2021 International Conference on Unmanned Aircraft Systems (ICUAS)
Athens, Greece. June 15-18, 2021

Inspection Path Planning for Aerial Vehicles via Sampling-based Sequential Optimization

Liping Shi[1], Golizheh Mehrooz[2] and Rune Hylsberg Jacobsen[1]

***Abstract*— Unmanned Aerial Vehicles (UAVs) commonly called drones are gaining interest for infrastructure inspection due to their ability to automize and monitor large areas more securely at a lower cost. Autonomous inspection and path planning are essential capabilities for the drone's autonomous flight. In this paper, we propose a novel inspection path planning method for achieving a complete and efficient inspection using drones. A point cloud generated from a 3D mapping service is used to represent complex inspection targets and provided as the input of the path planning method. The method is designed as a sampling-based sequential optimization to calculate and optimize an inspection path while considering the limitation of the sensors, inspection efficiency, and safety requirements of the drones. The proposed method is evaluated for both the use case of bridge inspection and power pylon inspection. A comparison between the proposed path search algorithm and TSP solver is made. Furthermore, the scalability of the method is assessed with different sizes of the inspection problem.**

I. Introduction

Drones have been widely adopted by infrastructure operators for some of the maintenance tasks, such as power line, highways, railways, and bridge inspections [1][2]. However, due to limitation of the current autonomous drone technology [3], a large number of standard inspection tasks are still labor-intensive, time-consuming, and sometimes even dangerous. To efficiently maintain safety for a large number of infrastructures, it motivates researchers to continuously improve and automate the data processing steps with drones. Today, many of the infrastructures have been digitized as geometric models with detailed structure information [4]. Some of the 3D reconstruction technologies [5][6] also provide quick mapping methods to acquire structure information using drones. Therefore, infrastructure operators are often interested in keeping their data up to date with the condition changes of the infrastructures (e.g. rust, abrasion, and degradation) and surrounding environment, or enriching their data by involving more advanced sensors to inspect from different perspectives.

We focus on the problem of path planning for inspecting infrastructures using autonomous drones. Our proposed method assumes the structure of the inspection target is provided as a dense 3D point cloud, which is a common format of the 3D reconstruction result. It requires to solve a coverage planning problem [7] to fully visit the inspection target and subject to different constraints from the vehicle and inspection sensors. Similar to [8], the proposed solution aims to provide a complete, safe, and efficient path guidance for drones. Our contributions are as follows:

- We propose a fast, and efficient inspection path planning processing chain via segmented voxel sampling, traversal path search, and sequential convex optimization. Different from existing practices, a combination of segmentation, voxelization and sampling is used for representing the structure of any general inspection target. A new traversal path search algorithm and a path optimization model are optimized towards low computation complexity.
- We demonstrate the utility of the path planning method for bridge inspections and power pylon inspections. We analyze the computation characteristics and show that the proposed algorithm exhibits linear scalability regarding the computation time and the size of the inspection target.

The paper unfolds as follows. Section II summarizes the background of the inspection path planning and 3D data processing. Section III formulates the inspection path planning problem. Subsequently, Section IV introduces the processing chain for inspection path planning and the use cases for bridge inspection, and power pylon inspection. The computation characteristics are analyzed in Section V. Conclusions are summarized in Section VI.

II. Related Work

There is a good body of research regarding the drone inspection path planning. In [9], the authors analyzed the most successful UAV 3D path planning algorithms that developed in recent years. Path planning methods have been classified into several categories such as node-based algorithms and mathematical model-based algorithms. An important section of the mathematical model-based method is formulated as an optimization problem. To model the constraints and the objective function for path optimization, the authors in [10] discussed the quadrotor dynamics and smooth trajectory generation. To maintain safety, the concept of a safety-distance and collision avoidance was investigated and modeled in [11]. To provide good initial values for the optimization calculation, sampling-based path planning and path-guided trajectory optimization methods were proposed [12], [13]. They present a two-step path planner, of which the results of the sampling-based path planning are formulated either as initial values or constraints of the optimization model in the trajectory optimization method. Therefore, the optimization process is benefited from the sampling-based path planning to resist errors caused by the local optimal,

[1]Department of Electrical and Computer Engineering, Aarhus University, Denmark.
[2]Department of Mathematics & Computer Science, University of Southern Denmark

and to fulfill extra path requirements by involving new constraints.

Coverage Path Planning (CPP) is the problem of determining a path that covers all parts of an area or volume of interest. Such as inspection applications are often modeled as CPP problems [14][15]. A survey of the coverage path planning methods can be found in [16]. In the CPP problem, the targeting area of interest is represented and decomposed as polygons [17], grids [18], or graphs [19]. To achieve a complete coverage on the represented area, a path for the mobile robots or vehicles can be calculated based on different algorithms, including Boustrophedon and Trapezoidal algorithms for polygonal planar space [20], [21], Wavefront for grid space [22], and graph search algorithms [23]. However, most algorithms are designed for 2D environments. To address the complex 3D structures, a two-step optimization scheme was proposed to find close to the optimal path by iteratively solving Art Gallery Problem (AGP)[24] and Traveling Salesman Problem (TSP) [8],[25]. Although the result of AGP benefits to shorten total travel-distance, the smoothness of the path is hard to be optimized. The scalability of the solution is also limited by the NP-hard time complexity of TSP [26].

Our proposed inspection path planning method retains a two-step scheme. However, the order of the two-step is changed. Besides, the TSP step is replaced by a new traversal path searching with smoothness and scalability considered. The AGP step is replaced by sequential convex optimization. This is driven by the consideration that a simple and smooth path is more critical for the autonomous flight regarding the safety. Many inspection tasks expect a detailed examination of the target. In this regard, the advantage of solving AGP is limited. The first step is to maintain complete coverage of the inspection target. Then the second step is to generate a feasible flight path and to optimize the path in terms of safety and efficiency.

