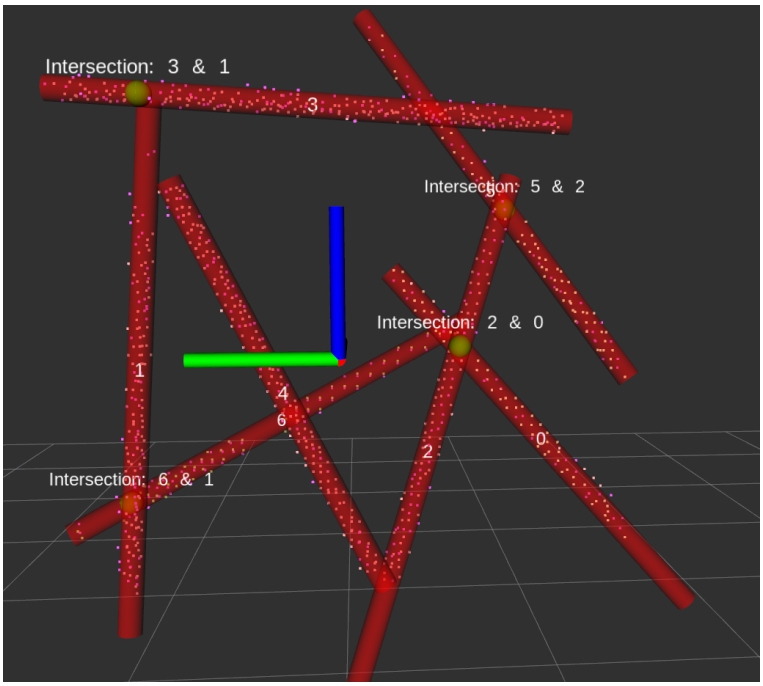
WS 2023-2024, Robotics, DISAL-SP183	Start: 19.09.2023 Finish: 05.01.2024

Point Cloud Segmentation of Infrastructural Steel Elements

Dimitri Jacquemont



Professor:	Alcherio Martinoli
Assistant:	Lucas Wälti

Contents

List of Figures

I am deeply grateful to Prof. Alcherio Martinoli and Lucas Wälti for their guidance and support in my **Point Cloud Segmentation of Infrastructural Steel Elements** project at the DISAL Lab.

General Notation

$\vec{a}_{\text{seg}}$: Starting points of `seg`

$\vec{b}_{\text{seg}}$: Direction vectors of `seg`

$\vec{proj}_{\text{seg}}(\vec{v})$: Projection of vector $\vec{v}$ onto `seg`

n_{seg} : Number of points used in `seg`

$\text{pca\_coeff}_{\text{seg}}$: PCA coefficient of `seg`

$\text{pca\_eig}_{\text{seg}}$: PCA eigenvalues of `seg`

diag_voxel : diagonal of the voxel used for voxel filtering

r_{seg}: Radius of `seg`

$\vec{n}$: Normal vector, perpendicular to both $\vec{b}_{\text{seg1}}$ and $\vec{b}_{\text{seg2}}$

$\vec{I}$: Intersection point

1 Introduction

This section discusses advanced techniques in steel infrastructure inspection, focusing on replacing conventional methods with Micro Aerial Vehicles and Time of Flight sensors. It covers the project's goal to create a real-time analysis system for identifying structural issues.

1.1 Enhancing Infrastructure Inspection

Ensuring the safety and integrity of steel structures like bridges and power lines is essential. Traditional inspection methods, being manual, are slow and carry risks. Utilizing Micro Aerial Vehicles (MAVs) equipped with advanced Time of Flight (ToF) cameras can revolutionize this process. MAVs enable safer and more efficient inspections, especially in hard-to-reach areas, reducing the hazards for human inspectors and offering an economical approach to maintenance.

