; Maurer, M. Model predictive control based trajectory generation for autonomous vehicles—An architectural approach. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Redondo Beach, CA, USA, 11–14 June 2017; pp. 798–805.
30. Andersson, O.; Ljungqvist, O.; Tiger, M.; Axehill, D.; Heintz, F. Receding-horizon lattice-based motion planning with dynamic obstacle avoidance. In Proceedings of the 2018 IEEE Conference on Decision and Control (CDC), Miami Beach, FL, USA, 17–19 December 2018; pp. 4467–4474.
31. Andersson, O.; Wzorek, M.; Rudol, P.; Doherty, P. Model-predictive control with stochastic collision avoidance using bayesian policy optimization. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 4597–4604.
32. Oleynikova, H.; Taylor, Z.; Fehr, M.; Siegwart, R.; Nieto, J. Voxblox: Incremental 3D euclidean signed distance fields for on-board mav planning. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 1366–1373.
33. Steinbrücker, F.; Sturm, J.; Cremers, D. Volumetric 3D mapping in real-time on a CPU. In Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 31 May–7 June 2014; pp. 2021–2028.
34. Curless, B.; Levoy, M. A volumetric method for building complex models from range images. In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH), New Orleans, LA, USA, 4–9 August 1996; pp. 303–312.
35. Oleynikova, H.; Millane, A.; Taylor, Z.; Galceran, E.; Nieto, J.; Siegwart, R. Signed distance fields: A natural representation for both mapping and planning. In Proceedings of the RSS 2016 Workshop: Geometry and Beyond-Representations, Physics, and Scene Understanding for Robotics, Ann Arbor, MI, USA, 19 June 2016.
36. Nießner, M.; Zollhöfer, M.; Izadi, S.; Stamminger, M. Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. Graph.* **2013**, *32*, 169. [CrossRef]
37. Karaman, S.; Frazzoli, E. Sampling-based motion planning with deterministic μ-calculus specifications. In Proceedings of the Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference, Shanghai, China, 16–18 December 2009; pp. 2222–2229.
38. Moore, A.W. An Introductory Tutorial on kd-Trees, 1991. Available online: https://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1991_1/moore_andrew_1991_1.pdf (accessed on 17 August 2021)
39. Murray, R.M.; Li, Z.; Sastry, S.S.; Sastry, S.S. *A Mathematical Introduction to Robotic Manipulation*; CRC Press: Boca Raton, FL, USA, 1994.
40. Houska, B.; Ferreau, H.; Diehl, M. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optim. Control Appl. Methods* **2011**, *32*, 298–312. [CrossRef]
41. Boggs, P.T.; Tolle, J.W. Sequential quadratic programming. *Acta Numer.* **1995**, *4*, 1–51. [CrossRef]
42. Ferreau, H.; Kirches, C.; Potschka, A.; Bock, H.; Diehl, M. qpOASES: A parametric active-set algorithm for quadratic programming. *Math. Program. Comput.* **2014**, *6*, 327–363. [CrossRef]
43. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009.

44. Kamel, M.; Stastny, T.; Alexis, K.; Siegwart, R. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In *Robot Operating System (ROS)*; Springer: Cham, Switzerland, 2017; pp. 3–39.
45. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*; Hutter, M., Siegwart, R., Eds.; Springer: Cham, Switzerland, 2018; pp. 621–635.
46. Meier, L.; Honegger, D.; Pollefeys, M. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 6235–6240.