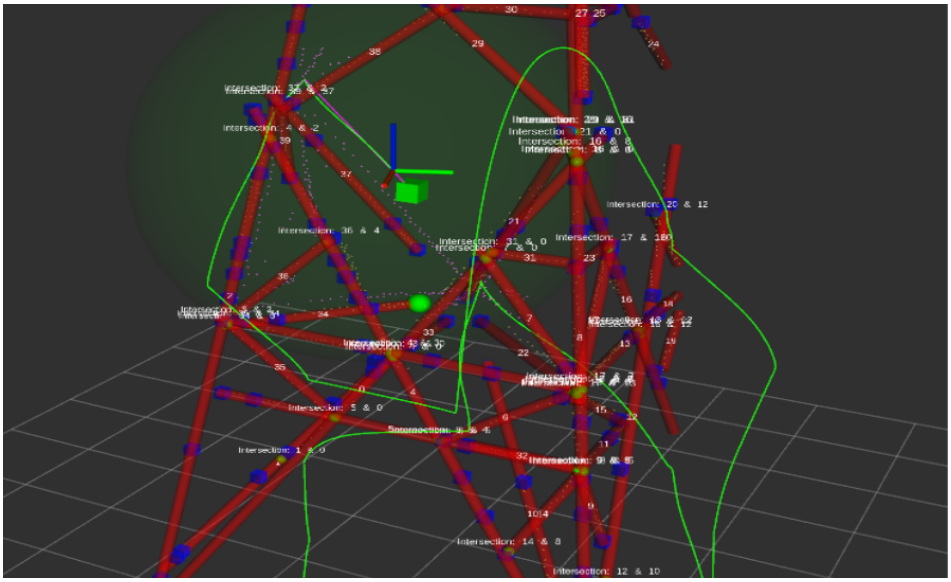
WS 2024-2025, SEL-EXC, DISAL-SP192

Start: 09.09.2024
Finish: 10.01.2025

Geometry-Informed Path Planning for Steel Structure Inspection

Zhu Zhengming



Professor: Alcherio Martinoli
Assistant: Lucas Walti

This page intentionally left blank.

Contents

Chapter 1 Introduction

The critical steel infrastructure that serves communities, such as energy transmission lines, telecommunication networks, steel bridges, and offshore platforms, requires regular inspections to ensure structural integrity and operational efficiency. Traditional inspection methods often rely on heavy machinery and visual assessments by technicians, putting them at risk in potentially dangerous conditions. Additionally, such inspections frequently result in downtime for the infrastructure, leading to inefficiencies and increased costs. The primary goal is, therefore, to detect signs of fatigue in the structure without requiring to stop its usage. Micro Aerial Vehicles (MAVs), equipped with visual and depth sensing technologies, present a promising solution to achieve this goal.

The previous work has primarily focused on sensor data processing with a forward-facing depth camera. They have studied the reconstruction of the structure using two methods, namely: 3D Hough transform and Iterative Closest Point (ICP) to map the structure and also compensate the drift accumulated over time in the state estimate of MAV by aligning consecutive measurements. These earlier studies emphasized mapping the structure and compensating for MAV drift. However, they did not tackle the critical aspect of path planning for MAV navigation to achieve autonomous full coverage of infrastructure.

In this work, I proposed a frontier-based navigation solution that utilizes the geometric understanding of the structure offered by the Hough transform and assesses how it allows the MAV to explore and cover the structure. The algorithm can be divided into two stages-one is evaluating the geometric information we have obtained, and another stage is path planning based on evaluation. This work will be carried out on a simulated MAV in Webots.

Chapter 2 Literature Review

2.1 Previous works

The implementation of this project was based on ROS system and simulated on Webots. Most codes are implemented in C++. In Webots, we used the MAV "Starling", which is equipped Time of Flight(ToF) sensors. The previous work has settled the segmentation and fusion on point clouds of steel structures, as well as improved Visual Odometry (VO) to enhance noise resilience caused by the drift of drone. However, the navigation strategy is pre-defined because the previous work only focus on sensor data processing. The ultimate goal is autonomous navigation by utilizing the processed information.

2.1.1 3D Iterative Hough Transform

To accurately detect lines within 3D point clouds, even in complex arrangements, the 3D Iterative Hough Transform[1] has demonstrated promising and robustness ability to accurately detect lines with 3D point clouds, which is well-suited for the steel beams. The point clouds were acquired from a ToF sensor and pre-processed by Point Cloud Library(PCL) to extract the valuable sensor data. The Iterative 3D Hough Transform algorithm detects lines in 3D point clouds by converting points into a parameter space. Principal Component Analysis(PCA) was applied for evaluating segments, retaining those met criteria.

2.1.2 Segments Fusion

The next part after detecting segments is fusion to reduce complexity, which includes integrating new segments with existing ones based on similar parameters, averaging segments with respect to PCA coefficients, and identifying intersections between segment by getting the shortest distance between them. As a result, this part[2] showed significant potential of improving autonomous steel structure inspection.

2.1.3 Noise Resilience

To tackle the major challenge of 3D Iterative Hough Transform that it is sensitive to noise, this work[3] focus on improving Visual Odometry(VO) using the Iterative Closest Point(ICP) algorithm and some variants like GICP to align point clouds. This method demonstrates significant enhancing the accuracy and reliability of steel structure inspection.

2.2 Related work

The problem is essentially an autonomous exploration so called active SLAM, which has been tackled from multiple perspectives. This work studied mainstream approaches to solve exploration problem from information theory perspective[4], they can be categorized as Traveling Salesman Problem(TSP)-based algorithm, Frontier-based algorithm and Partially Observable Markov Decision Process(POMDP)-based algorithm. Generally, the problem can be solved in two stages. The first stage could be evaluating sampled viewpoints to obtain more information from the environment, another stage is to generate optimal trajectory based on evaluation. Both parts may involve kinetics constrains and collision avoidance.

2.2.1 TSP-based algorithm

The Traveling Salesman Problem (TSP) involves finding the shortest possible route that visits each of a given set of locations exactly once. The coverage problem can also be solved as a TSP. In general, people solve the problem in two steps. In 2021 C.Cao[5] solved the problem by first sampling collision-free viewpoints uniformly in the local horizon and evaluate every candidate according to award function. Then they select a subset of viewpoints and find the order by maximizing an award function as an integer linear programming problem. [6][7][8][9] proposed several algorithms under this framework but they applied different approaches to sample and evaluate reward of viewpoints based the specific scene. However, for our scenario, it is not efficient to aimlessly sample viewpoint because we already extract useful geometric information like segments of steel structure.

2.2.2 Frontier-based algorithm

A common and straightforward formulation of the problem uses frontier, which is defined the boundary between covered and uncovered areas. When exploring, frontier guides robot towards the boundary between covered segments and uncovered segments, which extends the boundary of the covered areas until no more other uncovered area in the space. In 2016, Andreas Bircher[10] proposed The NBV-Planner algorithm, which applies RRT to

explore boundary, and executes in receding horizon. This method achieves real-time exploration in 3D environment. [11][12] also applied several methods to evaluate and get the frontiers with different value, so that algorithm is able to find the local goal point for the reason that most of them is greedy strategy.

2.2.3 POMDP-based algorithm

In robot exploration, the robot often has to explore an unknown environment while having limited information about the surroundings. POMDP-based algorithm is used to model and solve the exploration problem by helping the robot make decisions under uncertainty. The primary objective is to maximize the efficiency of information gathering, which involves balancing the trade-off between exploration (reducing uncertainty about unobserved areas) and exploitation (focusing on high-reward areas). The trick part also attaches much importance on the reward function, i.e, information gain function. Recently, it can also be solved by reinforcement learning to obtain ideal goal point.

Chapter 3 Methodology

3.1 System framework

In this work, previous research has successfully extracted important geometric features from steel structure. However, instead of relying on random viewpoint sampling, this work leverages segmentation information to guide the drone to explore and cover the steel structure. This approach significantly enhances exploration efficiency as opposed to randomly sampling by focusing on strategically evaluating key endpoints of the structure.

The core idea behind the approach is the usage of the geometric information identified through the segmentation phase. These outputs provide meaningful geometric information that can be utilized to guide the drone to explore. By evaluating the endpoints of these segments, the drone can more effectively decide the local goal point and plan its trajectory, avoiding unnecessary exploration of redundant or irrelevant areas. The frontier-based approach ensures that the drone only explores areas that offer the most valuable information gain.

