A Scalable Hierarchical Path Planning Technique for Autonomous Inspections with Multicopter Drones

Michele Bolognini and Lorenzo Fagiano *†

Abstract

Multicopter drones equipped with cameras can perform rapid inspections of large buildings, including those with hard to reach features, like bridge pylons. Drones can be made autonomous by providing them with a method to choose a path that maximizes the collected information during the limited flight time allowed by the battery. It is therefore crucial to optimize the trajectories to minimize inspection time. The problem of finding an approximately optimal path passing through a series of desired inspection points in a three-dimensional environment with obstacles is considered. A hierarchical approach is proposed, where the space containing the inspection points is partitioned into different regions and multiple instances of the TSP (Travelling Salesman Problem) are solved, decreasing the overall complexity. An extended graph is used in the TSP, in order to tackle the problem of collision avoidance while planning the trajectory between point pairs. This approach leads to an efficient and scalable method capable of avoiding obstacles, and significantly reduces the time needed to find an optimal path with respect to non-hierarchical methods. Simulation results highlight these features.

1 Introduction

The versatility and ever decreasing cost of drones have recently pushed their use in the construction and maintenance of buildings. Increased autonomy benefit inspection tasks by making them easier to perform, more repeatable and safer than those carried out by humans, which often entail climbing structures or rely on a human pilot to guide the drone. To exploit the full potential of UAVs (Unmanned Aerial Vehicles) it is necessary to systematically elaborate trajectories, that maximize the collected information while minimizing flight time and energy consumption. We address this problem, assuming that a 3D

*This research has been supported by the Italian Ministry of University and Research (MIUR) under the PRIN 2017 grant n. 201732RS94 "Systems of Tethered Multicopters"

†Michele Bolognini and Lorenzo Fagiano are with Department of Electronics, Information and Bioengineering, Politecnico di Milano University, Piazza Leonardo da Vinci 32, Italy. michele.bolognini@polimi.it, lorenzo.fagiano@polimi.it

model of the building to inspect is available and that a series of points in space are given where the drone must travel to carry out some task, for example take a picture of a structure.

This research constitutes one of the first building blocks in a larger project, whose aim is to develop a system composed of tethered [9] and non-tethered multicopters, specialized in performing inspections of the built environment. The final aim is to make monitoring and inspection tasks on buildings and infrastructure as autonomous as possible, including visual inspection of surfaces and possibly contact measurements.

1.1 State of the Art

The topic of path planning for autonomous vehicles has been addressed in different scenarios. For the particular goal of inspection and exploration, several techniques have been studied, most of which entail on-line path planning [1, 3] and/or on-line update and correction of a pre-computed trajectory [2, 7]. The latter is justified when the environment is either not completely known or uncertain. The problem of finding and explicitly describing an optimal trajectory in space and the related control inputs for the drone is difficult, because it requires the solution of multiple non-linear differential equations. The presence of obstacles makes the feasible space non-convex, so that the problem becomes extremely hard to solve analytically and computationally expensive to solve numerically. For these reasons, sub-optimal heuristics are used. One common approach is to partition the 3D space into voxels, [1], classifying them as occupied or not based on the presence of obstacles in them. The problem of navigating such simplified representation of space is then solved by formulating trajectories that span adjacent unoccupied voxels, ensuring no collision with obstacles takes place. Another approach is to distribute points in the 3D space such that they lie outside obstacles, then connect them to form a graph, based on their distance and the presence of obstacles between them. Different algorithms exist to connect points in different topologies to ensure desirable properties. One such algorithm is RRT (Rapidly exploring Random Trees) [12], which builds a tree of feasible paths connecting the nodes, that is then exploited to make the computation of trajectories faster [3, 4]. Another solution, employed in this paper, is to generate a graph and then use some algorithm to find paths and cycles through it, for example by formulating a TSP or using other well-known graph search techniques. The weights assigned to the graph connections can be tuned to represent the physical nature of the problem and the limits of the deployed system. In [5, 6], for example, they are chosen as the time it takes for a drone to travel from one point to the other, calculated as the solution of a boundary value problem formulated on a simplified model of the aircraft. This discretization of continuous 3D space significantly reduces the complexity of the initial problem by decoupling the search for an optimal trajectory from the issue of avoiding obstacles. Both approaches can be combined with additional heuristics to obtain smoother and faster trajectories. In [8], for example, the authors propose a combination of TSP and a genetic algorithm to calculate the optimal visiting order of a number of points and subsequently plan a smooth trajectory that avoids forbidden regions. The authors of [15] study a simpli-

fied version of the problem where the environment is supposed to be modular and they provide both an algorithm to efficiently solve the multi-agent version, and a theoretical lower bound on the sub-optimality of the obtained solutions. Unfortunately, no efficient algorithm for solving TSPs exists, and a guarantee of optimality is only available through enumeration of all possible solutions. In the worst case scenario the number of possible solutions for a graph of n nodes is $n!$, which quickly becomes impractical even for n in the order of tens. This poses a significant constraint on approaches that do not employ some heuristic to speed up the search, at the cost of optimality.

1.2 Main Contribution

The primary contribution of this research is a scalable algorithm that calculates a near-optimal, feasible path for a drone to visit a set of given points in space. It employs a hierarchical structure to significantly limit the computational complexity, while also guaranteeing an obstacle-free trajectory. The path is obtained as a series of waypoints to be sent to the drone, which is position-controlled, i.e. it tracks position set-points. The weight assigned to each edge of the graph corresponds to the actual distance an UAV would have to travel between its vertices. This means that the distance covered for avoiding possible obstacles is accounted for, instead of the simple geometric distance. The proposed algorithm is tested in simulation on a realistic bridge inspection task with 180 inspection points to be visited, distributed around the bridge pylons.

2 Problem description

We use bold symbols to indicate vectors. The T superscript is the matrix transposition operator. All vectors are intended as columns unless otherwise specified. For a given point $\boldsymbol{p} \in \mathbb{R}^n$ and convex set $O \subseteq \mathbb{R}^n$, the distance operator $\delta(\boldsymbol{p}, O)$ is defined as:

$$\delta(\boldsymbol{p}, O) = \min_{\boldsymbol{x} \in O} \|\boldsymbol{p} - \boldsymbol{x}\|_2,$$

while for two points $\boldsymbol{p}_i$, $\boldsymbol{p}_j \in \mathbb{R}^n$ the operator $\phi(\boldsymbol{p}_i, \boldsymbol{p}_j, O)$ is defined as:

$$\phi(\boldsymbol{p}_i, \boldsymbol{p}_j, O) = \begin{cases} 1 \text{ if } (\alpha \boldsymbol{p}_i + (1-\alpha)\boldsymbol{p}_j) \notin O, \, \forall \alpha \in [0,1] \\ 0 \text{ otherwise} \end{cases}$$

i.e. it returns a boolean indicating whether the segment connecting $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ is outside the set O. Without loss of generality, we consider a quadcopter drone for simulations. We adopt a rather standard model with six degrees of freedom, as described e.g. in [14].