1.2 Project Objectives

The primary objective of this project is to develop a sophisticated, real-time segmentation pipeline capable of processing and analyzing 3D point-cloud data gathered by MAVs. This pipeline aims at identifying signs of structural weaknesses in steel infrastructures accurately and efficiently. To achieve this, the project focuses on several key areas:

- Adaptation and Integration: Adapting and integrating the hough-3d-lines library [2], a tool for line detection in 3D point clouds, into ROS (Robotic Operating System) node. This integration is critical for processing the data collected by the ToF cameras on the MAVs.
- Real-Time Processing: Ensuring the pipeline operates in real time, which is essential for timely decision-making.
- Simulation and Testing: Utilizing the Webots simulator for testing different scenarios and environments, ensuring the robustness and reliability of the pipeline before real-world application.

2 Literature Review

This section examines point cloud segmentation algorithms, focusing on their efficacy in 3D geometry analysis of steel structures. It emphasizes the Iterative Hough Transform's role in line detection within 3D point clouds and discusses the Robot Operating System (ROS) for enhanced data processing and real-time analysis in MAV-based infrastructure inspections.

2.1 Previous Work in the Point Cloud Segmentation Project

In the previous semester project SP180 were studied a variety of point-cloud segmentation algorithms.

RANSAC (Random Sample Consensus)

Advantages: Powerful and very effective in finding planes in point clouds.

Disadvantages: Limited to separating planes and doesn't segment each metal beam; not suitable for iterative use with less powerful machines.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Advantages: Effective in separating points from noise and identifying large areas of interest.

Disadvantages: Time-consuming and unable to segment individual metal beams.

GMM (Gaussian Mixture Model)

Advantages: Effective in certain cases.

Disadvantages: Not very fast; requires knowledge of the number of elements to be segmented and struggles with complex arrangements.

Iterative Hough Transform

Advantages: Effective in complex arrangements and provides line equations useful for further analysis.

Disadvantages: Sensitive to the selection of parameters.

The main objectives include real-time analysis of 3D measurements and accurate representation of steel structures' geometry. Each method's advantages and limitations are discussed in the context of real-time processing

and accuracy. Different experiments are conducted using the studied segmentation methods.

The research project concludes that the Iterative Hough Transform method [1] shows significant potential for real-time analysis of steel structures, particularly in detecting and segmenting lines in complex arrangements. Based on the conducted experiments, it concludes that this method outperforms others in terms of accuracy and robustness.

2.2 Iterative Hough Transform

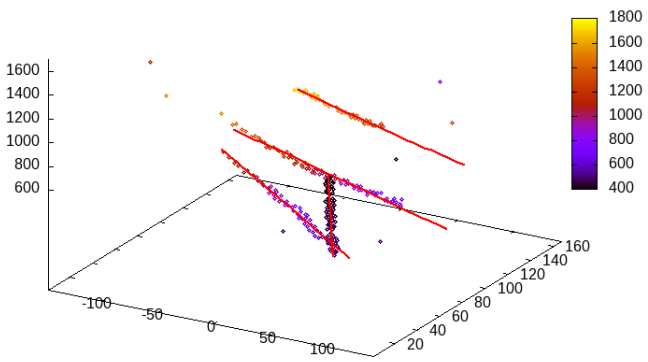


Figure 1: Iterative Hough Transform Demo [1]

The Iterative Hough Transform algorithm, as detailed in the paper **Iterative Hough Transform for Line Detection in 3D Point Clouds** [1], works as follows:

1. Parameter Space Discretization: The parameter space for all lines crossing the point cloud volume is discretized.

2. Hough Transform Application: The point cloud is transformed based on the discretization, creating a voting scheme in the parameter space.

3. Iterative Line Detection: The algorithm iteratively selects the line with the maximum votes, fits it using orthogonal least squares for accuracy, and then removes the points of this line from the point cloud.

4. Repetition of Steps: Steps 2 to 4 are repeated until the point cloud has too few points or the specified number of lines is found.

The algorithm aims to accurately detect lines in 3D point clouds by overcoming inaccuracies in parameter space discretization and voting, using an iterative approach and orthogonal least squares fitting.