III. Problem Formulation

The problem of inspection path planning is to find an efficient path that guides autonomous systems to inspect the region of interest with onboard sensors. In this paper, the rotor-based drone, as the autonomous system is mainly investigated. The inspection target, e.g. a bridge represented by point cloud, is assumed to be provided. However, to fit in a practical situation, the noise and uncertainty in the point cloud dataset generated from the 3D mapping service need to be addressed. Besides, the limitations of the onboard inspection sensors, the dynamic constraints of the drone are considered. An inspection is defined as a sequence of actions taken by the rotor-based drone that uses the onboard sensor such as a camera, to sense an area of the inspection target. The inspection path is defined as sequentially connected waypoints, which indicates the position of the drone and the sensor to fulfill the inspection task. The efficiency of the path is measured as the path length and the number of the direction changes. Therefore the problem is formulated as finding an efficient path with the constraints from the system, which guarantees a complete visibility of the inspection target represented as a point cloud. The solution to the coverage problem was calculated before the operation to provide a guidance plan for the system and used with an onboard motion planner for unforeseen circumstances during the mission.

IV. Methodology

This section explores a data preprocessing pipeline for inspection path planning. As shown in Fig. 1, the data processing chain consists of three modules described in the subsections IV-A, IV-B, and IV-C. Our proposed path search method reads a point cloud set, e.g. PLY file, as input and exports a sequence of waypoints with related sensor orientations, e.g. camera angles. Step 1 works on data cleaning, normal vector estimation, and downsampling. Step 2 generates a graph based on downsampled data and then searches a feasible traversal path for the graph. Based on the traversal path and the normal vectors, Step 3 constructs the optimization problem to find and optimize the flight waypoints and sensor orientations.

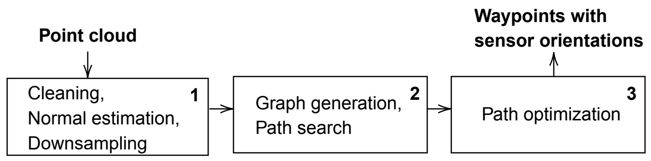
Point cloud
Waypoints with sensor orientations
Cleaning, Normal estimation, Downsampling
1
Graph generation, Path search
2
Path optimization
3

Fig. 1: Overview of the data processing steps.

A. Cleaning, Normal Estimation, and Downsampling

To represent the structure of complex 3D objects, e.g. buildings and bridges, reconstructing 3D scenes from 2D images is widely researched in computer vision as a structure From Motion (SFM) problem [27], [28]. Given the result of the 3D reconstruction, point cloud, the surface of the 3D object is simplified as a triangle mesh in [8]. However, the noise in the point may often jeopardize the quality of the triangle mesh surface representation, e.g. existing of many mesh holes, and overlapping triangles. A convex hull method is a way to simplify the surface. It has been adopted in many collision-free path planers [29], motivated by the features of the convexity, especially its low computation complexity.

In this work, we use voxel decomposition and sampling to simplify the surface of the inspection target to circumvent the uncertainty of triangle mesh generation. The point cloud of a bridge located in Villalvernia, Italy is used as illustrated in Fig. 2a. First, we select the area that requires inspection and crop out the relevant point cloud. To visualize and analyze our data, we are using the tool CloudCompare[1] for point cloud cropping. Fig. 2b shows the result of the cropping step. As shown, there is a lot of noise and outliers, which jeopardize the quality of the next step e.g. normal vector estimation. Therefore, some data cleaning steps are

[1]Available as an open-source tool at `https://www.danielgm.net/cc/`

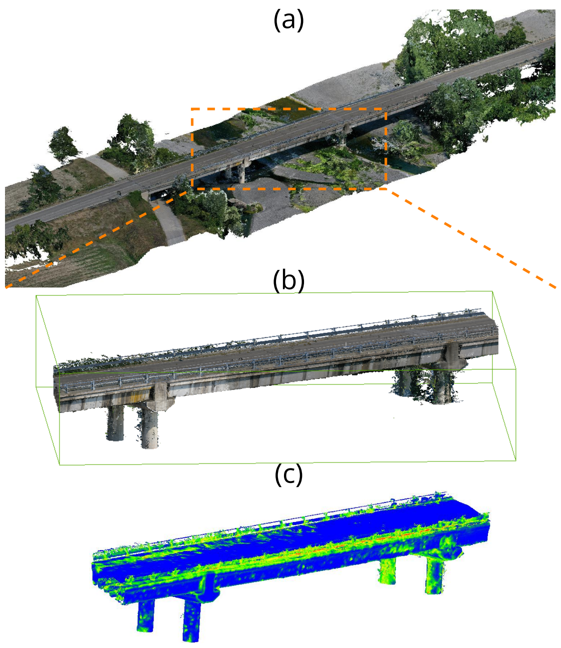

Fig. 2: (a) A point cloud (PLY file) is generated by the 3D mapping service. (b) The inspection area is selected and cropped. (c) The point cloud is cleaned and smoothed. The smoothness is represented by the difference in color.

completed. Obvious outliers are removed using the segment function in CloudCompare. Then an MLS (Moving Least Squares) smoothing is performed. Fig. 2c visualizes the output of the data cleaning step.

The Open3D library[2] is used for the point cloud processing including normal estimation, sampling, and visualization. The normal vector of each point is estimated and then oriented by using constructed tangent plane according to nearby points. To simplify the tangent plane construction, a voxel-based downsampling with a small voxel size (10 cm) is conducted beforehand. A point cloud with correctly estimated normal vectors is shown in Fig. 3a. Subsequently we reuse the voxel downsampling method to segment and discretize the surface of the inspection target. A larger voxel size parameter, 150 cm, is adopted. Each voxel represents an inspection area that can be scanned by a single trigger of the inspection sensor. It is designed to scan the voxel one by one. To acquire quality inspection, the voxel size is constrained by the maximal inspection distance and the field of view of the sensor. As the output of the first step, Fig. 3b presents the downsampled point cloud with normal vectors. In the next subsection, we introduce a graph generation and a graph-based path searching algorithm using the downsampled point cloud.