2.1 Environment Model

We start by introducing the hypothesis that a reliable model of the 3D structure to be inspected is available. Since our work is aimed at developing a system that will be used to perform cyclic inspections, it is reasonable to suppose that such

a representation of the environment, updated at each inspection, is sufficient to avoid most obstacles, and that the presence of unexpected ones appearing during the flight is dealt with by a reactive feedback controller, using e.g. LiDAR measurement, as in [13]. The environment model includes a number $n_o \in \mathbb{N}$ of polytopic obstacles O_j, $j = 1, \ldots, n_o$, collected in the set $\mathcal{O}$. Their shapes and positions are known with respect to a global reference frame (X, Y, Z). In particular, the obstacles O_j are stored as polytopes in inequality form:

$$O_j = \{\boldsymbol{x} \in \mathbb{R}^3 \mid A_j \boldsymbol{x} \leq \boldsymbol{b}_j\} \quad j = 1, \ldots, n_o.$$

Where $A_j \in \mathbb{R}^{n_{c,j} \times 3}$, $\boldsymbol{b} \in \mathbb{R}^{n_{c,j}}$ are a matrix and a vector depending on the specific obstacle. In addition, a set P containing n_p inspection points is given

$$P = \{\boldsymbol{p}_i \in \mathbb{R}^3\} \quad i = 1, \ldots, n_p.$$

Such points represent positions that the drone mus visit to perform a certain inspection task, therefore none of them lies inside an obstacle. The set P is supposed to be provided by an external agent; the inspections points depend both on the kind of task to be carried out and on the desired output quality. Depending on the physical dimension of the structure and the number of in-

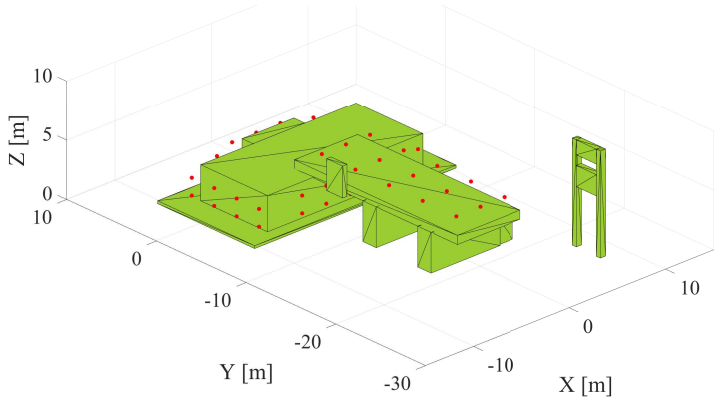

Fig. 1: A representation of an environment with obstacles (green shapes) and a few inspection points (red dots). In this case, a gas station is represented and obstacles approximate its shapes.

spection points, both sets are partitioned into a number m of subsets, to ease the computational load. •

* Mention orientation of drone too?

The lift forces and drag torques generated by the rotors are given by

$$\begin{array}{rcl} L_j(t) & = & b\,\Omega_j(t)^2, \quad j = 1, \ldots, 4 \\ T_j(t) & = & d\,\Omega_j(t)^2, \quad j = 1, \ldots, 4, \end{array} \tag{1}$$

where $t \in \mathbb{R}$ is the continuous time variable, b, d represent the thrust force and drag torque coefficients of the propellers, the index j indicates the j-th rotor

and Ω_j denotes its rotational speed, which is commanded by the flight controller via Electronic Speed Controllers (ESCs). Letting

$$\begin{array}{rcl} u_1(t) & = & \sum_{j=1}^{4} L_j(t) \\ u_2(t) & = & a_r\ (L_4 - L_2) \\ u_3(t) & = & a_r\ (L_3 - L_1) \\ u_4(t) & = & (T_2 + T_4) - (T_1 + T_3)\,, \end{array} \tag{2}$$

where a_r is the distance between the center of mass of the drone and the the rotor's hubs (assumed equal for the four rotors for simplicity), and applying Newton's law, we obtain the model equations

$$\begin{array}{rcl} \ddot{\mathbf{p}}(t) & = & \frac{1}{m_d} R(t)^T \begin{bmatrix} 0 \\ 0 \\ u_1(t) \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \\ \dot{p}(t) & = & \frac{I_y - I_z}{Ix} q(t) r(t) + \frac{u_2(t)}{Ix} - \frac{J_p}{Ix} q(t) \Omega_r(t) \\ \dot{q}(t) & = & \frac{I_z - I_x}{Iy} p(t) r(t) + \frac{u_3(t)}{Iy} + \frac{J_p}{Iy} p(t) \Omega_r(t) \\ \dot{r}(t) & = & \frac{I_x - I_y}{Iz} p(t) q(t) + \frac{u_4(t)}{Iz}. \end{array} \tag{3}$$

The first matrix equation describes the evolution of the drone's linear accelerations $\ddot{\mathbf{p}}(t)$ along the (X, Y, Z) inertial axes, through the rotation matrix $R(t)^T$ that translates local coordinates in global ones, for more detailed definitions, see [9]. g is the gravity acceleration, while $p(t), q(t)$ and $r(t)$ are the drone's angular speeds around, respectively, the x, y and z local axes of the drone. I_x, I_y, I_z are the elements of the matrix of inertia of the drone (assumed diagonal), J_p the moment of inertia of the propellers and $\Omega_r(t)$ a suitable linear combination of their rotational speeds. The drone is controlled with three nested feedback loops, so to track set-points of the form $\boldsymbol{p}_{ref}(t) = [x_r(t), y_r(t), z_r(t), \psi_r(t)]^T$, i.e. position and yaw angle references. The reader is referred to [9] for further details on the simulated model and the employed control scheme.

2.2 Problem Statement

The goal of the path-planning task is to plan the path that approximately minimizes the travel time required to visit all inspection points P, while avoiding the 3D obstacles in $\mathcal{O}$, starting from and returning to a given initial position, p^*. The result is a sequence S_{ref} of vectors $\boldsymbol{w}_i \in \mathbb{R}^4$, each representing a waypoint in the 3D space and a yaw reference:

$$S_{ref} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N\} \tag{4}$$

Such a sequence can be fed to the drone controller in order to simulate the related trajectory, using to a path following strategy detailed in the next section. We use simulations to validate the path planning and obstacle avoidance algorithms.

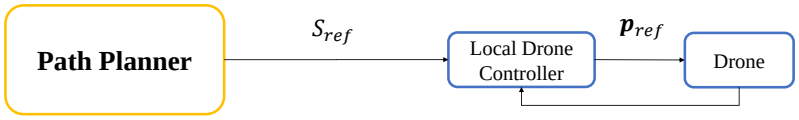

Fig. 2: A representation of the control scheme. Our research focuses on the highlighted path planner, an algorithm that produces a series of waypoints S_{ref}.

Given our specific parameter values, pertaining to a rather large drone platform with high inertia and with maximum speed bounded to a rather low value of $1.1m/s$, the simulated aircraft travels at constant maximum speed for most of the time, which in turn means that distance between points and actual travel time can be used interchangeably as cost metrics in the formulation of the problem. We chose geometric distance as metric because, contrary to travel time and required energy, it is immediately available given two points without any additional computation. Section 3.1 details how a series of waypoints is managed by the control system.