The system is designed as a two-step process. In the first step, the drone subscribes to the output of the segmentation part, which provides 3D position information of all segments. This dataset is processed and evaluated with information gain, which serves as the basis for determining which areas of the environment require further exploration. In the second step, the processed information is fed into the path planning module, which utilizes the viewpoint set with evaluated value to generate an optimized exploration path. The planning process is guided by information gain from the viewpoint, ensuring that the drone focuses on areas with the highest potential for further information discovery. The framework of whole system is shown in Figure 3.1:

3.2 Frontier evaluation

This part of the algorithm is designed to evaluate the information gain value of segment endpoints in 3D space, serving as the basis for guiding the path planning process. By balancing the information gain with the path cost, the drone can prioritize positions that are expected to maximize the

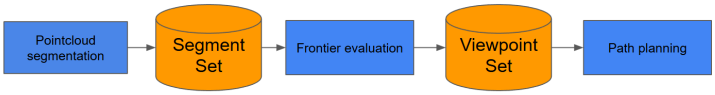


Figure 3.1: Overall pipeline

geometric information in the target region. This stage significantly enhances the efficiency of 3D exploration, avoids unnecessary redundant exploration, and ensures the autonomous navigation task in complex 3D environments.

3.2.1 Definitions

Segment model and frontier definition

The beam is modeled as segment with the following coefficients:

- $\vec{a_{seg}}$ is the starting point of segment.
- $\vec{b_{seg}}$ is the direction vector of segment.
- t_{min}, t_{max} are the parameters that correspond to the extremities of beam.

The segment is modeled by a ray line given as:

$$\vec{p} = t \cdot \vec{b_{seg}} + \vec{a_{seg}}, \quad t \in \mathbb{R}$$

.
Every segment has two endpoints, and the start point and end point are defined by:

$$\vec{e1_{seg}} = t_{min} \cdot \vec{b_{seg}} + \vec{a_{seg}}, \quad \vec{e2_{seg}} = t_{max} \cdot \vec{b_{seg}} + \vec{a_{seg}}$$

In the work, I defined both the start point and end point of segments as frontiers that we should pay attention.

FoV definition

The FoV will significantly affect the evaluation process, here defined the Fov as a frustum of pyramid, which can be represented as $FoV = (\alpha, \beta, d, p)$, the Figure 3.2 shows:

- α is the horizontal angular width of the FoV.

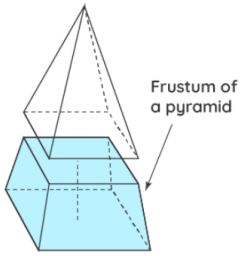


Figure 3.2: Definition of FoV

- β is the vertical angular width of the FoV.
- d is the sensor range.
- p is the vertex of the frustum.

It is worth noting that this defined FoV is used for evaluating frontiers instead of the real FoV of sensor. Therefore, $d_{define} < d_{real}$.

3.2.2 Pipeline

The **Algorithm 1** gives the procedure of frontier evaluation algorithm. For all frontiers, first, check whether each frontier is an endpoint and add the frontier in current FoV to an unordered set. Then, for each frontier that has been visited, sample viewpoints and evaluate them. Next, calculate the information gain for each viewpoint corresponding to the frontier. Finally, add the evaluated viewpoints to a set and pass it to next module. Figure 3.3 gives the pipeline of this stage.

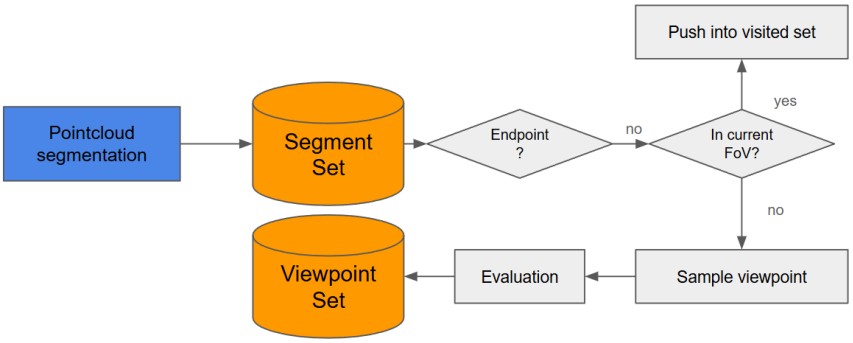


Figure 3.3: Frontier evaluation

Algorithm 1 frontier_evaluation Algorithm

Input: Set of frontiers $F = \{f_1, f_2, \ldots, f_n\}$, visited frontiers $V = \{v_1, v_2, \ldots, v_m\}$
Output: Set of evaluated viewpoints $\mathcal{V} = \{v_1, v_2, \ldots, v_p\}$
Initialize empty unordered_set $\mathcal{E}$ for endpoints
Initialize empty set $\mathcal{V}$ for evaluated viewpoints
for each frontier $f_i \in F$ **do**
    if is_endpointf_i **then**
        continue
    end if
    if is_in_cur_Forf_i **then**
        Add f_i to set $\mathcal{E}$
        continue
    end if
    Sample viewpoints from frontier f_i
    Calculate information gain for viewpoint v_j
    Add viewpoint v_j to $\mathcal{V}$
end for
Return the set of evaluated viewpoints $\mathcal{V}$

3.2.3 Information gain function

In the evaluation process, the information gain function represents the expected improvement in knowledge or certainty after performing an action or visiting a specific location in the environment. The information gain function can be used to guide the robot in selecting the most informative locations for exploration. The information gain function $Gain(q_k)$ at a query point $q_k \in \mathbb{R}^3$ is defined as:

$$Gain(q_k) = (\lambda_1 Vis(q_k, R_{search}) + \lambda_2 Den(q_k, R_{search})) \exp^{-\lambda_3 D(q_k, p_{cur}) + \lambda_4 W(q_k, p_{cur})}$$

Here, the $Gain(q_k)$ is composed of four key factors with respective weight that influence the overall information gain, which we will detail in the subsequent subsections.

Visibility gain

The visibility gain function $Vis(q_k, R_{search})$ represents the increased geometric information of the location q_k, where R_{search} denotes the area inside the FoV of viewpoint. It captures how much further visible information the drone can detect around q_k. The basic idea is to extend each segment until it intersects the boundaries of the FoV(five planes of the frustum), and the additional length that becomes visible is added to the total visibility gain. The steps of the algorithm are outlined below.

1. **Identity frontiers in** R_{search}:The drone only focus on the frontiers within the R_{search}. For each segment S_i, it needs to calculate how much additional part of the segment becomes visible if extending the segment to the boundary of R_{search}.

2. **Calculate possible intersection points**: The boundary of the FoV is represented by five hyperplanes (for the frustum in 3D), where each hyperplane represents a boundary of the FoV. For simplicity, I assumed that these planes are infinite in extent. For each segment S_i, extend the segment along its line in both directions. The goal is to calculate all potential intersection points between the extended segment and the five boundaries of the FoV.

3. **Find the true intersection**: To calculate the true intersection point of a segment with R_{search}, we can simply select the possible intersection that is closest to the frontier.

This process can be expressed in 2D for better readability as shown in Figure 3.4, in this case, red lines are covered segments, $Vis(qk, R_{search})$ is the overall length of blue dash lines:

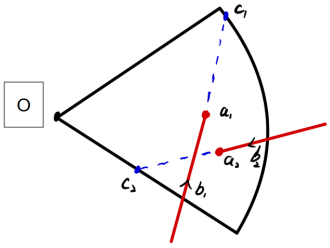


Figure 3.4: Visibility gain calculation in 2D

To mathematically formulate the intersection between a ray line and a hyperplane, let the equation of a hyperplane be given by:

$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{p_0}) = 0$$

where:

- $\mathbf{n}$ is the normal vector to the plane.
- $\mathbf{r}$ is the position vector of a point on the plane, it could be the intersection.
- $\mathbf{p_0}$ is the known point on the plane like the vertex of the frustum.