2.3 Robot Operating System

The Robot Operating System [3] (ROS) is a cornerstone in current robotics software creation, serving as a dynamic framework for robot software development. It brings capabilities like hardware abstraction, control of low-level devices, standard functionalities, inter-process messaging, and package management. Its modular structure, along with tools for simulation, visualization, and debugging, have made it a favorite in the robotics field, both in academia and industry. For this project, ROS is vital for integrating various elements and enabling efficient data handling and communication, crucial for analyzing point-cloud data from MAVs in real time and enhancing MAVs' functionality for detailed tasks like inspecting steel structures.

3 Methodology

This section explains the steps of the point cloud segmentation algorithm. It begins with describing how beams are modeled in the algorithm and then covers the entire process, from getting point cloud data to visualizing the results in 3D. This includes key steps like processing the data, identifying lines, organizing segments, and combining them to understand their intersections and overall structure.

3.1 Segment Model

In the algorithm, the beam is modeled by an object `segment` containing the following information:

- Coefficients of the line best representing both position and orientation of the beam in 3D space ($\vec{a}_{\text{seg}}, \vec{b}_{\text{seg}}$).
- Parameters corresponding to the extremities of the beam $t_{\min}$, $t_{\max} \in \mathbb{R}$.
- Radius of the modeled beam r_{seg}, user defined.
- Number of points represented by this segment model n_{seg}.
- PCA Coefficient of the point distribution pca_coeff_{seg}.
- PCA eigenvalues of the point distribution pca_eig_{seg}.
- Points extracted from the point cloud generating this instance of segment points_{seg}.

The position and orientation of the detected beam are modeled by a line, and the parametrization is given by:

$$\vec{p} = t \cdot \vec{b}_{\text{seg}} + \vec{a}_{\text{seg}}, \quad t \in \mathbb{R}$$

The extremities, initial and end points of a beam, are defined by:

$$\vec{e1}_{\text{seg}} = \text{tmin}_{\text{seg}} \cdot \vec{b}_{\text{seg}} + \vec{a}_{\text{seg}}, \quad \vec{e2}_{\text{seg}} = \text{tmax}_{\text{seg}} \cdot \vec{b}_{\text{seg}} + \vec{a}_{\text{seg}}$$

3.2 Pipeline

This section outlines the sequential steps and methodologies employed in processing and analyzing point cloud data. It begins with Point Cloud Acquisition, capturing raw sensor data from a ToF sensor. This is followed by Pre-processing, where various filtering methods are applied to refine the data for better analysis. Next, the Iterative Hough Transform is utilized to detect lines in the point cloud, followed by Segments Sorting based on

specific criteria to ensure data quality. The pipeline then includes Segments Frame Conversion, transforming segment data from drone to world frame, and Segments Fusion, integrating new data with existing segments. Lastly, the pipeline examines Segments Intersections to identify any intersecting points, and concludes with Data Visualization, using tools like RViz to visually represent the processed data in 3D.

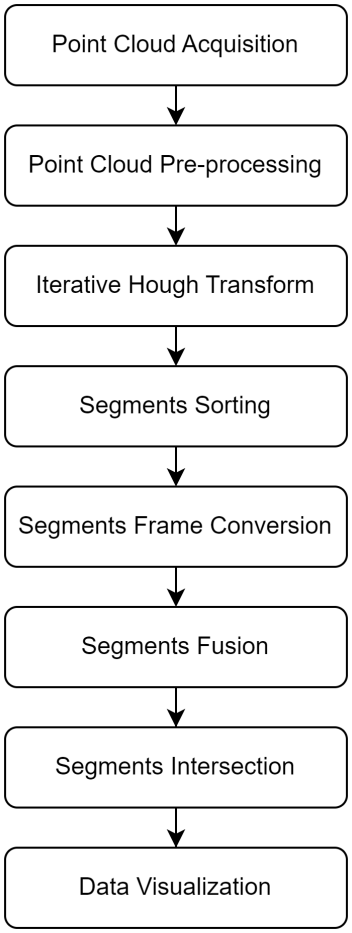


Figure 2: Algorithm Pipeline

3.2.1 Point Cloud Acquisition

The system acquires raw sensor data in the form of point clouds from a Time of Flight (ToF) sensor. To ensure accuracy, the time stamp of each point cloud is precisely synchronized with the drone's pose data, enabling a match between the drone's exact position and the point cloud data.