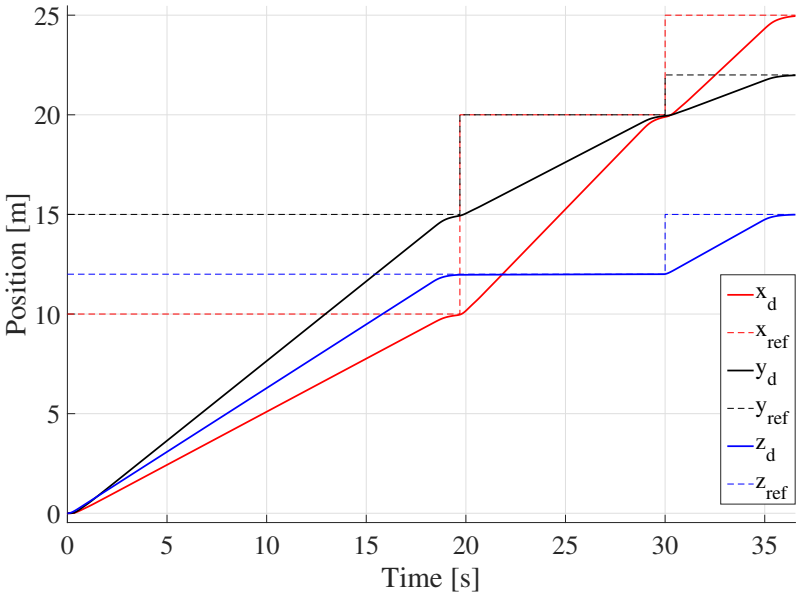

Fig. 3: A simulated example of the drone tracking a series of position references in space. Dashed lines represent the references. Note how, for most of the time, the position evolves as a straight line whose gradient corresponds to the drone's maximum speed.

3 Proposed approach

3.1 Path-Following Strategy

As anticipated in the introduction, our approach is based on a TSP where the cost function to be minimized is the total distance required to visit all points in P. To move the aircraft between two generic inspection points $\boldsymbol{p}_i, \boldsymbol{p}_j \in P$, we feed the path following controller a sequence of waypoints of the form:

$$S_{ref_{i,j}} = \left\{\boldsymbol{p}_i, \boldsymbol{w}_1 \ldots, \boldsymbol{w}_{N_{i,j}}, \boldsymbol{p}_j\right\}$$

where the intermediate waypoints $\boldsymbol{w}_\ell \in \mathbb{R}^4, \ell = 1, \ldots, N_{i,j}$ are chosen as described in Section 3.2, in order to avoid obstacles. The path following strategy then sets $\boldsymbol{w}_1$ as the initial reference position and yaw angle for the drone and switches from $\boldsymbol{w}_i$ to $\boldsymbol{w}_{i+1}$ when the drone is sufficiently close to $\boldsymbol{w}_i$, within a user-defined distance $\bar{r}$. Thus, the drone does not stop at each waypoint, rather the latter are used to shape its trajectory in space. On the other hand, after the last switching event from $\boldsymbol{w}_{N_{i,j}}$ to $\boldsymbol{p}_j$, the latter is kept as reference until the drone reaches it and its linear and angular speed values are negligible (e.g., 10^{-6} m/s and rad/s, respectively). A sequence comprising more than two points of interest, such as the final result S_{ref} of our algorithm, is followed by applying the same switching strategy, where the drone stops at each point of interest and travels through the intermediate waypoints.

3.2 Local Path Generation Strategy

The proposed solution method starts by partitioning the set of obstacles $\mathcal{O}$ and the set of inspection points P into n_z subsets or "zones". Each zone is therefore characterized by some obstacles and by the inspection points of P that are closest to them. More precisely, we calculate the distance between each inspection point i and each obstacle j as the geometrical distance between the point and the polytope describing the obstacle selecting a finite number of "zones", each one containing a subset of obstacles and of inspection points. In particular, let us indicate with $\mathcal{O}_k,\ k = 1, \ldots, n_z$, a finite number of subsets of $\mathcal{O}$ such that:

$$\bigcup_{k=1}^{n_z} \mathcal{O}_k = \mathcal{O}; \quad \mathcal{O}_{k_1} \cap \mathcal{O}_{k_2} = \emptyset \ \forall \ k_1 \neq k_2. \tag{5}$$

Thus, the sets $\mathcal{O}_k$ partition $\mathcal{O}$ into n_z non-overlapping regions. For each set $\mathcal{O}_k$, let us denote with P_k the set of inspection points that are closer to an obstacle $O_i \in \mathcal{O}_k$ than to any other obstacle outside $\mathcal{O}_k$:

$$\begin{aligned} P_k = \{\mathbf{p} \in P \ : \ & \delta(\boldsymbol{p}, O_i) < \delta(\boldsymbol{p}, O_j), \forall O_j \in \mathcal{O} \setminus \mathcal{O}_k \\ & \wedge O_i \in \mathcal{O}_k\}, k = 1, \ldots, n_z \end{aligned} \tag{6}$$

where

$$d_{i,j} = dist(\mathbf{p}_i, o_j), \quad i = 1, \ldots, n_p; \ j = 1, \ldots, n_o. \tag{7}$$

Then, we name a *zone* z_k the pair $(\mathcal{O}_k, P_k)$. see e.g. Fig. **??** (upper plot). Let us denote $n_{p,k}$ the number of point of interest in zone z_k. For each of the n_z zones

obtained this way, a cloud of $n_{i,k}$ intermediate waypoints named I_k is obtained by randomly distributing them in a limited region of space surrounding the obstacles of the zone. In particular, they must lie within a bounding box of the obstacle, but at a distance of at least d_o from the nearest obstacle. A random distribution was chosen to represent a general situation, but it is also possible to discretise the space by distributing points in ordered patterns. In any case it is crucial to make sure that none of the points lies inside an obstacle and it is useful to add a safety margin by ensuring all points lie at least at an arbitrary minimum distance from the nearest obstacle. An undirected weighted graph $\mathcal{G}_k$ is then drawn, where nodes V_k include both inspection and intermediate points of zone z_k

$$V_k = P_k \cup I_k, \tag{8}$$

for a total of $n_{p,k} + n_{i,k}$ nodes. Edges E_k are drawn between a pair of nodes $(\boldsymbol{v}_i, \boldsymbol{v}_j) \in V_k$ (be they inspection or intermediate points) if their distance is below a certain threshold $\bar{d}$ and no obstacle lies between them. The edges are collected in the set E_k:

$$\begin{aligned} E_k(V_k, \mathcal{O}_k, \bar{d}) = \big\{ (\boldsymbol{v}_i, \boldsymbol{v}_j) \ : \ \|\boldsymbol{v}_i - \boldsymbol{v}_j\|_2 \leq \bar{d} \\ \wedge \ \phi(\boldsymbol{v}_i, \boldsymbol{v}_j, O), \ \forall O \in \mathcal{O} \big\} \end{aligned} \tag{9}$$