Because $r = \vec{a} + t\vec{b}$, we can obtain the intersection parameter:

$$t = \frac{\mathbf{n} \cdot (\mathbf{p_0} - \mathbf{a})}{\mathbf{n} \cdot \mathbf{b}}$$

We need to make sure $t > 0$ for the reason that we are only interested in the extended length of the segment.
For each segment, once we have computed the intersections of the extended segment with the five hyperplanes, we select the closest intersection point to the endpoint of segment P_{end}. This point represents the new visible boundary of the segment within the FoV. The difference between the original length of the segment and the distance to this intersection point gives the uncovered length that can now be observed from q_k, let the uncovered length be:

$$L_{uncovered} = ||P_{end} - P_{intersection}||$$

The total visibility gain at the viewpoint q_k is then the sum of the uncovered lengths for all segments within the search region R_{search}:

$$Vis(q_k, R_{search}) = \sum_i L_{uncovered}(S_i)$$

Density gain

The density gain function aims to evaluate how much useful information can be obtained from regions where there is a higher concentration of potential unexplored segment. Regions with a higher density on the frontier often represent more opportunities for exploration and greater reward. Therefore, the drone should prioritize regions where the density of unexplored segments is high.
To compute the Density Gain, we evaluate the segment density within the search region R_{search}. The density at a point q_k is influenced by how many unexplored frontiers exist within the R_{search}. Let us define the density function $Den(q_k, R_{search})$ as the sum of frontier number within FoV of the viewpoint.

Distance punishment

The Distance Punishment function penalizes exploration towards regions that are farther from the current position of drone. The farther away a region is, the less desirable it is to explore it, because it will require more time and energy for the robot to travel to and explore it. To be straightforward, I simply define the distance punishment function $D(q_k, p_{cur})$ as Euclidean distance:

$$D(q_k, p_{cur}) = ||q_k - p_{cur}||$$

P_{cur} is the current position of drone. By incorporating the distance punishment term into the Gain function (as part of the exponential term), faraway viewpoints have an exponentially lower likelihood of being selected. This punishment term decreases the visibility gain and density gain of faraway frontiers, encouraging the drone to prioritize closer regions.

Z-axis Distance weighting

The Z-axis distance weighting function is designed to penalize exploration at elevations significantly different from the current height of the drone. In the context of a drone inspecting a steel infrastructure, it is beneficial to prioritize frontiers that lie at the same or a similar horizontal level. This reduces energy consumption by minimizing unnecessary vertical displacement, while also improving its perception of the environment in a structured and layered manner. The approach drives drone cover the structure layer by layer in an upward progression, which is particularly effective for maintaining the stability and consistency of the overall structural segmentation. We defined the Z-axis Distance Weighting function $W(q_k, p_{cur})$ as follows:

$$W(q_k, p_{cur}) = exp(|z_{q_k} - z_{p_{cur}}|)$$

Also, the absolute height difference is then exponentiated to create a penalty function that grows exponentially with the difference in height. This ensures that the farther the frontier is from the current elevation, the more it contributes to the penalty.

3.2.4 Viewpoint sampling

In this part, we describe the approach for viewpoint sampling around each frontier during the evaluation process. The viewpoint sampling is crucial for identifying the most optimal inspection position that maximizes the information gain by capturing useful data from the environment. I applied the $Vis(q_k, R_{search})$ function to evaluate viewpoint, which helps select the most advantageous viewpoint for further exploration.
Given that the drone is assumed to remain in a flat state during exploration, the viewpoints for each frontier are sampled in a 2D circular region around the frontier. The viewpoints are defined as points in 3D space with the parameters (x, y, z, θ), where:

- (x, y, z) are the coordinates of the viewpoint in 3D space.
- θ is the orientation of the drone at that viewpoint.

For each frontier, a set of viewpoints is sampled in a circular region centered on the frontier. This ensures that the viewpoints are distributed symmetrically around the frontier and represent different perspectives of the target

area. The parameters here need to be determined are *radius* and N_{view}. After that, we need to filter out viewpoints that would lead the drone into a collision with obstacles. The collision detection step ensures that only safe viewpoints are retained for further evaluation, which will be illustrated in later chapter. Finally, choosing the most valuable viewpoint for every frontier as the candidate local goal point and pushing it into the viewpoint set.

3.2.5 Update

Record visited viewpoints

It is crucial to keep track of the visited frontiers to avoid redundant computation. This reduces the computational burden by ensuring that the robot does not waste time and resources revisiting areas it has already explored. I managed the explored frontiers with hash table to handle the frontiers that have been visited. a coordinate (x, y, z) of a frontier could be hashed into a unique value to represent its presence in the set.
To achieve this, I assumed that once a frontier has been explored, the segments in the FoV of the drone will no longer extend anymore, i.e., they can be considered as endpoints of segments that will not grow.

3.3 Path planning

3.3.1 Pipeline

With the evaluation stage of exploration completed, the next step is to utilize the processed viewpoint set-with each viewpoint carrying an evaluated information gain to perform hierarchical path planning. This section describes the path planning pipeline, as shown in the provided in Figure 3.5, and outlines the hierarchical strategy, the use of priority queues, the RRT* path planner, and trajectory optimization to maintain orientation toward the steel structure.
The algorithm of path planning will be illustrated in **Algorithm 2**.

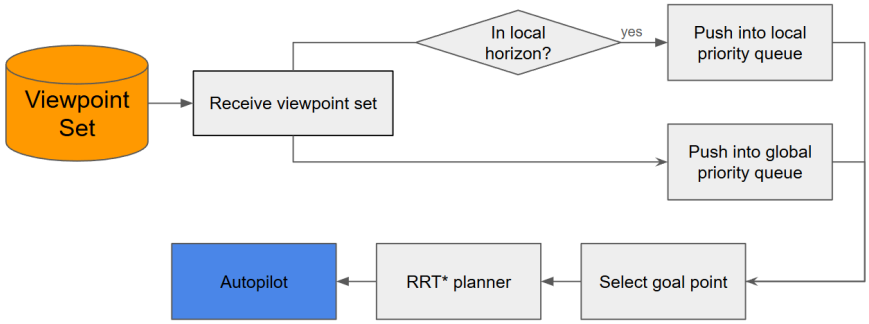


Figure 3.5: Path planning pipeline

Algorithm 2 path_planning Algorithm

Input: Viewpoint set $\mathcal{V}$
Output: Optimized trajectory for execution
Initialize empty priority queue $\mathcal{Q}_{local}$ for local viewpoints
Initialize empty priority queue $\mathcal{Q}_{global}$ for global viewpoints
for each viewpoint $v_i \in \mathcal{V}$ **do**
    if v_i is in local horizon **then**
        Push v_i into $\mathcal{Q}_{local}$
    end if
    Push v_i into $\mathcal{Q}_{global}$
end for
Initialize goal_point $\leftarrow$ null
if $\mathcal{Q}_{local}$ is not empty **then**
    goal_point $\leftarrow$ highest-priority viewpoint in $\mathcal{Q}_{local}$
else
    goal_point $\leftarrow$ highest-priority viewpoint in $\mathcal{Q}_{global}$
end if
Generate trajectory using RRT* planner
Optimize the generated trajectory
Pass the optimized trajectory to the autopilot

3.3.2 Hierarchical goal point decision making

The path planning pipeline follows a hierarchical approach that separates the viewpoints into the global and the local set. This distinction ensures that the drone prioritizes local exploration (within its local horizon) before considering farther viewpoints, which contributes to both exploration efficiency and reduced energy consumption. The hierarchical strategy proceeds as follows:

Viewpoint Categorization:

- All unexplored viewpoints are stored in a global priority queue.
- Among these viewpoints, the subset of viewpoints within the local horizon is identified and stored in a separate local priority queue. The local horizon represents a predefined region around the drone like a sphere or a horizontal band.

Select goal point:

Both the global and local queues are implemented as priority queues, with the information gain serving as the key for sorting. Viewpoints with higher information gain are given higher priority in the queue. By maintaining separate queues, the system allows the planner to first prioritize and process viewpoints in the local horizon before falling back to the global queue if no high-value local options remain.

3.3.3 RRT* planing

With the goal point determined, the RRT* (Rapidly-exploring Random Tree Star) planner is invoked to generate a collision-free path from the current position of the drone to the selected goal point. Here I will illustrate vital part-collision check and an improvement regarding to the tower-like structure.

Collision check

In this setup, collision detection is performed by treating both the trajectory branches and the segments of the steel structure as cylinders. This approach simplifies the geometric computations needed to check for potential collisions.
A branch of the trajectory generated by RRT* is modeled as a cylinder. And the radius of the cylinder is equal to the safety distance around the drone, accounting for its physical dimensions and a margin for safe operation. This step essentially checks the surface intersection between two cylinders shown in Figure 3.6:

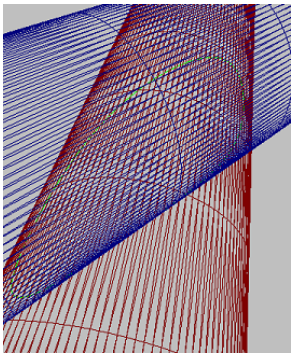


Figure 3.6: Collision check

The collision check is solved by *Geometric Tools for Computer Graphics*. Available at: https://www.geometrictools.com/index.html.