3.2.2 Point Cloud Pre-processing

To improve the Iterative Hough Transform's processing of point clouds, several filtering methods leveraging the capabilities of the Point Cloud Library (PCL) [4] were applied on the raw data :

- Filtering thresholds within a specified window using PCL filter module `PassThrough` to exclude points excessively distant. Only points with $x \in [0, 1.5]$, $y \in [-1.5, 1.5]$, $z \in [-1.5, 1.5]$ in meters, in the drone frame are kept.
- Voxel grid filter using PCL filter module `VoxelGrid` to ensure uniform density throughout the point cloud, and reducing the number of points requiring computation. The voxel leaf size is calculated from the expected beam radius and the ratio of radius to leaf size chosen by the user.

The formula for `leaf_size` is given as:

$$\text{leaf\_size} = \frac{r_{\text{seg}}}{\text{rad\_2\_leaf\_ratio}}$$

with variable rad_2_leaf_ratio corresponds to the user defined ratio between the segment radius and voxel leaf size.

3.2.3 Iterative Hough Tranform

The 3D Hough Transform [2] is an algorithm designed to identify lines in three-dimensional point clouds. It functions by converting points from 3D space into a parameter space. In this parameter space, points that align along a line intersect at a specific location, signaling the presence of a line. The process is iterative; in each cycle, the most prominent line is detected and then removed from consideration, enabling the detection of multiple lines sequentially. The points participating in the detection of each segments are stored for later analysis and visualization.

Parameters in the 3D Hough Transform include :

`opt_dx`, a discretization setting affecting accuracy and computational load.

`minimum_votes`, a minimum threshold on the number of points to confirm line detection.

`opt_nlines`, defining the maximum lines that can be detected.

`granularity`, detailing direction discretization.

3.2.4 Segments Sorting

The points used for the detection of each segment `seg` then undergo a Principal Component Analysis (PCA). A PCA coefficient and the eigenvalue vector are calculated. The coefficient, which measures how much the points in segment `seg` are elongated, is calculated as follows:

$$\text{pca\_coeff}_{\text{seg}} = \frac{\text{pca\_eig}_{\text{seg}}[0]}{\text{pca\_eig}_{\text{seg}}[0] + \text{pca\_eig}_{\text{seg}}[1] + \text{pca\_eig}_{\text{seg}}[2]} \in [0, 1]$$

with $\text{pca\_eig}_{\text{seg}}$ corresponding to the eigenvectors of the points of the segments.

Post Iterative Hough Transform, segments are retained based on multiple criteria:

Number of Points: The segment `seg` has to contain enough points to be considered a valid segment. This is estimated from both the extracted segment's characteristic and user set parameters.

The radius of the extracted beam is found and the initial point and endpoint of the segment are computed:

$$\vec{e1}_{seg} = \text{tmin}_{seg} \cdot \vec{b}_{seg} + \vec{a}_{seg}, \quad \vec{e2}_{seg} = \text{tmax}_{seg} \cdot \vec{b}_{seg} + \vec{a}_{seg}$$

The minimum number of points is then deduced:

$$\text{min\_nb\_points} = 2 \cdot \frac{r_{\text{seg}} \cdot \|\vec{e2}_{seg} - \vec{e1}_{seg}\|}{\text{rad\_2\_leaf\_ratio} \cdot 2 \cdot \text{diag\_voxel} \cdot 2 \cdot \text{diag\_voxel}}$$

with variable diag_voxel defined as $\sqrt{3}\cdot$leaf_size, this value corresponds to the maximum possible distance between two points in a voxel of side leaf_size.

Computed Radius Matching Predefined Radius: The radius of the detected beam is measured and compared with a user-set expected radius. This comparison is done by checking how far the furthest point of the segment is from the beam's central axis, which is defined by $\vec{a}_{seg}$ and $\vec{b}_{seg}$. The segment is considered valid if the difference between this measured radius and the expected radius is less than diag_voxel.