The $free(p_i, p_j)$ function is only true if no convex combination of the two points p_i, p_j lies inside an obstacle:

$$\begin{aligned} &free(p_i, p_j) = \\ &\begin{cases} True & A_j \left[\alpha p_i + (1-\alpha) p_j\right] \leq b_j \quad \forall \alpha \in [0,1], \ \forall O_j \in \mathcal{O}_k \\ False & otherwise. \end{cases} \end{aligned} \tag{10}$$

The weight of an edge is equal to the distance between its extremes, i.e. $\|\boldsymbol{v}_i - \boldsymbol{v}_j\|_2$ for the edge connecting points $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$. We denote with D_k the ensemble of such edge weights.

$$D_k : (p_i, p_j) \rightarrow \|p_i - p_j\|_2, \tag{11}$$

The graph pertaining to zone z_k is finally denoted as:

$$\mathcal{G}_k = (V_k, E_k, D_k). \tag{12}$$

The edges represent all the feasible paths between points in space, thus solving the problem of collision avoidance, provided that the aforementioned safety margins are added in the generation of the intermediate points. Such margins are necessary because the trajectory of the drone will not coincide perfectly with the edges of the graph. The distribution of intermediate points can be manipulated to obtain a denser point cloud in certain zones, such as around corners or close to the obstacle surface, to provide more options to the path planner. The distance threshold $\bar{d}$ is a tunable parameter to modify the number of edges in the graph, forcing only pairs of sufficiently close nodes to be connected. A lower value of $\bar{d}$ significantly reduces the overall computational time needed to calculate shortest path without impacting much the quality of the

output. The reason is that, in practice, a path that visits all points of interest exactly once is much more likely to contain shorter edges than longer ones, corresponding to a situation where from each point a closer one is visited next, instead of a farther one, if the choice is available. Furthermore, the presence of an obstacle between two points is more likely if they farther away, so that longer edges tend to violate the conditions in (9). On the other hand, the parameter $\bar{d}$ should be chosen large enough to obtain a connected graph. A theoretical guarantee that the obtained graph will be connected in the absence of obstacles is obtained if

$$\bar{d} \geq d^* = \min_{\boldsymbol{p_i} \in V_k} \left\{ \max_{\boldsymbol{p_j} \in V_k} \left(\|\boldsymbol{p_i} - \boldsymbol{p_j}\|_2 \right) \right\}, \tag{13}$$

i.e. if the distance parameter is larger than the minimum worst-case distance between each pair of points in V_k. This is easily proven: the single point whose maximum distance from other points is minimized will be connected to all nodes, including the farthest one, which lies d^* away. If all node graphs are connected to this single node, in turn, the graph is connected. Note that the property of being connected might also be obtained with $\bar{d} < d^*$, depending on how the points are arranged in space. The condition expressed in (13) guarantees the property independently of point distribution. In presence of obstacles, though, the condition is no longer sufficient, as edges between some nodes might become unfeasible. In this case the probability that the resulting graph is connected can be increased by raising the density of intermediate points.

Once the graph is obtained, for each pair of inspection points $(\boldsymbol{p}_i, \boldsymbol{p}_j) \in P_k$s, the shortest path that connects them across the graph is computed, thus obtaining a sequence $S_{ref_{i,j}}$ that is fed to the simulated model of the drone, as described in Section 3.1, to obtain the flight time $F_t(\boldsymbol{p}_i, \boldsymbol{p}_j)$. The simulation starts from one end of such path, where the drone is fed the coordinates of the first set-point as reference. Once the three shortest paths are simulated the one yielding the lowest time is deemed to be the fastest and both the path and the time taken are stored in memory.

We can now generate a new, smaller, undirected, weighted graph $\mathcal{T}_k$ whose nodes are only the inspections points P_k and whose edges represent the paths found at the previous step. In this graph, the weight of each edge is the simulated flight time of the fastest path between its extremes, see Fig. 7. It is also possible to introduce a distance threshold, like the $\bar{d}$ parameter before, to neglect connections between inspection points that are far from one another, but due to their number being much lower than the number of nodes in the previous graph, the time saved is usually negligible. The last step we carry out on the zone z_k, $k = 1, \ldots, n_z$ is to decide where to start the inspection from and in what order the inspection points should be visited. Note that for a single zone, the path does not need to be cyclic, rather we are looking for the shortest Hamiltonian path, i.e. the shortest path that visits all the nodes of this reduced graph exactly once. This is achieved by solving an instance of the TSP where the graph is augmented with one dummy node, that is connected to all other nodes through an edge of weight zero. The dummy node is chosen as the starting point for the TSP, so that the solver returns a cycle from which we can obtain the Hamiltonian path by removing the dummy node. This popular technique

also solves the problem of choosing what node the inspection should start from to minimize the length of the path. The problem is thus formulated as follows:

$$\min_{e_{i,j}} \sum_{i=1}^{n_{p,k}} \sum_{j=1}^{n_{p,k}} e_{i,j} d_{i,j} \qquad i,j = 1,\dots,n_{p,k} \tag{14a}$$

subject to

$$e_{i,j} \in \{0,1\} \qquad i,j = 1,\dots,n_{p,k} \tag{14b}$$

$$\sum_{i=1}^{n_{p,k}} e_{i,j} = 1 \qquad j = 1,\dots,n_{p,k} \tag{14c}$$

$$\sum_{j=1}^{n_{p,k}} e_{i,j} = 1 \qquad i = 1,\dots,n_{p,k} \tag{14d}$$

$$\sum_{i \in Q} \sum_{j \in V_k \setminus Q} e_{i,j} \geq 1 \qquad \forall Q \subset V_k, |Q| \geq 1, \tag{14e}$$

where the decision variable $e_{i,j}$ is equal to 1 if the edge between i and j appears in the solution and $d_{i,j} = d_{j,i} = F_t(\boldsymbol{p}_i, \boldsymbol{p}_j)$ is the weight of the edge connecting the inspection points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$. (14b) ensures the decision variables are boolean, (14c), (14d) force each node to have exactly one incoming and one outgoing edge in the solution, while (14e) imposes that the cyclic path is unique for the whole graph. In other words, for every possible subset of nodes Q, there has to be at least one edge that connects that subset to the rest of the graph, $V_k \setminus Q$.

All TSP instances in this paper are solved with LKH (Lin-Kernighan-Helsgaun) [10], an effective implementation of the Lin-Kernighan heuristic commonly adopted in literature [5, 6]. The solution for the k-th zone is saved as a sequence $S_{ref,k}$ passing through all the inspection points pertaining to that zone, and the process is repeated for all $k = 1,\dots,n_z$.