Horizon-first sampling

Here is a sampling optimization of RRT* regarding to the horizon-first path finding. In the horizontal plane (x, y), the sampling is performed uniformly across the entire feasible space. This ensures that the RRT* planner explores the full range of potential paths in the horizontal plane. While in the vertical direction (z), the sampling is biased using a Gaussian distribution centered around the current elevation of drone to encourage RRT* to find the trajectory horizontally.
For a sampling point (x, y, z):

- $x \sim \text{Uniform}(x_{\min}, x_{\max})$
- $y \sim \text{Uniform}(y_{\min}, y_{\max})$
- $z \sim \mathcal{N}(z_{\text{cur}}, \sigma_z^2)$

The RRT* algorithm generates a path P as a sequence of waypoints. It is worth considering that, to avoid restricting the sampling capability of RRT* in the vertical direction. σ should not to be too small.

$$P = \{p_0, p_1, \cdots, p_n\}$$

3.3.4 Active orientation control

In order to enhance the odometry of drone and the overall segmentation, we should optimize the trajectory for maintaining visibility of the covered structure. First, system needs to compute the center axis of the structure. This is finished by weighting the positions of frontiers based on their relative vertical distances from the current height of drone. The goal is to assign

lower importance to frontiers at elevations farther from the current plane of operation.

Weighted center axis computation

The geometric information of known segments can be represented by own endpoints. Here we utilized the 3D position of endpoints directly to calculate the weighted center axis. For a set of frontiers $F = \{f_i = x_i, y_i, z_i\}$, the weighted center axis (x_c, y_c) is computed as follows:

$$\omega_i = \exp -\lambda_z \cdot |z_i - z_{cur}|$$

$$x_c = \frac{\sum_{i=1}^{N} w_i \cdot x_i}{\sum_{i=1}^{N} w_i}, \quad y_c = \frac{\sum_{i=1}^{N} w_i \cdot y_i}{\sum_{i=1}^{N} w_i}$$

Trajectory optimization

Once the center axis of known structure is determined, the next step is to interpolate the trajectory generated by the RRT* algorithm and extend the waypoints to include *yaw* orientation. This ensures that the drone maintains visibility of the structure as it moves along the trajectory. The path $\hat{P}$ is a set of waypoints that $P_{optimized}$ can be represented as:

$$P_{optimized} = (x, y, z, \theta), \quad \text{where} \quad \theta = arctan2(y_c - y, x_c - x)$$

The process shows as Figure 3.7. Note that the last waypoint of optimized trajectory $\hat{P}$ should head to the target orientation instead of interpolated orientation.

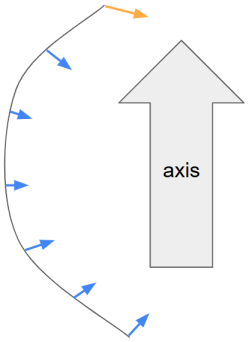


Figure 3.7: Interpolation

Chapter 4 Experimental setup

4.1 Implementation

4.1.1 Webots

In our project, we use Webots to simulate the drone environment, which features an electric pole designed to represent the steel structure of long-distance electrical networks. Figure 4.1 provides a view of the simulated environment:

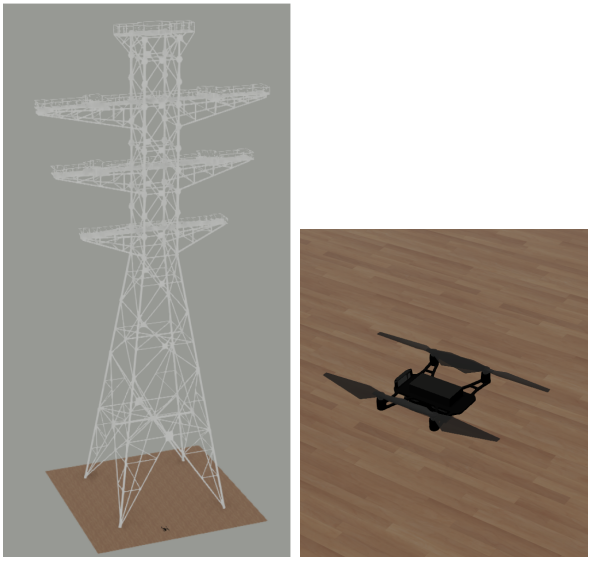


Figure 4.1: Webots simulation

The drone used in this simulation is based on the Starling drone model. It is equipped with a single camera capable of capturing images at 10 frames per second (fps) and a Time-of-Flight (ToF) sensor operating at 5 fps. For orientation and acceleration tracking, the drone relies on an Inertial Measurement Unit (IMU). Additionally, the position of drone is precisely provided by the Webots supervisor, ensuring accurate navigation throughout the simulation.

4.1.2 ROS

The project is implemented in ROS Noetic and interact with other components such as the auto pilot. There is some ROS nodes including **frontier_evaluation**, **optimal_path_NBV**, **visualization** and **metrics**.

ROS message

In this package, I defined two ROS messages **Segment** and **Viewpoint** for convenient data exchange among ROS nodes:

- **Segment**:
  int32[] seg_id
  geometry_msgs/Point[] starts
  geometry_msgs/Point[] ends

- **Viewpoint**:
  geometry_msgs/Pose[] pose
  float64[] info_gain

ROS nodes

The ROS nodes graph is shown by Figure 4.2:

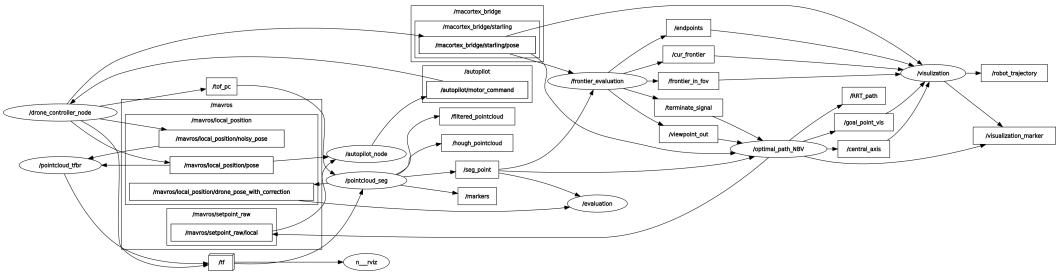


Figure 4.2: ROS node graph

TF relationship

The TF relationship of the ROS system is shown in Figure 4.3, the world frame is a legacy from the real MAV and that it designates the ToF camera frame.

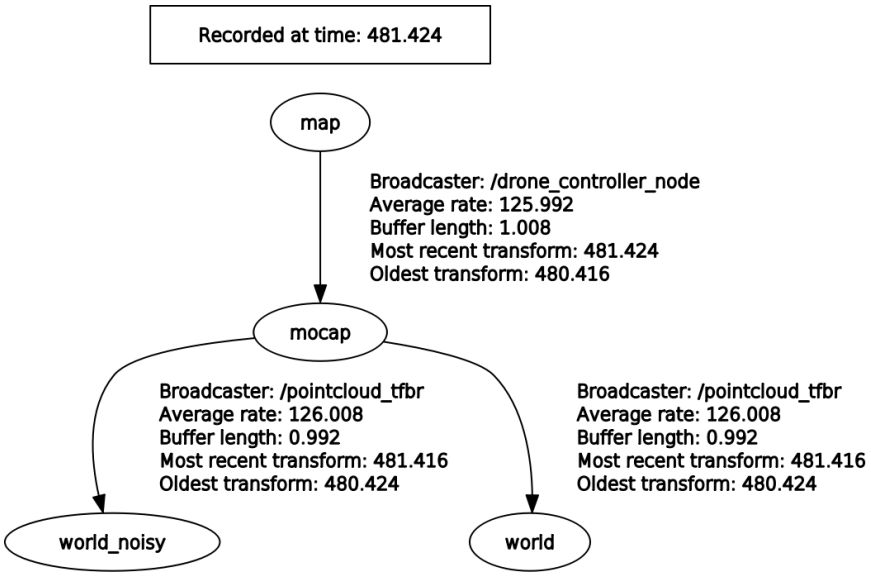


Figure 4.3: TF

4.1.3 Third-party library

To enhance the functionality of our project and streamline the implementation process, I incorporated the following third-party libraries:

Matplotlib

A widely used Python library for data visualization, was employed for analyzing and visualizing the results of the simulation. Using Matplotlib-cpp, I generated more understandable results for analysis in C++.