Segment Point Distrinution & Density: A maximum distance of $2 \cdot$ diag_voxel between 2 point of the same segment is allowed to ensure the segment's integrity and consistency in its point distribution.

PCA Coefficient Value: The PCA coefficient value must be above a user-defined minimum threshold.

3.2.5 Segments Frame Conversion

All retained segment are then transformed from drone frame to world frame using a rotation matrix derived from the orientation of the drone with respect to the world frame, and the drone's position with respect to the world frame. The rotation matrix is computed from the drone orientation's quaternion.

Let $q = w + xi + yj + zk$, where w, x, y and z are the components of the quaternion. The corresponding rotation matrix R for this quaternion can be computed as follows:

$$R = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2zw & 2xz + 2yw \\ 2xy + 2zw & 1 - 2x^2 - 2z^2 & 2yz - 2xw \\ 2xz - 2yw & 2yz + 2xw & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

From this rotation matrix R and drone position $\vec{P}_{world}$ in the world frame, segments in the drone frame with coefficients $\vec{a}_{\text{drone}}$, $\vec{b}_{\text{drone}}$, can be transformed to the world frame.

$$\vec{a}_{\text{world}} = R \cdot \vec{a}_{\text{drone}} + \vec{P}_{world}$$

$$\vec{b}_{\text{world}} = R \cdot \vec{b}_{\text{drone}}$$

After transferring all segments to the new frame, each new segment is inspected to ensure that both ends are higher than a user-defined height above the floor.

3.2.6 Segments Fusion

To integrate newly computed segments from the drone with pre-existing segments, criteria are established to determine whether a segment was previously stored or is new.

Similarity Detection Solution 1 - Middle Position & Segment Parameters Matching

This primary method entails comparing the line coefficients of two segments-specifically $\vec{a}_1$, $\vec{b}_1$ for segment 1, and $\vec{a}_2$, $\vec{b}_2$ for segment 2. Segments are considered similar if their starting points, $\vec{a}_1$ and $\vec{a}_2$, as well as their direction vectors, $\vec{b}_1$ and $\vec{b}_2$, are within a certain proximity to each other. This proximity is defined by the conditions:

$$\|\vec{a}_2 - \vec{a}_1\| \leqslant T_a \quad \text{and} \quad \|\vec{b}_2 - \vec{b}_1\| \leqslant T_b$$

where T_a and T_b are predetermined static thresholds. Lower values for these thresholds mean a stricter similarity criterion, while higher values permit a broader range of differences for segments to be considered similar.

To further verify the actual similarity of segments and not just their alignment, the middle points of both segments are compared:

$$\vec{p}_{\text{seg}} = \left(\frac{\text{tmax}_{\text{seg}} - \text{tmin}_{\text{seg}}}{2} + \text{tmin}_{\text{seg}} \right) \cdot \vec{b}_{\text{seg}} + \vec{a}_{\text{seg}}$$

$$\|\vec{p}_2 - \vec{p}_1\| \leqslant T_p$$

Here, T_p denotes another predefined static threshold.

Similarity Detection Solution 2 - Middle Position & Direction Check

This alternate method assesses both the direction and relative position of two segments. To evaluate the directional similarity, the dot product of their direction vectors $\vec{b}_{\text{seg}}$ is used:

$$\vec{b}_1 \cdot \vec{b}_2 \geqslant T_d$$

where T_d is a predefined threshold, with values closer to 1 indicating a higher degree of similarity in direction.

In addition to direction, the middle points of both segments are compared to assess positional similarity:

$$\vec{p}_{\text{seg}} = \left(\frac{\text{tmax}_{\text{seg}} - \text{tmin}_{\text{seg}}}{2} + \text{tmin}_{\text{seg}} \right) \cdot \vec{b}_{\text{seg}} + \vec{a}_{\text{seg}}$$

$$\|\vec{p}_2 - \vec{p}_1\| \leqslant T_p$$

Here, T_p represents a predefined static threshold for assessing the proximity of the segment's central points.