B. Graph Generation and Path Search

The drone flight becomes efficient when the guided path is simple, e.g. a path that avoids repeating trajectories or directional changes. It enhances the safety to leave more margin to react to unexpected situations in the field. However, a path search, e.g. finding the shortest traversal path from a graph representation of an inspection target, often leads to complex results. To control the complexity of the path search, a segmentation step is involved, as described in Algorithm 1. n_p denotes the normal vector of the point. n_f denotes the facet normal vector of the bounding box. The point cloud is segmented into 6 clusters based on the normal vectors of the 6 facets of the oriented bounding box, (Fig. 4a). Fig. 4b represents an example of a segmented point cluster of which the point normal vectors are close to the normal vector of the top facet of the bounding box.

[2]Python package available at `http://www.open3d.org/docs/release/`

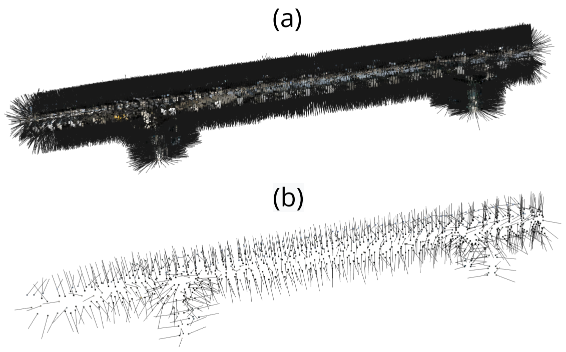

Fig. 3: (a) A normal vector of each point is estimated and oriented perpendicular to the consistent tangent plane. (b) The point cloud with its normal vectors is downsampled using voxelization.

Algorithm 1: A point cloud segmentation

```
  Input: A point cloud with normal vectors
  Output: A set of segmented point clouds
1 Function Segmentation:
2     Get bounding box around the object
3     Calculate facet normal vectors ($n_f$) of the
        bounding box
4     for Point in Point cloud do
5         $n_f^* = \underset{n_f}{argmax}\ dot(n_p,\ n_f)$
6         Segment into point cluster based on $n_f^*$
7     end
8 return
```

A directed graph is generated based on the downsampled point cloud. The NetworkX library [30] is used for graph storage. Each point corresponds to a node in the graph with related attributes, e.g. position and visiting status. The directed edges of the graph represent the feasible paths from the current point to the next point. Edges are added by connecting a predefined number of neighboring points. The cost for selecting the edge as part of the path is calculated and stored as an attribute in each edge object defined in NetworkX. The cost function is formulated to push the path search to gradually explore the inspection areas while

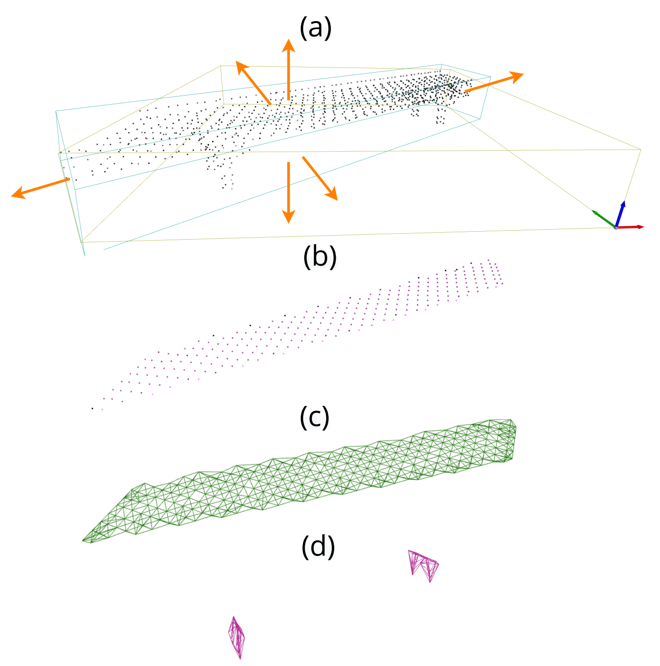

Fig. 4: (a) The point cloud is clustered based on the normal vectors shown with orange arrows, of the oriented bounding box in blue. The axis-aligned bounding box is shown in yellow with a local coordinate system visualized in the bottom right corner. (b) A cluster of the points that represents the top part of the bridge. (c) A graph is generated based on the point cluster in (b). (d) A graph generated from a different point cluster contains two components, i.e., two induced subgraphs in which any two vertices are connected to each other by paths.

evenly spreading from the initial position. The cost function considers both the distance from the last node and the distance from the start node, Eq. (1).

$$cost = dist(a, a_n) + dist(0, a_n) \quad (1)$$

where the a_n denotes a neighboring node of a. The function $dist()$ calculates the Euclidian distance between two nodes. The node 0 denotes a predefined starting node for path search. The cost function Fig. 4c presents a generated graph based on the point cloud shown in Fig. 4b.

A traversal path search algorithm is developed (Algorithm 2) to efficiently search a path that represents the sequence of inspection actions for a complete inspection. Algorithm 2 reads the generated graph and a predefined start node as input and outputs a traversal path. The traversal path is a list of node indexes. The start node is determined by finding the closest node to the initial take-off position of the drone or the position of the drone at the end of the last mission. Due to the segmentation step, the graph is not guaranteed to be a connected graph, as evident from Fig. 4d. To achieve traversal across multiple components (unconnected graphs), $Check\_unvisited\_node()$ enables jumps between components. The function $Get\_next\_node()$ searches the best next node based on the current node, the costs of edges to the neighboring nodes as defined in Eq. (1), and the visiting status of the neighboring nodes. Each node is designed to be visited only once. The edge with the lowest cost is selected. The standard $Breadth\_first\_search$ method provided by the NetworkX library [30] is involved to enable jumps between nodes that are not directly connected by an edge.