Geometric tools library

The Geometric Tools Library was utilized for implementing collision check. This library provides efficient and robust methods for geometric computations, including detecting intersections between complex shapes such as cylinders, planes, and other 3D objects. In our project, it played a crucial role in ensuring safe and feasible trajectory planning by accurately checking for potential collisions between the trajectory and the steel structure.

4.2 Metrics

4.2.1 Experiment setting

Hardware

- CPU: Intel(R) Core(TM) i7-7700@3.6GHz
- Cores:8
- Architecture: 64 bits

Fixed parameters

- Exploration timeout: $800s$
- FoV: $[\alpha = \pi/3, \beta = \pi/3, d = 1.5m]$
- Space size: $5m \times 5m \times 18m$
- max velocity: $1.0m/s$
- max acceleration: $0.8m/s^2$

4.2.2 Metrics definition

- **Exploration efficiency**: This metric calculates the max exploration rate, i.e. the max increased length of all segments within one second.

$$E_{rate} = \frac{\Delta L_{segments}}{\Delta t} \quad \text{where} \quad \Delta t = 1s$$

- **Probability of successful exploration**: This metric evaluates the likelihood that the drone successfully completes its exploration task within the limited time.

$$P_{success} = \frac{N_{success}}{N_{total}}$$

- **Total exploration time cost**: This metric records the time cost when drone has finished the exploration task.

$$T_{total} = T_{end} - T_{strat}$$

- **Energy consumption**:This metric evaluates the energy consumption under different strategies or parameters. [13] suggests that the induced drag plays a significant role in drag generation. Besides, we also need to consider the energy consumption caused by overcoming gravity. Therefore, after ignoring hover energy consumption, the drone will

overcome gravity and induced drag when moving up, only overcome induced drag when moving down and free to go down.

$$E_{total} = E_{up} + E_{down} + E_{horizon}$$

$$E_{up} = \int_0^{d_{up}} (F_{gravity} + F_{drag}) ds$$

$$E_{horizon} = \int_0^{d_{up}} F_{drag} ds, \quad E_{down} = 0$$

This induced drag is directly caused by the rotating blades of the MAV as they move through the air, which is modeled as:

$$F_{drag} = -\tau \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{v}_a$$

Where $d = 0.024 s/m$, $\tau = 2.89N$ is the thrust, $\mathbf{v}_a = 1m/s$ is the stable speed when exploration.

- **Algorithm efficiency**:This metric evaluate the algorithm response speed during the exploration task, here I divided it into two parts, one is the max frontier evaluation time cost $E_{evaluation}$, another is the max path planning time cost $E_{planning}$. The max execution time shows the ability of responding the worst situation, which significantly decides the performance of algorithm.

$$E_{evaluation} = \max T_{evaluation}$$

$$E_{planning} = \max T_{planning}$$

Chapter 5 Results

5.1 Algorithm performance

This section is mainly evaluating the performance of whole system. The key parameters are set as follows:

- Mode: horizon-first planning
- Jump size $= 0.5m$
- $\lambda_1 = 6.0, \quad \lambda_2 = 0.5, \quad \lambda_3 = 1.5, \quad \lambda_4 = 4.0$

5.1.1 Exploration efficiency

According to Figure 5.1, we can get the max exploration rate is $6.993m$ length increasing per second under the pre-defined parameters, which suggests the exploration efficiency.

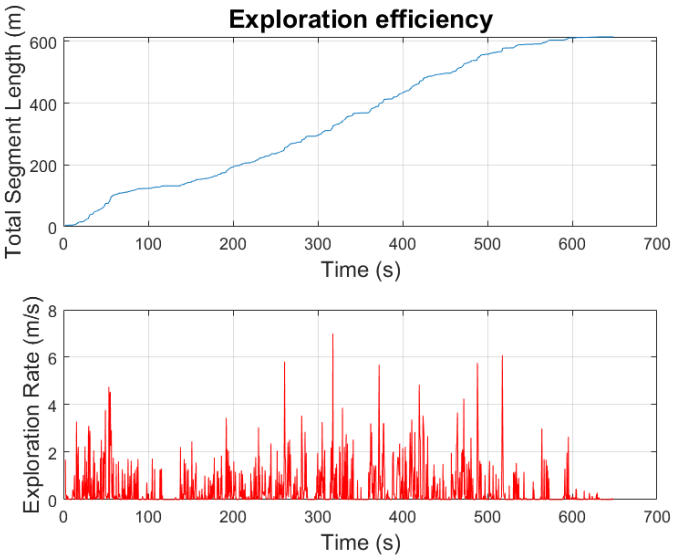


Figure 5.1: Total segment length and Exploration rate over time.

5.1.2 Probability of successful exploration

We have experimented 10 times to measurement the probability of successful coverage within the fixed time limitation. As shown in Table 5.1, the success rate is 70%, and it may fail because of partial exploration or collision with segments.

Experiment number	**Result**	**Total time cost**
1	Succeed	490s
2	Fail	partial exploration
3	Succeed	440s
4	Succeed	440s
5	Succeed	427s
6	Fail	collision with segments
7	Succeed	481s
8	Succeed	474s
9	Succeed	489s
10	Fail	collision with segments

Table 5.1:

5.1.3 Total exploration time cost

The total exploration time cost is 463s averaged from successful explorations. And the standard deviation is 24.51s.

5.1.4 Algorithm efficiency

The algorithm is divided into two parts where one is evaluation time cost, and another one is planning time cost. The Figure 5.2 and Table 5.2 shows that the average evaluation time cost is 38.92ms, executed 10517 iterations, the average planning time cost is 955.88ms, executed 614 iterations. Besides, we also need to consider the max response time of both parts. They show the ability of handling the worst situation.

- The max evaluation time cost: 152ms
- The max planning time cost: 1984ms

	mean	std	min	max	total time
Evaluation efficiency	38.92ms	25.28ms	0	152ms	409s
Planning efficiency	955.88ms	247.28ms	0	1984ms	586s

Table 5.2:

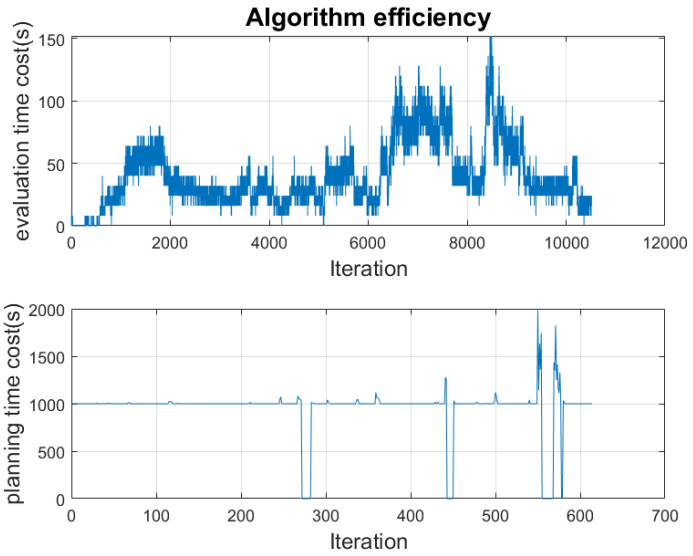


Figure 5.2: Shows evaluation time cost and planning time cost every iteration through entire exploration task.

5.2 Comparison Experiment

This section compares the performance of different settings, which shows how specific parameters or strategies affect the system regarding steel structure inspection task.

5.2.1 Horizon-first planning vs Vicinity-first planning

Horizon-first Planning and Vicinity-first planning are different strategies when selecting goal point in the local horizon. The local horizon of the former is a horizontal band with given width, whereas the latter has a sphere local horizon. Given that:

- Spherical radius: $2.5m$
- Horizontal band width: $2.0m$
- Jump size $= 0.5m$
- $\lambda_1 = 6.0, \quad \lambda_2 = 0.5, \quad \lambda_3 = 1.5, \quad \lambda_4 = 4.0$

Impact of Trajectory on the Structure's Reconstruction

As shown in Figure 5.3, the trajectories of different strategies are significantly different. The Horizon-first planning drives drone mapping from the bottom to the top and layer by layer, whereas the Vicinity-first planning makes drone explore randomly. The latter strategy may cause the offset of segments.