Similarity Detection Solution 3 - Segment Projection

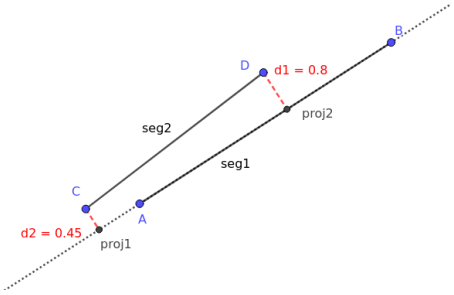


Figure 3: Segment Projection

This third solution is based on evaluating the distances between the endpoints of segment 2 and their projections on segment 1. These distances help indicate the relative proximity of segment 1 to segment 2. If the distances are below a dynamic threshold ϵ, and the radii of segment 1 and segment 2 are identical.

$$\vec{proj}_2(\vec{e1}_1) \leqslant \epsilon \quad \text{and} \quad \vec{proj}_2(\vec{e2}_1) \leqslant \epsilon$$

The dynamic threshold ϵ is defined as:

$$\epsilon = 2 \cdot \text{diag\_voxel} + r_1 + r_2$$

with r_{seg} corresponding to the radius of segment `seg`.

Fusion Solution 1 - Mean Segment

In an initial approach, the new segment is computed as the average of two fused segments.

For the starting point $\vec{a}_{\text{update}}$:

$$\vec{a}_{\text{update}} = \vec{proj}_1(\vec{e1}_2) + \frac{1}{2} \cdot (\vec{e1}_2 - \vec{proj}_1(\vec{e1}_2))$$

This equation first projects the starting point of segment 2 ($\vec{e1}_2$) onto segment 1, then averages it with the original starting point of segment 2.

For the direction $\vec{b}_{\text{update}}$:

$$\vec{b}_{\text{update}} = (\vec{proj}_1(\vec{e2}_2) - \vec{proj}_1(\vec{e1}_2)) + \frac{1}{2} \cdot ((\vec{e2}_2 - \vec{proj}_1(\vec{e2}_2)) - (\vec{e1}_2 - \vec{proj}_1(\vec{e1}_2)))$$

This equation calculates the direction based on the mean of the projections of segment 2's endpoints on segment 1.

If segment 1 and segment 2 partially or completely overlap in this new segment, the stored segment is replaced with the new fused segment, which incorporates information from both.

Fusion Solution 2 - Weighted Fusion

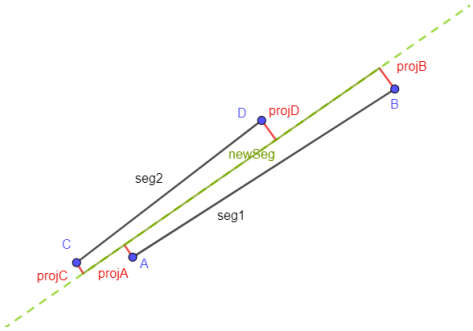


Figure 4: Segment Weighted Fusion

A fusion coefficient, f, is calculated to determine the proportion of information from the new segment to be retained. It blends the pca coefficient `pca_coeff` of the two segments, weighted by the number of points in each (given by w), giving priority to the segment with more points:

$$w = \max\left(\frac{n_2}{n_1 + n_2}, w_{\min}\right) \quad f = \frac{\text{pca\_coeff}_2 \cdot w}{\text{pca\_coeff}_1 \cdot (1 - w) + \text{pca\_coeff}_2 \cdot w}$$

In the fusion process of two segments, segment 1 and segment 2, the updated segment's starting point and direction are calculated as follows:

For the starting point $\vec{a}_{\text{update}}$:

$$\vec{a}_{\text{update}} = \vec{proj}_1(\vec{e1}_2) + f \cdot (\vec{e1}_2 - \vec{proj}_1(\vec{e1}_2))$$

This equation projects the starting point of segment 2 ($\vec{e1}_2$) onto segment 1, and then adjusts it based on the fusion coefficient f, which balances the information from both segments.