3.3 Global Path Generation Strategy

After the optimal path within each zone has been generated, we now consider the whole set of obstacles $\mathcal{O}$ to derive the desired complete path for the drone, with a conceptually similar approach at a higher level. To this end, instead of considering all the inspection points P, only those which resulted to be start or end point for the single zones are kept into account, i.e. the first and last points visited in the solutions of each of the single zones. This is because now only the order of travel between zones, not within them, must be chosen. The starting position of the drone is added to the set and used as starting node. The approach is very similar to the one for single zones: first a set I_g of waypoints, all lying outside obstacles, is randomly distributed in the space around the latter, then a graph is generated by connecting nodes that are sufficiently close, based on a tunable distance $\bar{d}_g$ and provided that no obstacle lies between them. The new graph $\mathcal{G}_g$ is thus indicated as:

$$\mathcal{G}_g = (V_g, E_g, D_g), \tag{15}$$

where the node set V_g contains the randomly distributed intermediate points I_g and, among the inspection points $\boldsymbol{p} \in P$, only those that correspond to entry and exit points for each of the n_z zones, plus the starting/final position of the drone, $\boldsymbol{p}^*$. Entry and exit points of a zone are respectively the first and last inspection points appearing in the solution for the single zone $S_{ref,k}$, indicated as $S^{(1)}_{ref,k}$ and $S^{(\text{end})}_{ref,k}$, respectively. We group these points in the set

$$P_g = \bigcup_{k=1}^{n_z} \left(S^{(1)}_{ref,k} \cup S^{(end)}_{ref,k} \right) \cup \boldsymbol{p}^*. \tag{16}$$

Edges between such nodes are drawn with the same criterion,

$$\begin{aligned} &E_g(V_g, \mathcal{O}, \bar{d}_g) = \{(p_i, p_j) \mid dist(p_i, p_j) \ \leq \bar{d}_g \ \wedge \ free(p_i, p_j)\} \\ &i, j = 1, \ldots, 2n_z + n_i + 1. \end{aligned} \tag{17}$$

The same remarks on the role of the distance threshold apply. The weight associated to each edge the function D_g is once again the geometrical distance between the nodes it connects. Then, a new graph $\mathcal{T}_g$ is defined, whose nodes are the points in P_g and edges are weighted with the flight distance values. A last instance of the TSP is solved on this "global" graph, with $\boldsymbol{p}^*$ as the starting node. Note that this time the path we are looking for is cyclic, hence we need not add the dummy node. The problem is formulated so that the edges between each entry point and the corresponding exit point of a zone appear in the optimal solution to the TSP, thus ensuring that compatibility with the previously found paths inside each zone. To obtain this feature, with reference to formulation (14), the following constraints are added:

$$e_{i,j} = 1, \, \forall (i,j) : \boldsymbol{p}_i = S^{(1)}_{ref,k} \wedge \boldsymbol{p}_j \in S^{(\text{end})}_{ref,k}, \, k = 1, \ldots, n_z \tag{18}$$

effectively reducing the number of decision variables. The solution S_g of the global TSP is combined with those of each zone, by inserting the sequences $S_{ref,k}$ in the corresponding edges of S_g, in order to eventually obtain the desired global series of waypoints for the drone, S_{ref}.

The described procedure is summarized in the following algorithm.

Algorithm 1 - Hierarchical path planner

1. Initialize the problem by subdividing the obstacles and inspection points sets, $\mathcal{O}$ and P, into n_z zones;
2. Compute the local optimal paths. For each zone z_k, $k = 1, \ldots, n_z$:
   (a) consider its obstacles and its inspection points, $\mathcal{O}_k$ and P_k. Distribute $n_{k,i}$ randomly chosen points around the obstacles, then build a first graph $\mathcal{G}_k$ encompassing points P_k and I_k;
   (b) For each pair of inspection points in $\mathcal{G}_k$, find the shortest path connecting them and save the total distance;

(c) Based on the calculated distances, build another graph $\mathcal{T}_k$, whose nodes are the inspection points P_k and edges correspond to the actual paths between them;

(d) Find the ideal order of navigation $S_{ref,k}$ between the nodes of $\mathcal{T}_k$ as the solution of a TSP augmented with one dummy node.

3. Compute the global optimal path:

   (a) Consider all obstacles $\mathcal{O}$ and the set of points P_g, comprising the first and last point for each of the n_z local solutions plus the starting position of the drone;

   (b) Build a graph $\mathcal{G}_g$ based on those nodes plus a set I_g of randomly distributed intermediate points lying outside the obstacles;

   (c) Generate the shortest paths between any pair of nodes of interest;

   (d) Build another graph $\mathcal{T}_g$ connecting the points in P_g with edges representing the simulated paths among them;

   (e) Find the sequence S_g through all zones and back to the starting position by solving another TSP, forcing to 1 the edges that connect each entry point of a zone to its exit point.

4. Build the global series of waypoints S_{ref} by combining the information in S_g and $S_{ref,k}$.

4 Simulation results

We carried out simulations in an environment that represents six identical pylons of a bridge (Fig. 12). All computations have been performed using Matlab and Simulink, on the same machine with an Intel® Core™ i7 2.20 GHz processor running Ubuntu 18. For the sake of simulating different conditions, the positions of inspection points were chosen randomly in confined regions of space near obstacle surfaces. The approach is evaluated in terms of both the time taken to elaborate a series of waypoints and the time taken by the simulated drone to travel between them, referred to as cost. The algorithm was tested five times with the same obstacles and the same number of randomly distributed inspection points. All runs consistently produced feasible paths. In particular, choosing $|P_k| = 30$ inspection points and $n_i = 1000$ random intermediate points per pylon, given the initial position $\boldsymbol{p}^* = [10,\ -1,\ 0]^T$, $\bar{d} = 3$ and $\bar{d}_g = 12$, it took on average $t = 35.4\ s$ to elaborate a solution. Attempting to find a new solution by fixing the inspection points and changing the randomly chosen intermediate ones yielded very similar solutions in terms of simulated cost: repeating the procedure in an instance with a final simulated cost of $c = 936\ s$ (equivalent to 820 m traveled) yielded results contained within $\pm 0.2\%$ of the first one, which leads us to conclude that the placement of random points does not significantly affect the final solution.

If we were to solve the same problem without the hierarchical structure, i.e. by grouping all random points and inspection points in one set instead of

partitioning, building the graph as described in Section 3 and solving a single instance of the TSP, the time taken would significantly increase: it takes 4.9 hours to reach the globally optimal solution, that shows a simulated travel time of $c = 906s$ (equivalent to 794 m traveled), just 3.2% faster than the one obtained by our hierarchical technique. Figs. 9 and 10 show the dependence of the obtained results on problem complexity. The same problem was solved with identical parameters multiple times in a similar environment, starting from only considering two pylons and adding one at a time. The solutions were obtained both with the method presented in Section 3 and by applying the same approach in a non-hierarchical fashion. The time taken was subdivided into total obstacle check and graph creation time (t_{graph}) and total TSP formulation and solution time (t_{TSP}). The fact that by far most of the computational time is spent checking for collisions along edges and building the enlarged graph instead of solving the TSP instances (see Fig. 9 and 10) was unexpected to us. Nevertheless, our hierarchical approach also tackles that problem by partitioning space into zones, thus originating many smaller graphs instead of a single one and decreasing the computational burden, in this case by three orders of magnitude. Furthermore, the TSP solution time depends in general on the number of inspection points in each zone, and our approach still is computationally much faster. Simulations also indicate that, as discussed in Section 3.2, reducing the number of edges in the graph with intermediate and inspection points by only connecting nodes whose distance is less than the threshold $\bar{d}$ positively impacts the solution time, by reducing the complexity of the graph. While using the distance threshold in the previously mentioned instance of the problem yields a solution with cost $c = 936\ s$ in $t = 35.4\ s$, removing the threshold (thus considering all possible feasible edges) increases the time to $t = 935\ s$, while still leading to the same solution. This simplification proves reasonable because edges between nodes represent the time it takes to go form one to the other, thus we can expect that the optimal solution tends to contain shorter edges, rather than the same number of longer ones.