Algorithm 2: Traversal path search

```
Input: start_node, graph
Output: traversal_path
Function Get_next_node($current\_node, graph$):
    Find neighbors of $current\_node$ in $graph$
    if $unvisited neighbors \geq 1$ then
        Read edge cost of unvisited neighbors
        return neighbor with minimum edge cost
    else
        return closest unvisited node or None =
          Breadth_first_search($current\_node$, $graph$)
    end
Function Main:
    current_node = start_node
    traversal_path = [start_node]
    while $node$ != $None$ do
        node = Get_next_node(current_node, graph)
        if $node$ != $None$ then
            traversal_path.append(node)
        else
            node = Check_unvisited_nodes(graph)
            if $node$ != $None$ then
                Check visiting status of neighbors
                if $neighbor is visited$ then
                    traversal_path.insert(node) after the
                      neighbor index
                else
                    traversal_path.append(node)
                end
            else
                Break
            end
        end
        node's visiting status = visited
        current_node = node
    end
    return traversal_path
```

By searching paths for 6 clusters of the segmented point cloud based on the generated graphs, the inspection sequences for 6 clusters are calculated. As the output of the second step, Fig. 5 presents the segmented cloud point (6 clusters), normal vectors, and the traversal path, which guaranteed a complete visit of all points. Each point cluster is visualized with the axis-aligned bounding box. The axis represents the local coordinate system.

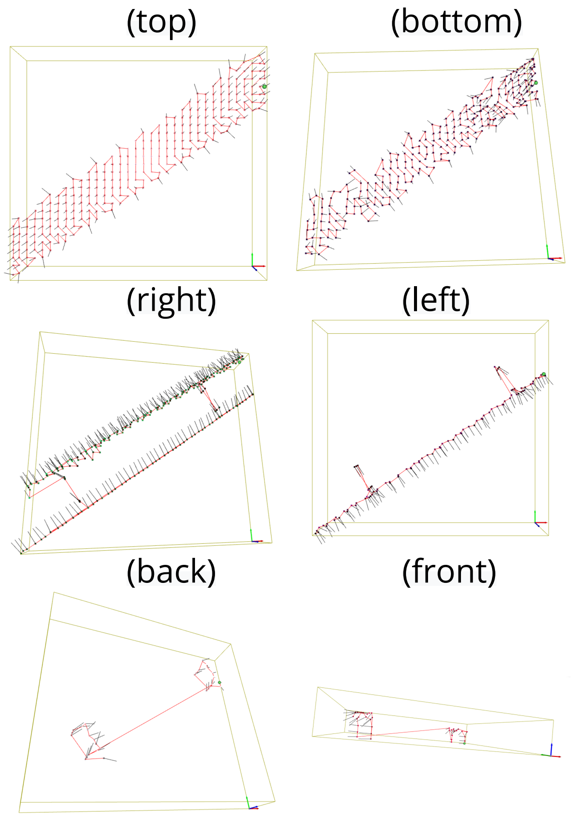

Fig. 5: An efficient path, shown in red, is searched to fully visit all points in each point cluster of the bridge, which is generated based on normal vectors representing 6 different directions. The green sphere, in the top left corner, denotes the start point of the path search.

C. Path Optimization

The path optimization step converts the inspection sequence (traversal path) to an efficient flight path while considering safety constraints, and sensor limitations. The flight path is optimized in terms of the flight distance and the path smoothness. The path optimization is constructed as a sequential convex optimization by combining point cloud data, normal vectors and the traversal path:

$$\begin{aligned} \min_{\Delta_j} \quad & \sum_{j=1}^{J} \|\Delta_j\|_2 \\ & + \beta \sum_{j=1}^{J-1} \|\Delta_{j+1}^{x,y} - \Delta_j^{x,y}\|_2 + \mu \sum_{j=1}^{J-1} \|\Delta_{j+1}^{z} - \Delta_j^{z}\|_2 \end{aligned} \tag{2}$$

$$\begin{aligned} s.t. \quad & g_1 = P_{start} + \Delta_1 \\ & g_{j+1} = g_j + \Delta_{j+1}, \ \forall j \in \{1, 2, ..., J-1\} \\ & (g_j - v_{j1})^T n_j \geq d_{min} \\ & -(g_j - v_{j1})^T n_j \geq -d_{max} \\ & (g_j - v_{ji})^T h_{ji} \geq 0, \ \forall i \in \{1, 2, 3, 4\} \end{aligned} \tag{3}$$

where $\Delta_j = [\Delta_j^x, \Delta_j^y, \Delta_j^z]$ denotes the position control variable in 3D space. $\Delta_j^{x,y}$ denotes the x and y-axis components of Δ_j. Δ_j^z denotes its z-axis component. β and μ are constant values that represent the scalar of the constructed cost function. P_{start} indicates the start position of the inspection task. $g_j = [g_j^x, g_j^y, g_j^z]$ denotes the position of j^{th} viewpoint, where the inspection sensor, e.g. the camera, is triggered to sense an area defined based on the j^{th} sampled point in the point cloud. The number of sampled points is J. An efficient traversal sequence of the sampled points is calculated at the path searching step. Eq. (2) describes the objective function for the path optimization, which minimizes the total path length and the change of the consecutive position control variable at the x, y, and z-axis. We select $\beta = 1$ and $\mu = 10$ to punish more for the difference generated from the z-axis, i.e., the altitude of the drone. Eq. (3) defines the constraints. It begins with equality constraints, defining the initial value and the update function for the calculation of viewpoint g. Then the inequality constraints represent the incidence angle limitations, the safety minimal distance, and the maximal inspection distance. Inequality constraints are visualized in Fig. 6, where the incidence angle limitations are constructed by the inner product between the vector $(g_j - v_{ji})$ and the normal vector h_{ji} of the hyperplanes. The hyperplane is defined by the incidence angle and the edge of the square constructed perpendicular to the normal vector n of the point p. The orientation of the sensor for each inspection area is defined by g and p, where the sensor is placed at the position of g and oriented towards the position of p, i.e., the sensor orientation is settled by the vector $(p - g)$.