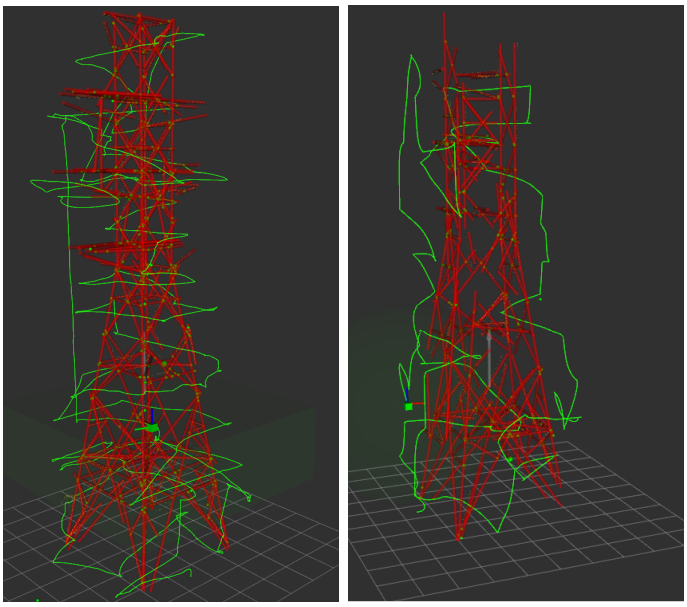


Figure 5.3: Left:Horizon-first; Right:Vicinity-first

Energy consumption

Here we ignored the energy consumption caused by hovering. It is known to us that driving drone moving up costs the most energy comparing to other motions. Obviously, the Vicinity-first planning will cost more energy, because drone will repeat up and down movements more often. The induced drag can be calculated from the result in [13] after reading thrust in z-axis. The comparison is shown in Figure 5.4.

- Horizon-first planning: $E_{total} = E_{up} + E_{horizon} = 106.39J$
- Vicinity-first planing: $E_{total} = E_{up} + E_{horizon} = 154.27J$

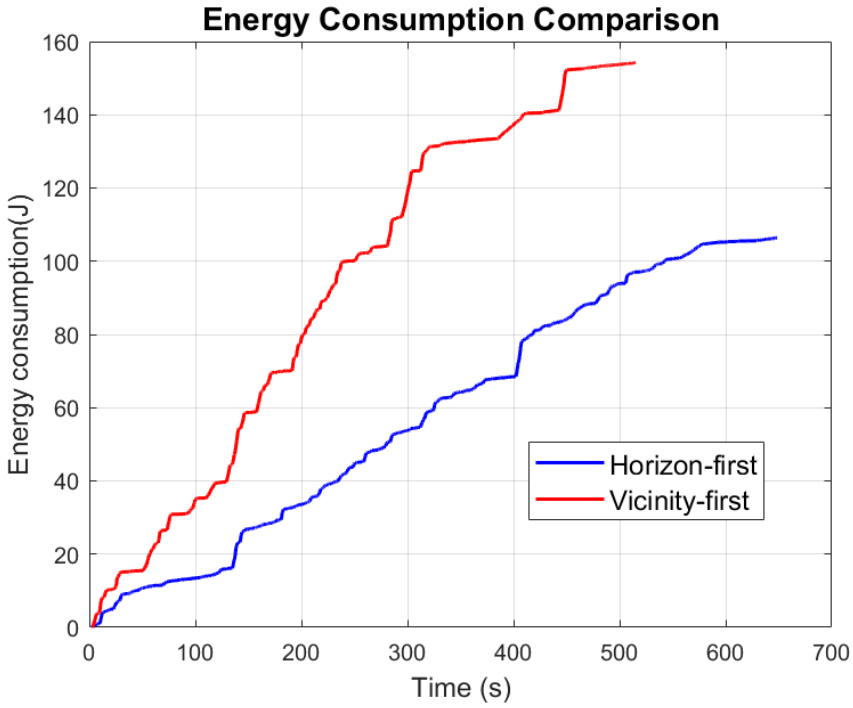


Figure 5.4: Comparison of energy consumption between Horizon-first planning and Vicinity-first planning.

5.2.2 RRT* jump size

The jump size of RRT* decides the max length of extended branch, which is an important parameter when searching feasible trajectory. Here I experimented the task with jump_size $\in \{0.2m, 1.0m\}$.

Exploration efficiency

The exploration efficiency depends a lot on the path planning part, As one can see from Figure 5.5 and Table 5.3, the system with smaller jump size has a higher exploration rate.
In complex steel structures, a smaller jump size increases the probability of finding a feasible path and reduces the impact of local traps, also it is not easy to miss some complicated corners.

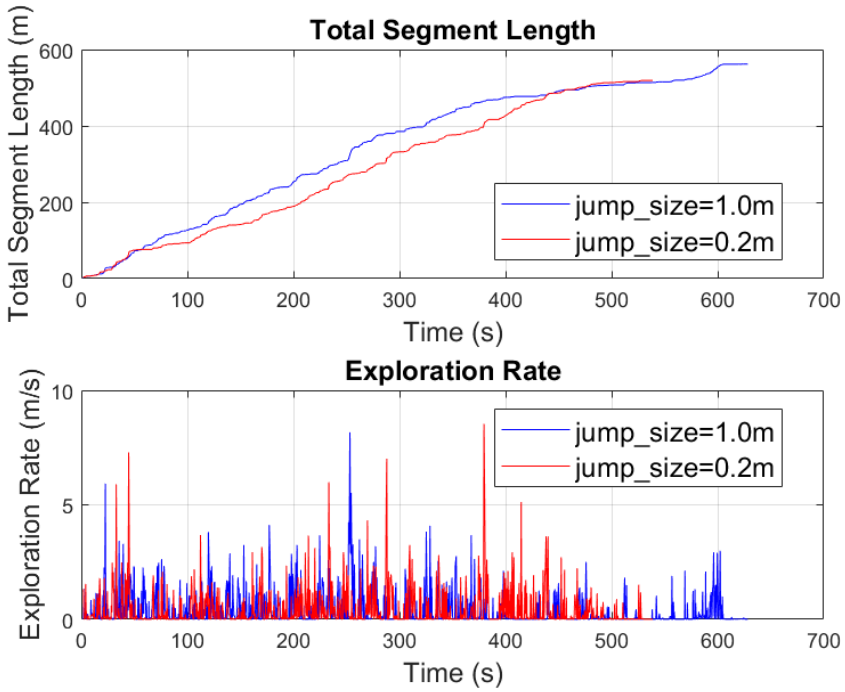


Figure 5.5: Comparison of exploration efficiency between $jump\_size = 0.2m$ and $jump\_size = 1.0m$.

Algorithm efficiency

Also, this parameter affects both the evaluation time cost and planning time cost. The result is also shown in Figure 5.6 and Table 5.3.
It makes sense that smaller jump size causes longer planning time cost for the worst situation. The max evaluation time cost of result with $jump\_size = 1.0m$ is $224ms$, which is longer than another one. The reason is that the drone left more unvisited frontiers during exploration process. Besides, the planning time cost with $jump\_size = 0.2m$ has a larger standard deviation, because it can easily find a path nearby but struggle to generate a trajectory quickly when goal point is far from current position.

Jump size	Exploration rate	Max evaluation time	Max planning time
$0.2m$	$8.53m$	$192ms$	$2176ms$
$1.0m$	$8.16m$	$224ms$	$1096ms$

Table 5.3:

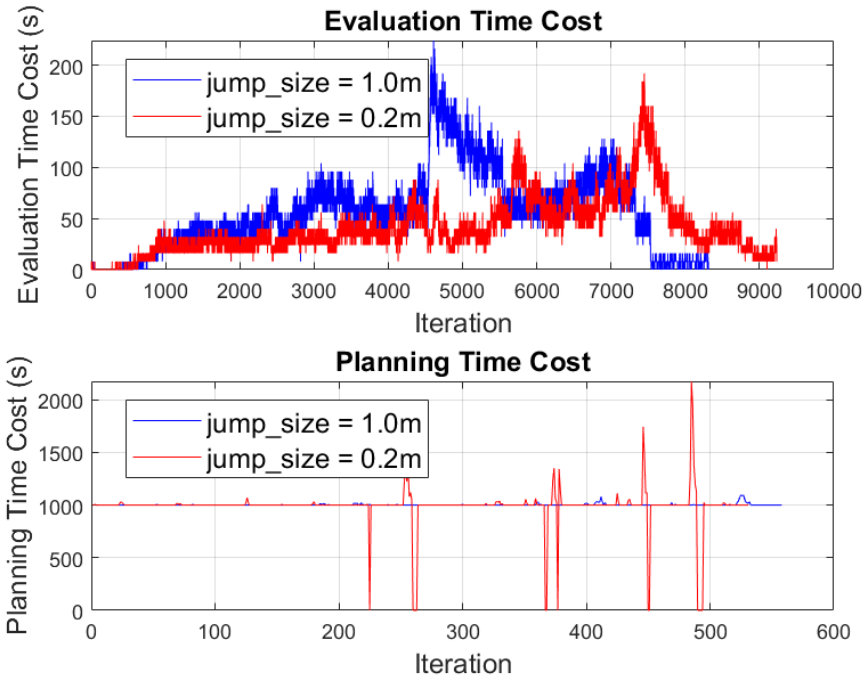


Figure 5.6: Comparison of evaluation time cost and planning time cost between $jump\_size = 0.2m$ and $jump\_size = 1.0m$.