5 Conclusions and future work

The presented algorithm is found to return scalable and nearly optimal solutions to the considered path planning problem. This is achieved by effectively decoupling the obstacle avoidance problem from the planning itself, through a discrete approximation of space that is also useful to determine the visiting order of the points of interest, by partitioning the space so that more, smaller instances of the TSP can be solved instead of a single large one, and by introducing a tunable distance threshold to reduce the number of graph edges to be considered when solving the TSP. While theoretically the optimality of the solution is impacted, from a practical standpoint it is clear that the optimal solution is really unlikely to contain edges with high weights, as they represent a geometric distance. In other words, optimal solutions are expected to connect a point to its close neighbors rather than to a distant one. Neglecting one edge of the graph also means that it is not necessary to check if it traverses an obstacle, which is rather costly. The required computational time is sufficiently low for an off-line application, and it is achieved through two main steps:

* Partitioning the space so that more, smaller instances of the TSP can be solved instead of a single large one. This comes at the cost of a certain sub-optimality, since the local solution for each zone, together with entry and exit point, is already fixed when the global solution is calculated. While the simulations show that the impact on the global solution is negligible, this issue will be addressed in the future development phases of the project.

* Introducing a tunable distance threshold to reduce the number of graph edges to be considered in solving the TSP. While theoretically the optimality of the solution might be impacted, from a practical standpoint it is clear that the optimal solution is really unlikely to contain edges with high weights, as they represent a value that is conceptually similar to a geometric distance. In other words, optimal solutions are expected to connect a point to its close neighbors rather than to a distant one. Neglecting one edge of the graph also means that it is not necessary to check if it traverses an obstacle, which is rather costly.

This approach is also quite flexible, as the cost assigned to edges of the graph of inspection points can be modified to contain various parameters such as the simulated energy consumption, not only the time spent.

5.1 Future work

As mentioned in Section 1, this research is part of a larger project aimed at building a prototype of a system of both tethered and untethered drones to be used in automated recurrent inspections of the built environment. This algorithm will likely be developed and expanded into a multi-agent version, where more drones are available to perform the inspection tasks and they have to be coordinated. It will also be complemented with a procedure for the automated generation of inspection points starting from information regarding the obstacles and as a function of the type of inspection to be performed. The issue of sub-optimality due to entry and exit points being fixed will be addressed, as there is an attempt currently under way to use the position of those points as decision variables in an optimization problem aimed at minimizing the length of the overall trajectory.

References

[1] L. Heng, A. Gotovos, A. Krause and M. Pollefeys, Efficient Visual Exploration and Coverage with a Micro Aerial Vehicle in Unknown Environments, *in 2015 International Conference on Robotics and Automation*, Seattle, Washington, 2015, pp. 1071-1078.

[2] R. Du and R. V. Cowlagi, Interactive Sensing and Path-Planning with Incremental 3D Path Repair for a Quadrotor UAV in Cluttered and Partially Known Environments, *in 56th Annual Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 933-938.

[3] M. Kothari, I. Postlethwaite and D. W. Gu, Multi-UAV Path Planning in Obstacle Rich Environments Using Rapidly-exploring Random Trees, *in joint 48th Conference on Decision and Control and 28th Chinese Control Conference*, Shanghai, Chine, 2009, pp. 3069-3074.

[4] B. Sakcak, L. Bascetta and G. Ferretti, Homotopy aware kinodynamic planning using RRT-based planners, *in 18th European Control Conference*, Napoli, Italy, 2019, pp. 1568-1573

[5] K. Alexis, C. Papachristos, R. Siegwart and A. Tzes, Uniform Coverage Structural Inspection Path-Planning for Micro Aerial Vehicles, *in 2015 International Symposium on Intelligent Control*, Sydney, Australia, 2015, pp. 59-64.

[6] A. Bircher, K. Alexis, M. Burri, P. Oettershagen, S. Omari, T. Mantel and R. Siegwart, Structural Inspection Path Planning via Iterative Viewpoint Resampling with Application to Aerial Robotics, *in 2015 International Conference on Robotics and Automation*, Seattle, Washington, 2015, pp. 6423-6430.

[7] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan and N. Joshi, Submodular Trajectory Optimization for Aerial 3D Scanning, *in 2017 International Conference on Computer Vision (ICCV)* 2017, pp. 5324-5333.

[8] F. Cakici, H. Ergezer, U. Irmak and M. K. Leblebicioglu, Coordinated guidance for multiple UAVs, *in 2016 Transactions of the Institute of Measurement and Control*, Vol 38(5), pp. 593-601.

[9] L. Fagiano, Systems of tethered multicopters: modeling and control design, *in IFAC-PapersOnLine*, Vol 50(1), pp. 4610-4615.

[10] K. Helsgaun, An effective implementation of the Lin–Kernighan traveling salesman heuristic, *European Journal of Operational Research*, Vol 126(1), pp. 106-130.

[11] J. Y. Yen, Finding the k shortest loopless paths in a network, *management Science*, 1971, Vol 17(11), pp. 712-716.

[12] S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning, 1998.

[13] M. Bolognini and L. Fagiano, LiDAR-Based Navigation of Tethered Drone Formations in an Unknown Environment, *International Federation of Automatic Control World Congress, in press, available at https://arxiv.org/abs/2003.12981*, Berlin, Germany.

[14] S. Formentin and M. Lovera, Flatness-based control of a quadrotor helicopter via feedforward linearization, *50th IEEE Conference on Decision and Control and European Control Conference*, 2011, pp. 6171-6176.

[15] M. Salaris, A. Riva and F. Amigoni, Multirobot Coverage of Modular Environments, *International Conference on Autonomous Agents and Multiagent Systems*, Auckland, 2020, pp. 1178–1186.