A Python-embedded modeling language, CVXPY [31], is used for modeling the sequential convex optimization problem. The embedded conic solver (ECOS) [32], an interior-point solver, is mainly used for calculating the optimal result. Fig. 7 presents the result of the path optimization, where the incidence angle limitation is set to 30 degrees, the minimal safety distance and the maximal inspection distance (d_{min} and d_{max}) are set to 1 and 2.5 meters, respectively.

D. Deployment

It is recommended to deploy the proposed processing steps on a cloud computing platform, which provides sufficient computation power for path search, path optimization, as well as point cloud visualization, especially for large-scale inspection tasks. Cloud deployment also leads to convenient integration with the 3D mapping service, e.g. webODM, which provides input data for the processing chain. The processing chain is designed to provide path guidance for the inspection task. It is sufficient to generate the path before the task stated and then send the path data to autonomous drones as global guidance. The drone is expected to have an on-board local path planner for waypoints following and obstacle avoidance to respond to unexpected turns of events during the inspection.

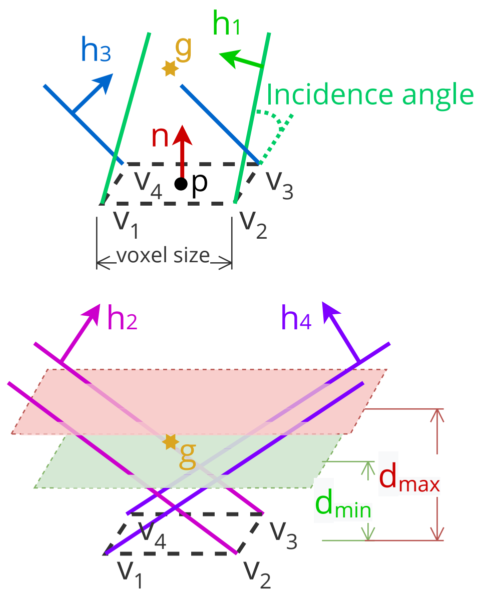

Fig. 6: A convex constraint space is constructed. p is a point in the downsampled point cloud. n is its estimated normalized normal vector. Marks $v_i, i \in \{1,2,3,4\}$ denote the vertices of the constructed square, of which the center is the position of p, the length is defined by the voxel size in the voxel downsampling step. Vectors $h_i, i \in \{1,2,3,4\}$ represent the normalized normal vectors of the hyperplanes, which are constructed based on edges of the square and the incidence angle constraint, e.g. h_1. Two hyperplanes in green and red parallel to the square are constructed at the side defined by the normal vector n and the distance defined by d_{min} and d_{max}. The star g denotes the viewpoint, of which the position is to be optimized subject to the constraints

V. Evaluation

To address the challenge of open structures, we introduce a new inspection case with an open structure by considering the metal frame of a power pylon. For this case, the computation demand of the proposed steps for different scales of the problem are presented. Furthermore, the proposed traversal path search algorithm is compared with the basic greedy TSP solver in terms of the computation time and the path results.

A. Power Pylon Case Study

We evaluate the proposed inspection path planning method by using the power pylon as a case study. Different from the bridge, which mainly consists of a solid structure, the power pylon is designed as a metal frame structure, of which the inner structures become visible. Therefore the point cloud generated by the 3D mapping service shows both the outer surface and the visible inner structures, which leads to the difficulty to estimate the normal vectors by constructing tangent planes using neighboring points. However, to avoid collisions during the inspection, it is critical to correctly model the convex constraint space (Fig. 8), which requires

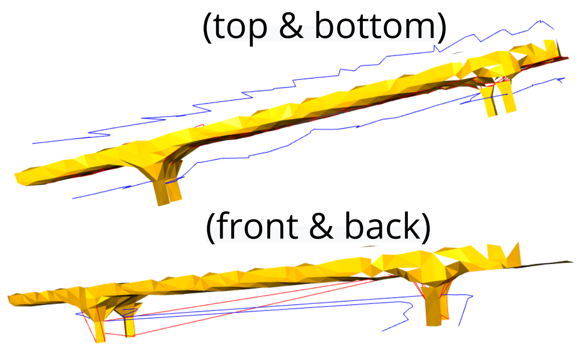

Fig. 7: The bridge is represented as a triangle mesh in yellow. Red lines indicate the result of the path search. Optimized inspection paths for the bottom-side, front-side, back-side, and front-side of the bridge are presented in blue.

accurate extraction of surface points and estimate their normal vector directions.

To ensure the safety of the flight path, the convex hull is extracted from the point cloud to represent the surface of the inspection target with the frame structure. The convex hull is the smallest convex polygon that contains all the points. Fig. 8b visualizes the convex hull of a power pylon Fig.8a. With subdivision, a triangle mesh model representing the convex hull is generated, Fig. 8c. Afterward, points are sampled from the mesh with a predefined surface-point density. At this stage, the data is ready to connect to the inspection path planning method introduced in Section IV.