5.2.3 Visibility gain weight

We want to measure how the weight of different factors affect the performance of system. Here I focus on the weight of visibility gain. Recall the information gain function:

$$Gain(q_k) = (\lambda_1 Vis(q_k, R_{search}) + \lambda_2 Den(q_k, R_{search})) \exp^{-\lambda_3 D(q_k, p_{cur}) + \lambda_4 W(q_k, p_{cur})}$$

We tried two experiments with high weight $\lambda_1 = 6.0$ and low weight $\lambda_1 = 0.5$ respectively.

Exploration efficiency

As shown in Figure 5.7 and Table 5.4, the exploration rate is higher when setting a higher weight of visibility gain $\lambda_1 = 6.0$, because a drone can explore the environment intentionally based on the evaluation of the information gain.

λ_1	**Exploration rate**	**Max evaluation time**	**Max planning time**
6.0	$6.993m$	$152ms$	$1984ms$
0.5	$6.003m$	$216ms$	$1552ms$

Table 5.4:

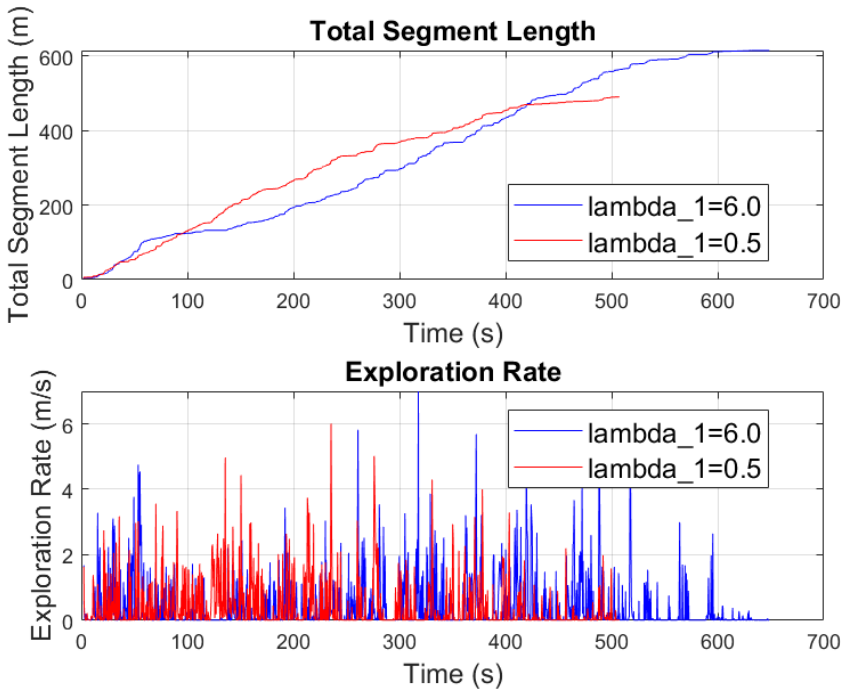


Figure 5.7: Comparison of exploration efficiency between $\lambda_1 = 6.0$ and $\lambda_1 = 0.5$

Algorithm efficiency

With a higher weight assigned to the visibility gain during the calculation of information gain, the drone's exploration becomes more purposeful, driving it toward areas with the highest potential for information gain. This targeted approach ensures that the drone prioritizes regions with a higher density of boundary points, leading to a more efficient exploration strategy. Consequently, the overall number of unvisited frontiers requiring evaluation decreases significantly, which significantly decides the evaluation time cost throughout the exploration process.

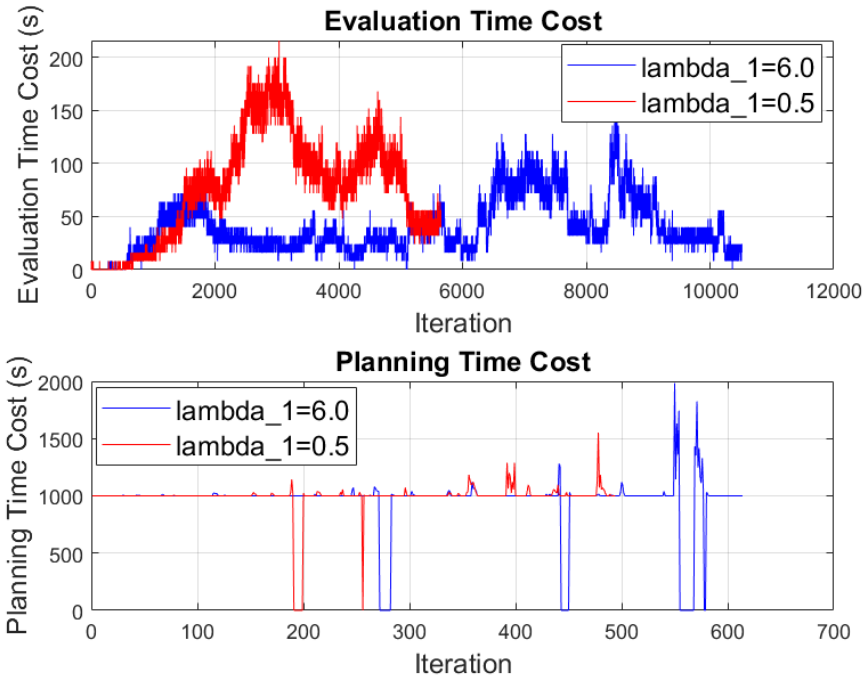


Figure 5.8: Comparison of evaluation time cost and planning time cost between $\lambda_1 = 6.0$ and $\lambda_1 = 0.5$

Chapter 6 Discussion

6.1 Conclusion

In this project, I primarily implemented a path planning algorithm based on the geometric information provided by previous work. The core idea of the algorithm revolves around leveraging the processed geometric data to guide the drone in active exploration and ensure complete coverage of the steel structure.

The main concept behind this work is a frontier-based algorithm, which is divided into two key stages. The first stage involves evaluating and managing the frontiers that can be utilized by the drone, based on an information gain function that I designed. This function helps assess the potential usefulness of each frontier, and the output is a set of viewpoints with associated information gain values. These viewpoints are then passed on to the second stage-path planning.

In the path planning stage, I employed a hierarchical strategy that utilized a priority queue to manage the viewpoints according to their information gain. The strategy prioritizes decision-making within the local horizon, where the drone first aims to visit all viewpoints within this region before expanding to the global horizon. Once the local horizon is fully explored, the algorithm selects target viewpoints from the global horizon.

To generate obstacle-free paths, I applied the RRT* planner, ensuring efficient path generation in complex environments. Additionally, to avoid losing perception of the steel structure, I optimized the generated trajectory by interpolating additional orientation towards the axis of covered structure.

The method successfully guides the drone in its exploration task, offering a systematic approach to covering large, complex structures efficiently.