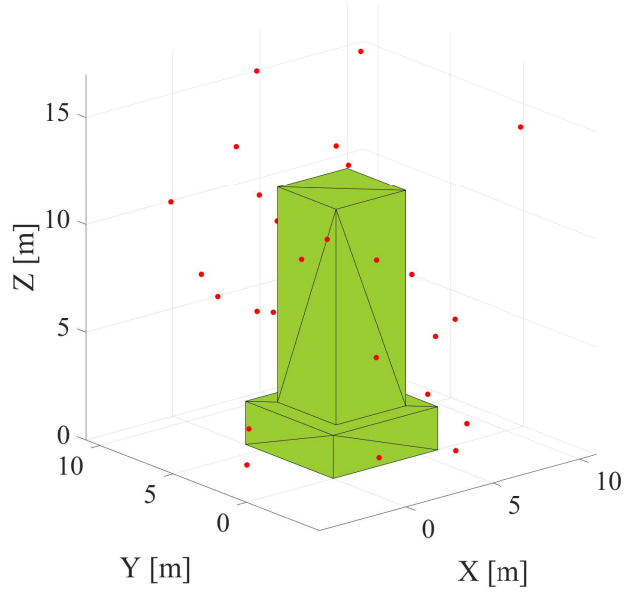

Fig. 4: A representation of an obstacle (green) with the related inspection points (red).

Algorithm 1: Pseudocode for the illustrated approach.

Result: Series of waypoints for the drone
$[n_z, \mathcal{O}_{1:n_z}, P_{1:n_z}] = partition(\mathcal{O}, P)$;
for $k = 1, \ldots, n_z$ **do**
  $I_k = randomDistribution(n_i, \mathcal{O}_k)$;
  $G_k = graph(P_k \cup I_k, E_k, D_k, \mathcal{O}_k, \bar{d})$;
  $S_k = shortestPaths(G_k, P_k)$;
  $W_k = simulate(S_k)$;
  $\mathcal{T}_k = graph(P_k, S_k, W_k)$;
  $zoneSol_k = HamiltonianPath(\mathcal{T}_k)$;
  $P_g = [P_g, zoneSol_k(1), zoneSol_k(end)]$;
end
$I_g = randomDistribution(n_i, \mathcal{O})$;
$G_g = graph(P_g \cup I_g, E_g, D_g, \mathcal{O}, \bar{d}_g)$;
$S_g = shortestPaths(G_g, P_g)$;
$W_g = simulate(S_g)$;
$\mathcal{T}_g = graph(P_g, S_g, W_g)$;
$globalSol = TSP(\mathcal{T}_g)$;
$solution = combine(globalSol, zoneSol_{1:n_z})$;

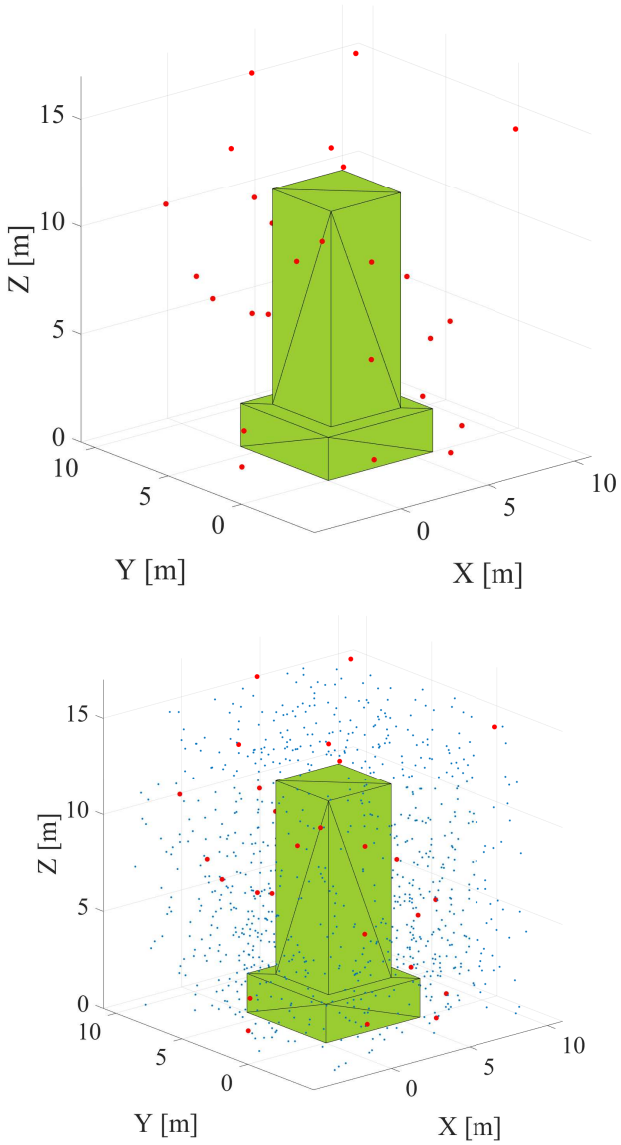

Fig. 5: Representation of an obstacle (green) with the related inspection points (red) and intermediate points (blue).

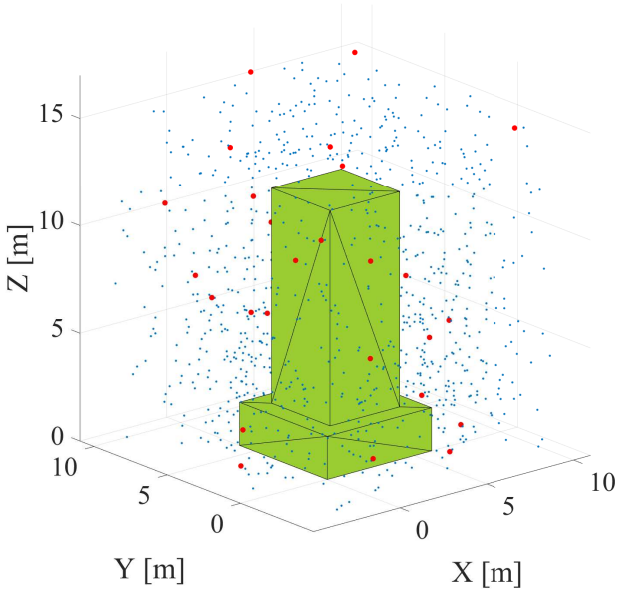

Fig. 6: The same representation of fig. 5, with added intermediate points (blue).

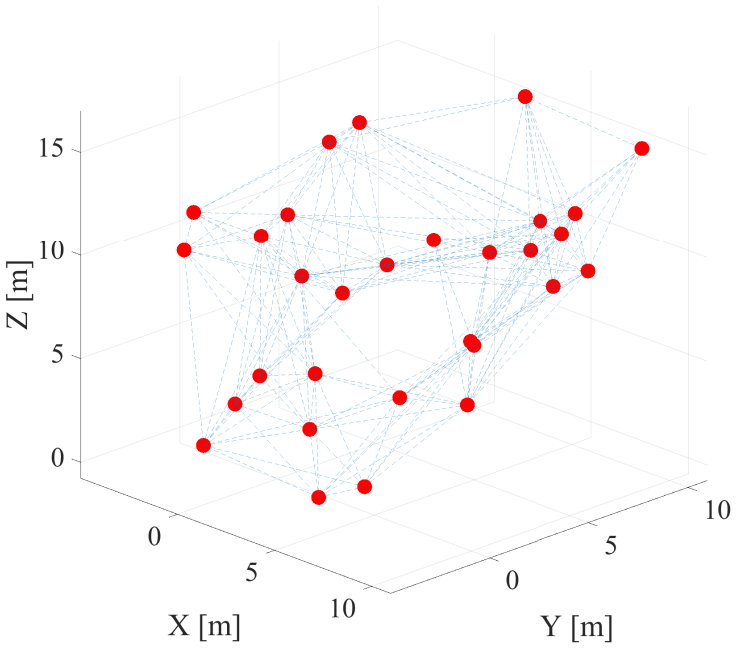

Fig. 7: The graph of inspection points obtained from the point cloud in Fig. 5 (right) after simulations.