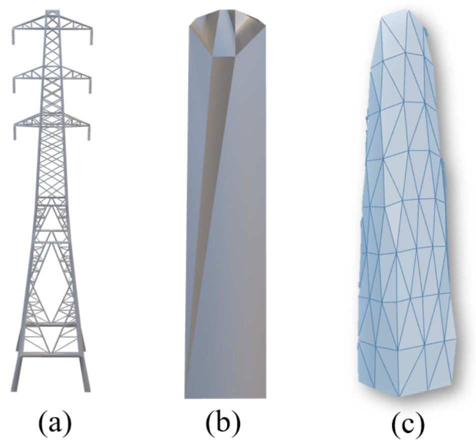

Fig. 8: (a) The power pylon point cloud. (b) A convex hull representation of the power pylon. (c) A triangle mesh model of the convex hull generated with subdivision.

Fig. 9 shows the result of power pylon inspection path planning using the proposed inspection path planning method. The size of the power pylon is approximately 4.5x4.2x20.2 meters. The voxel size for the voxel downsampling is set to 1 meter. The minimal safety distance is defined as 1 meter, while the maximal inspection distance is defined as 2.5 meters. The incidence angle limitation of the sensor is

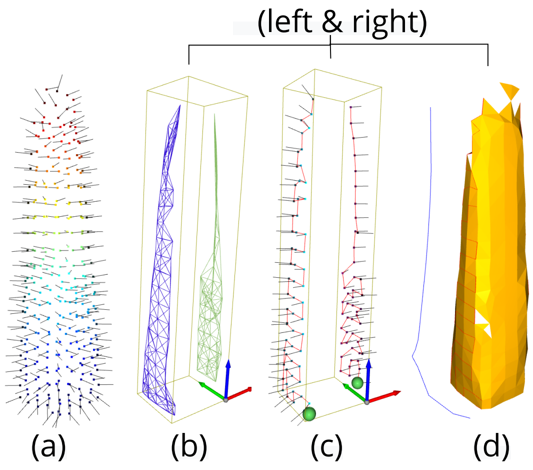

Fig. 9: Inspection path plan for a power pylon. (a) Sampling and normal vector estimation. Each sampled point locates in the center of a 1x1 meter inspection area. (b) Decomposition and graph generation. Components representing the left and right sides of the power pylon are visualized. (c) Path searching. Green spheres represent the starting inspection area from the left and right sides of the power pylon. (d) Optimized inspection paths are visualized in blue.

set to 30 degrees. We set $\beta = 1$ and $\mu = 10$ for the Eq. (2), the objective function for the path optimization.

B. Computation Analysis

A computation analysis is presented to provide an overview of the time consumption of the proposed method. The evaluation of the computation uses Intel i7-8650 CPU as the platform. Fig. 10 presents the time consumption of the path searching and the path optimization for processing different sections of the bridge. As introduced in Fig. 5, the bridge point cloud was segmented into 6 point clusters. The point cluster with more points needs a longer computation time. In the path optimization step, the computation time also depends on the speed of the solver. The total time consumption of the proposed method is presented in Fig. 11. The time consumption shows a linear increase with the sampling points. The total number of sampled points depends on the voxel size, a parameter for the voxel downsampling. As shown in figure 11, the majority of the time consumption is used for processing path optimization. Normal estimation, graph generation, and decomposition are the subsequent steps that are time-consuming.

To evaluate the scalability of the method, we measured the computation time of the path optimization for 5 different sizes of the problem, 20 measurements for each problem. Modeled as a sequential convex optimization, Eq. (2) and (3), the proposed optimization algorithm has O(n) time complexity. It is observed in Fig. 12, a linear relation between the computation time of the path optimization and the number of sampled points (J in Eq. (3)), reflecting the size of the problem. The number of sampled points is influenced by the voxel size parameter required by voxel downsampling step.

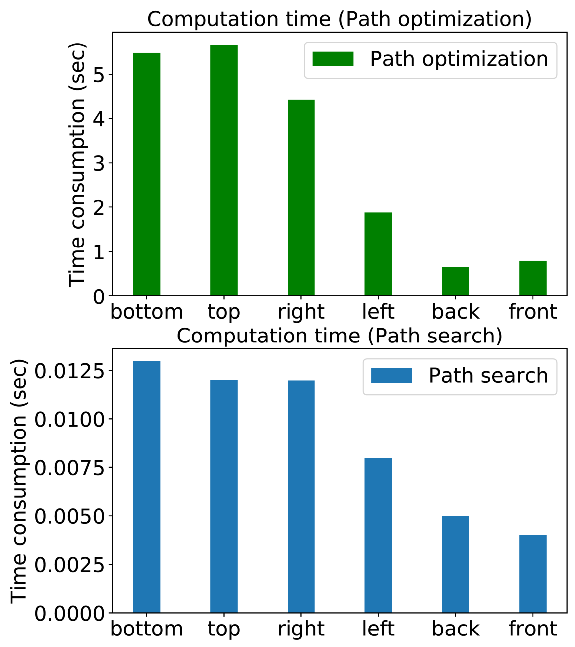

Fig. 10: Computation time of the path optimization and path search for different decomposed point clusters.

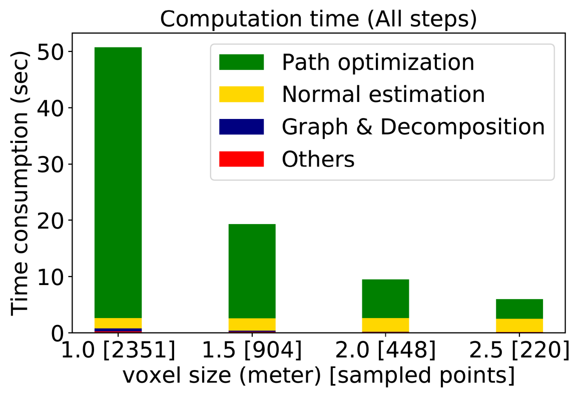

Fig. 11: The total computation time of the proposed method in terms of the voxel size. The number of the sampled points using voxel downsampling is given. The time consumption of different steps is provided individually to show the distribution. The red section (Others) includes steps of voxel downsampling, bounding box calculation, and path searching.