6.2 Limitations

6.2.1 Segment shift

During the exploration process, the segments may experience shifts due to the accumulative drift that occurs over time. The accumulated drift of drone can lead to the misalignment of perceived segments, which may result in an inaccurate segmentation of the exploration area. Although this issue can be mitigated, it remains a significant limitation in the current methodology. Figure 6.1 shows this situation:

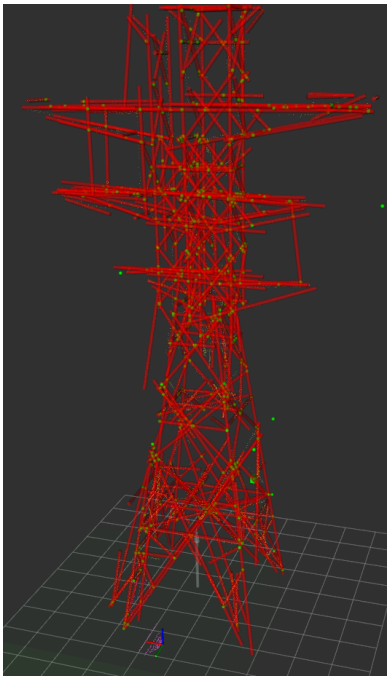


Figure 6.1: Shift

6.2.2 Stuck in segments

Another limitation arises from the field-of-view (FoV) constraints of the forward-facing sensors. These limitations can cause situations where the robot gets "stuck" within the middle of segments. Due to the restricted range of the sensors, certain areas of the environment might not be detected or properly mapped, especially when the drone is navigating through a complex steel structure. This issue can lead to incomplete mapping the steel structure, as Figure 6.2 shows.

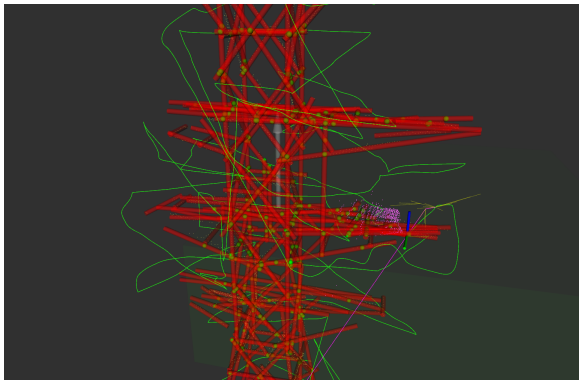


Figure 6.2: Stuck

6.2.3 Partial coverage

A third limitation involves the possibility of the exploration process terminating prematurely shown in Figure 6.3. This situation involves several factors. For example, the system sampled some bad viewpoints, which did not provide much new geometric information. Sometimes, the offset of segments causes more new frontiers at visited position, so that the drone always moves back and forth till it exceeds the exploration time limitation. The RRT* planer searching time out also causes exploration task end in advance. Therefore, the appropriate parameters affect the performance of the system a lot.

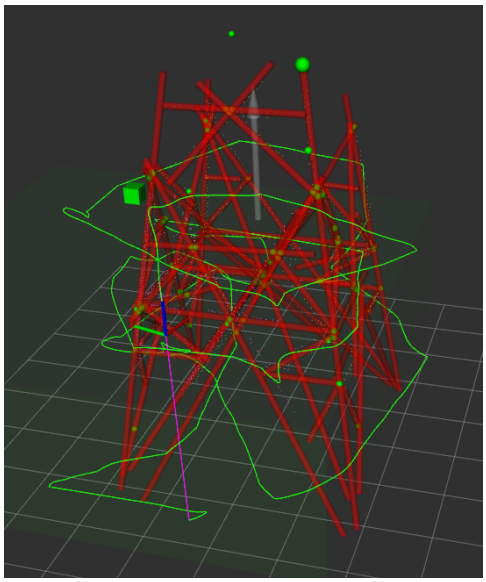


Figure 6.3: Partial coverage

6.3 Future improvements

According to the results of the experiment, we found that there are also some aspects need to be improved in the future work.

6.3.1 Limit FoV

limited Field-of-View (FoV) in drone can cause incomplete perception of the environment, increasing the likelihood of collisions or incomplete coverage, especially in complex environments. Here is a potential solutions for this issue like using a swarm system, where drones collaborate to cover a larger area. Drones with limited FoV can communicate with each other and adjust their paths to ensure complete environmental coverage. In that case, it can reduce blind spots and increasing the overall coverage.

6.3.2 Drift

The drift issue arises due to factors like sensor noises, and errors in localization. The previous work only improved Visual Odometry (VO) to enhance noise resilience caused by the drift of drone. In future work, drone can combine visual data from cameras and inertial data from IMUs to reduce drift. The drift not only affects localization, also causes the offset of segments.

Bibliography

[1] B. Koffler, "Point cloud segmentation in steel structure," Technical Report DISAL-SP180, DISAL, Swiss Federal Institute of Technology, 2023.

[2] D. Jacquemont, "Point cloud segmentation of infrastructural steel elements," Technical Report DISAL-SP183, DISAL, Swiss Federal Institute of Technology, 2024.

[3] R. Junod, "Geometric reconstruction of a steel structure from onboard sensor data," Technical Report DISAL-SP188, DISAL, Swiss Federal Institute of Technology, 2024.

[4] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1686–1705, 2023.

[5] C. Cao, H. Zhu, H. Choset, and J. Zhang, "Tare: A hierarchical framework for efficiently exploring complex 3d environments.," in *Robotics: Science and Systems*, vol. 5, p. 2, 2021.

[6] Z. Meng, H. Qin, Z. Chen, X. Chen, H. Sun, F. Lin, and M. H. Ang, "A two-stage optimized next-view planning framework for 3-d unknown environment exploration, and structural reconstruction," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1680–1687, 2017.

[7] C. Papachristos, K. Alexis, L. R. G. Carrillo, and A. Tzes, "Distributed infrastructure inspection path planning for aerial robotics subject to time constraints," in *2016 international conference on unmanned aircraft systems (ICUAS)*, pp. 406–412, IEEE, 2016.

[8] C. Feng, H. Li, M. Zhang, X. Chen, B. Zhou, and S. Shen, "Fc-planner: A skeleton-guided planning framework for fast aerial coverage of complex 3d scenes," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8686–8692, IEEE, 2024.

[9] C. Cao, J. Zhang, M. Travers, and H. Choset, "Hierarchical coverage path planning in complex 3d environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3206–3212, IEEE, 2020.

[10] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding horizon" next-best-view" planner for 3d exploration," in *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 1462–1468, IEEE, 2016.

[11] A. K. Burusa, E. J. van Henten, and G. Kootstra, "Gradient-based local next-best-view planning for improved perception of targeted plant nodes," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15854–15860, IEEE, 2024.

[12] S. Song and S. Jo, "Surface-based exploration for autonomous 3d modeling," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4319–4326, IEEE, 2018.

[13] L. Wälti and A. Martinoli, "Lumped drag model identification and real-time external force detection for rotary-wing micro aerial vehicles," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3853–3860, IEEE, 2024.

[14] L. Geometric Tools, "Geometric tools for computer graphics," 2025.

Chapter 7 Appendix

7.1 Installing/configuring instructions

7.1.1 Third-parties library

The **Matplotlib-cpp** and **Geometric Tools Library** are already in the path */lib*.

7.1.2 Configuration

The parameters of package **path_planing** is in path */config/path_planning_config* with descriptions.

General

- exploration_rate: Sign of full coverage
- time_limitation: Exploration time limitation

Frontier evaluation

- fov(h,v,dist): Definition of FoV
- lambda_1: Weight of visibility gain
- lambda_2: Weight of density gain
- lambda_3: Weight of distance penalty
- lambda_4: Vertical weight for z-axis distance.
- radius: Viewpoint sampling radius
- sample_num: Number of sampled viewpoint
- collision_thre: Collision check for sampled viewpoint

Path planning

- model: local horizon model: 0 - sphere, 1 - horizontal band
- local_horizon_scale: The size of local horizon
- interpolate_size: The distance between every interpolated waypoint.
- minGainThreshold: Threshold for removing viewpoint with low information gain.
- position_thre: Position arrival check
- angle_thre: Orientation arrival check

RRTStar parameters

- goal_radius: Radius around the goal to consider it reached
- goal_sampling_prob: Probability of sampling the goal point
- jump_size: Max jump distance for RRTstar
- disk_size: Ball radius around which nearby points are found
- threshold_distance: Distance threshold for collision checking
- limit_x_low
- limit_x_high
- limit_y_low
- limit_y_high
- limit_z_low
- limit_z_high
- timeout: Timeout for RRTstar

Metrics

- drag_h: Coefficient of horizon motion consumption
- drag_v: Coefficient of upwards motion consumption
- hover: Coefficient of hover consumption
- online_plot: Plot output online via Matplotlib
- save_file: Save output files

7.1.3 Running

You can directly run the *all.launch* of ROS package **path_planning**.

```
roslaunch path_planning all.launch
```