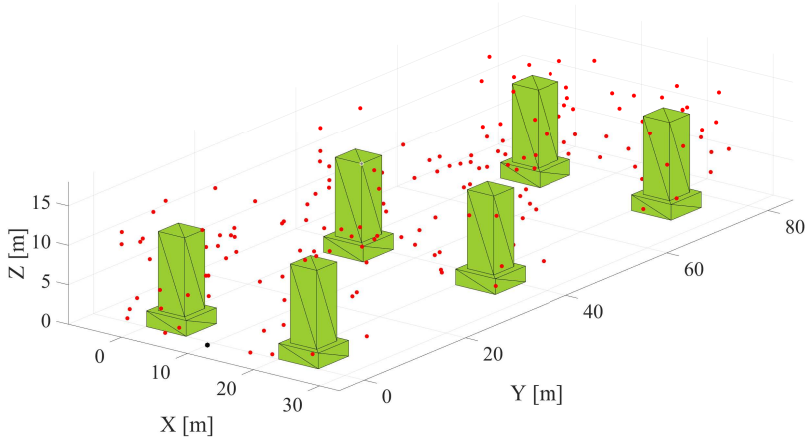

Fig. 8: The simulated environment. The dots represent inspection points.

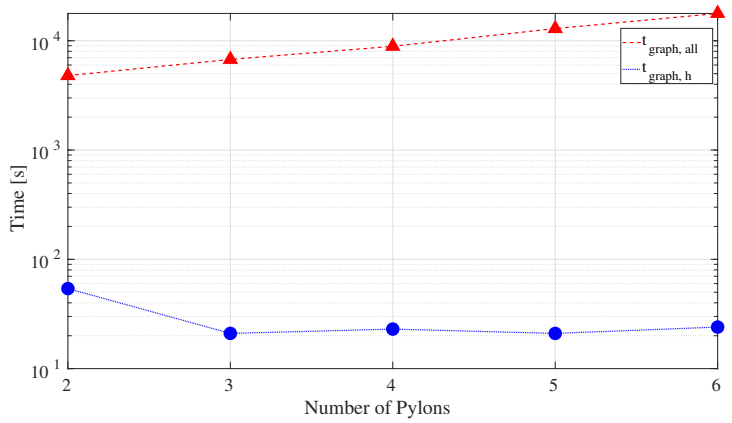

Fig. 9: In both approaches, the most time consuming task is the creation of the graph. With the proposed technique, the total graph creation time, $t_{graph,h}$, is significantly reduced. Note the logarithmic scale. The subscript h indicates the hierarchical approach. The time is highest with two pylons with the hierarchical approach because in that case the free space around the obstacles has a volume comparable to the obstacles themselves. This in turn means that it takes more time to randomly distribute points in free space.

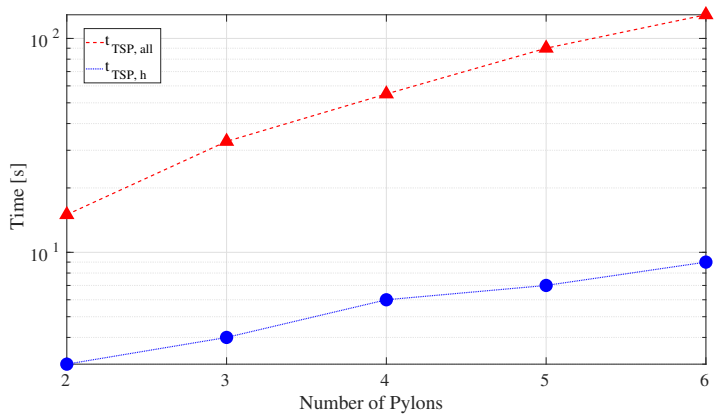

Fig. 10: In the non-hierarchical case, the time needed to actually solve the TSP problem increases exponentially with the problem complexity, though with the chosen parameters it is just a fraction of the total solution time. Our approach (blue) significantly reduces it with respect to a non-hierarchical one (red) and has a linear increase, which is the expected behavior when

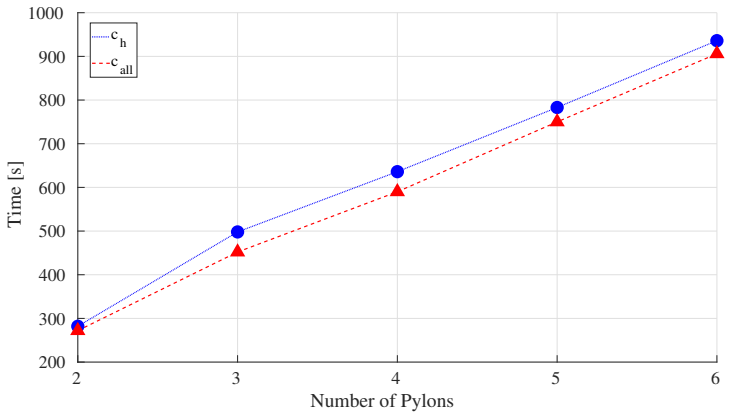

Fig. 11: The difference in simulated cost c is always rather small, a sign that the solution obtained by the proposed approach is nearly optimal.

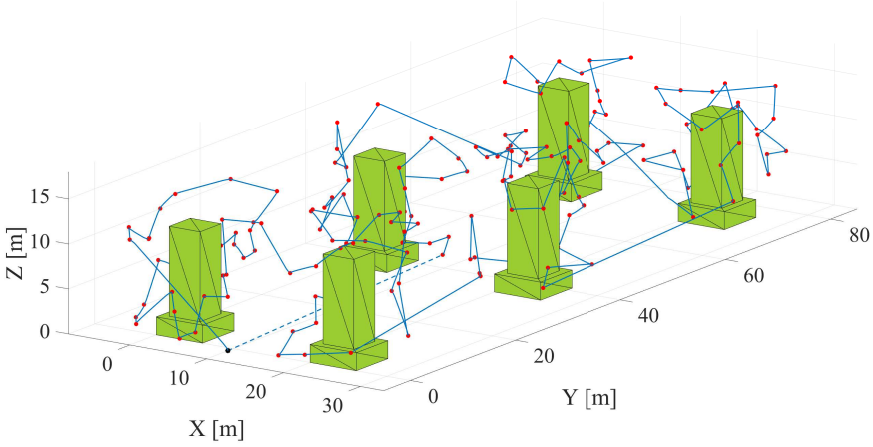

Fig. 12: The series drone trajectory (blue) for the solution found in the instance of the previous figure. The black dot at position $[10, -1, 0]$ represents the initial position. The dotted line is the last segment of the trajectory, to illustrate the direction.