C. Comparison of TSP with our Path Search Method

This section evaluates the proposed path searching algorithm. The scalability and the result of the path searching are discussed. As the path is required to be a traversal, Fig. 13 shows a comparison of time consumption between the proposed method and the basic greedy Traveling Salesman Problem (TSP) solver [33], which provides a pure Python code for searching sub-optimal solutions to the TSP. As shown in this figure, the proposed method costs less time than the TSP solver. In the case that the problem size increases approx. 3 times, from 262 points to 770 points, the

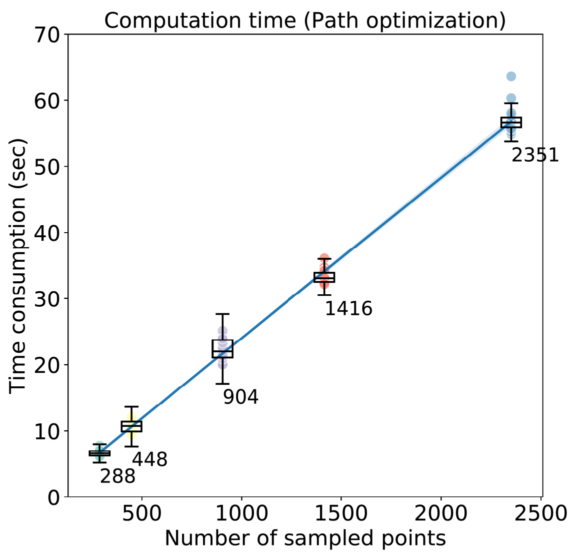

Fig. 12: Computation time of the proposed path optimization algorithm. The labels below the scatter plot points indicate the number of sampled points. A linear regression yields a R^2 value of 0.9945.

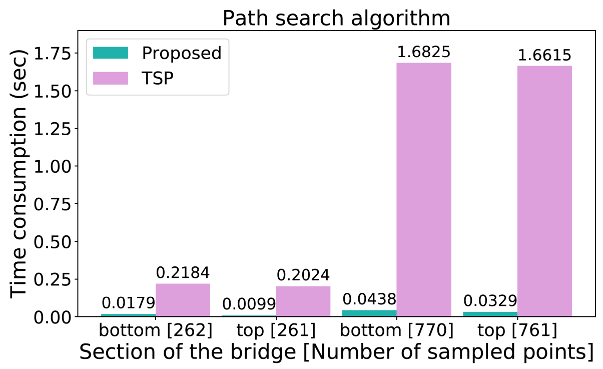

Fig. 13: Comparison between the proposed path search algorithm and the basic greedy Traveling Salesman Problem (TSP) solver. In the case of voxel size as 1.5 meters, the bottom side point cluster contains 262 points and the topside cluster contains 261 points. In the case of voxel size as 1 meter, the bottom side point cluster contains 770 points and the top side point cluster contains 761 points.

computation time of the proposed methods increases approx. 3 times as well. On the other hand, the TSP solver costs approx. 8 times.

The inspection task using autonomous drones prefers a simple and efficient global path guidance to ease the flight control and to maintain safety under uncertainties during the operation. Fig. 14 presents an example of the path searching result generated from the proposed method and the TSP solver. Both methods generate a traversal path that guarantees a complete visit for all points. The path length of the proposed result is 415.15 meters. Whereas the path length of the TSP result is 416.24 meters. The accumulated turning angle of the proposed result and the TSP result is 203.16

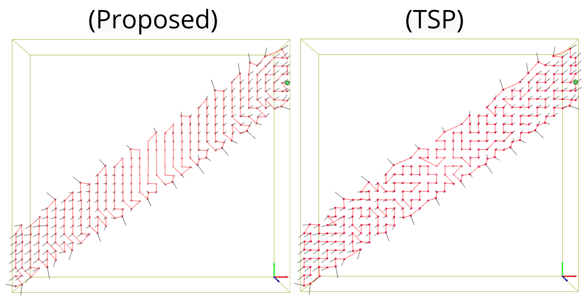

Fig. 14: Comparison of the path search results for the top side point cluster between the proposed path search algorithm and the basic greedy TSP method. The red lines represent the paths generated by two path search algorithms. The green sphere on the top-right corner denotes the starting point of the path.

and 331.42 radians respectively. While both of the methods provide an efficient path with regards to travel distance, our proposed path search method provides a simpler path structure than the TSP solver.

VI. Conclusion

This paper proposes a low complexity inspection path planning processing method for aerial vehicles via sampling-based sequential convex optimization. The point cloud generated from the 3D mapping service is used to represent the inspection target and used as the input of the proposed data processing chain. The proposed processing chain first uniformly samples the point cloud based on a voxel downsampling method. Then it is segmented based on the bounding box facets, and a graph is generated for each point cluster. The completeness of the inspection is guaranteed by the proposed traversal path search algorithm to sequentially visit and inspect each area of the inspection target. With the calculated sequence, a sequential convex optimization is formulated to find and optimize an inspection flight path subject to sensor limitations, efficiency, and safety requirements. The proposed method is validated with a dataset of a bridge and a power pylon. The computation time and the scalability of the method have been analyzed. The proposed method generates a complete and efficient inspection path within a reasonable time and shows linear scalability concerning the size of the inspection target.

VII. Acknowledgement

This project has received funding from the European Union's Horizon 2020 research and innovation project Drones4Safety under grant agreement No 861111 and the Innovation Fund Denmark project Drones4Energy with project number J. nr. 8057-00038A. We would like to thank Chiara Casarotti and Martina Mandirola from Eucentre for support with insights and data for the bridge inspection use case.

REFERENCES

[1] J. Seo, L. Duque, and J. Wacker, “Drone-enabled bridge inspection methodology and application,” *Automation in Construction*, vol. 94, pp. 112 – 126, 2018.

[2] M. Morita, H. Kinjo, S. Sato, T. Sulyyon, and T. Anezaki, “Autonomous flight drone for infrastructure (transmission line) inspection (3),” in *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, 2017, pp. 198–201.

[3] G. Mehrooz and P. Schneider-Kamp, “Optimal path planning for drone inspections of linear infrastructures,” *6th International Conference on Geographical Information Systems, Applications and Management*, pp. 326–336, 2020.

[4] S. Barba, M. Barbarella, A. Di Benedetto, M. Fiani, L. Gujski, and M. Limongiello, “Accuracy assessment of 3d photogrammetric models from an unmanned aerial vehicle,” *Drones*, vol. 3, no. 4, 2019.

[5] J. A. Cardoso and M. Louhaichi, “Protocol for data collection and processing from uavs imagery using opendronemap,” *Workflow for acquisition, processing and analysis of multidimensional multispectral and radar data for trial sites at CIAT headquarters*, pp. 326–336, 2019.

[6] M. Cubero-Castan, K. Schneider-Zapp, M. Bellomo, D. Shi, M. Rehak, and C. Strecha, “Assessment of the radiometric accuracy in a target less work flow using pix4d software,” in *2018 9th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2018, pp. 1–4.

[7] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258 – 1276, 2013.

[8] 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 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6423–6430.

[9] Liang Yang, Juntong Qi, J. Xiao, and Xia Yong, “A literature review of uav 3d path planning,” in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, 2014, pp. 2376–2381.

[10] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[11] D. Mellinger, A. Kushleyev, and V. Kumar, “Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2012, pp. 477–483.

[12] F. Baldini, S. Bandyopadhyay, R. Foust, S.-J. Chung, A. Rahmani, J.-P. de la Croix, A. Bacula, C. M. Chilan, and F. Hadaegh, “Fast Motion Planning for Agile Space Systems with Multiple Obstacles.” American Institute of Aeronautics and Astronautics (AIAA), sep 2016. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2016-5683

[13] B. Zhou, F. Gao, J. Pan, and S. Shen, “Robust Real-time UAV Replanning Using Guided Gradient-based Optimization and Topological Paths.” Institute of Electrical and Electronics Engineers (IEEE), sep 2020, pp. 1208–1214.

[14] C. Gao, Y. Kou, Z. Li, A. Xu, Y. Li, and Y. Chang, “Optimal multirobot coverage path planning: Ideal-shaped spanning tree,” in *2013 IEEE Aerospace Conference*, no. 10, 2018, pp. 1–8.

[15] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart, “A uav system for inspection of industrial facilities,” in *2013 IEEE Aerospace Conference*, 2013, pp. 1–8.

[16] E. Galceran and M. Carreras, “A survey on coverage path planning for robotics,” *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258 – 1276, 2013.

[17] ——, “A survey on coverage path planning for robotics,” *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, dec 2013.

[18] X. Zheng, S. Jain, S. Koenig, and D. Kempe, “Multi-robot forest coverage,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, 2005, pp. 2318–2323.

[19] R. Mannadiar and I. Rekleitis, “Optimal coverage of a known arbitrary environment,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2010, pp. 5525–5530.

[20] H. Choset and P. Pignon, “Coverage Path Planning: The Boustrophedon Cellular Decomposition,” in *Field and Service Robotics*. Springer London, 1998, pp. 203–209.

[21] H. Choset, K. Lynch, S. Hutchinson, and G. Kantor, *Principles of robot motion: theory, algorithms, and implementation*. The MIT Press, 2005.

[22] A. Zelinsky, A. Zelinsky, R. Jarvis, J. C. Byrne, and S. Yuta, “Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot,” *IN PROCEEDINGS OF INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS*, vol. 13, pp. 533—-538, 1993.

[23] J. van LEEUWEN, “Graph Algorithms,” in *Algorithms and Complexity*. Elsevier, jan 1990, pp. 525–631.

[24] C. WORMAN and J. M. KEIL, “Polygon decomposition and the orthogonal art gallery problem,” *International Journal of Computational Geometry and Applications*, vol. 17, no. 02, pp. 105–138, 2007.

[25] F. S. Hover, R. M. Eustice, A. Kim, B. Englot, H. Johannsson, M. Kaess, and J. J. Leonard, “Advanced perception, navigation and planning for autonomous in-water ship hull inspection,” *The International Journal of Robotics Research*, vol. 31, no. 12, pp. 1445–1464, oct 2012.

[26] C. H. Papadimitriou, “The Euclidean travelling salesman problem is NP-complete,” *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, jun 1977.

[27] S. Ramalingam, Y. Taguchi, J. K. Pillai, D. J. Burns, and C. R. Laughman, “Method for reconstructing 3d scenes from 2d images,” 04 2017, uS Patent 9,595,134.

[28] L. Liu and I. Stamos, “A systematic approach for 2d-image to 3d-range registration in urban environments,” *Computer Vision and Image Understanding*, vol. 116, no. 1, pp. 25 – 37, 2012, virtual Representations and Modeling of Large-scale Environments (VRML).

[29] F. Gao, L. Wang, B. Zhou, X. Zhou, J. Pan, and S. Shen, “Teach-Repeat-Replan: A Complete and Robust System for Aggressive Flight in Complex Environments,” *IEEE Transactions on Robotics*, pp. 1–20, may 2020.

[30] “NetworkX — NetworkX documentation,” accessed 2021-02-03. [Online]. Available: https://networkx.org/

[31] S. Diamond and S. Boyd, “CVXPY: A Python-embedded modeling language for convex optimization,” *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[32] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP solver for embedded systems,” in *European Control Conference (ECC)*, 2013, pp. 3071–3076.

[33] “dmishin/tsp-solver: Travelling Salesman Problem solver in pure Python + some visualizers,” accessed 2021-02-02. [Online]. Available: https://github.com/dmishin/tsp-solver