For the direction $\vec{b}_{\text{update}}$:

$$\vec{b}_{\text{update}} = (\vec{proj}_1(\vec{e2}_2) - \vec{proj}_1(\vec{e1}_2)) + f \cdot ((\vec{e2}_2 - \vec{proj}_1(\vec{e2}_2)) - (\vec{e1}_2 - \vec{proj}_1(\vec{e1}_2)))$$

This equation first calculates the relative direction between the projected end points of segment 2 on segment 1, and then adjusts this direction by the fusion coefficient f, blending the orientations of the two original segments.

The projection operation $\vec{proj}_{\text{seg}}(\vec{v})$ is defined as the projection of a point $\vec{v}$ onto a line segment seg. Given that seg is defined by its anchor point $\vec{a}_{\text{seg}}$ and direction vector $\vec{b}_{\text{seg}}$, the projection is calculated as follows:

$$\vec{proj}_{\text{seg}}(\vec{v}) = \vec{a}_{\text{seg}} + \frac{(\vec{v} - \vec{a}_{\text{seg}}) \cdot \vec{b}_{\text{seg}}}{\vec{b}_{\text{seg}} \cdot \vec{b}_{\text{seg}}} \cdot \vec{b}_{\text{seg}}$$

This formula finds the point on the line segment seg closest to $\vec{v}$ in the Euclidean space. It computes this by determining the point along the direction of $\vec{b}_{\text{seg}}$ from $\vec{a}_{\text{seg}}$ that minimizes the distance to $\vec{v}$.

If segment 1 and segment 2 partially or completely overlap in this new segment, the stored segment is replaced with the new fused segment, which incorporates information from both.

When two similar segments are fused, new `pca_coeff` and `pca_eig` are computed using w to reflect the point count from each segment:

$$\text{pca\_coeff}_{\text{update}} = \text{pca\_coeff}_1 \cdot (1 - w) + \text{pca\_coeff}_2 \cdot w$$

$$\text{pca\_eig}_{\text{update}} = \text{pca\_eig}_1 \cdot (1 - w) + \text{pca\_eig}_2 \cdot w$$

Algorithm 1 Segment Similarity Detection Solution 3 & Fusion Solution 2

Require: Newly computed segments seg_{new} from the drone, pre-existing segments seg_{old}
  for all pairs of segments seg_{old}, seg_{new} **do**
    Calculate distances between endpoints of seg_{new} and their projections on seg_{old}
    Define dynamic threshold $\epsilon = 2 \cdot \text{diag\_voxel} + r_1 + r_2$
    if distances $< \epsilon$ **and** radii of seg_{old}, seg_{new} are identical **then**
      Calculate fusion coefficient f
      Fuse segments to create new segment $\vec{a}_{\text{update}}$, $\vec{b}_{\text{update}}$
      if seg_{old} and seg_{new} overlap **then**
        Replace stored segment with fused segment
      else
        Add seg_{new} to seg_{old}
      end if
    else
      Add seg_{new} to seg_{old}
    end if
  end for

3.2.7 Segments Intersections

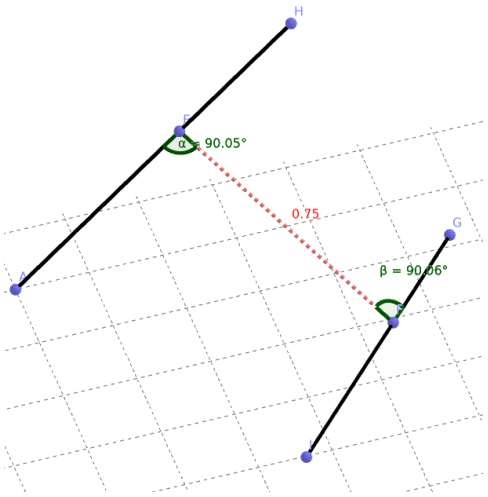


Figure 5: Segment Intersection

Once segments are fused and the segment list is updated, intersections between segments are checked. The process involves finding the shortest distance between two segments to determine if they intersect.

The cross product of the direction vectors of the two segments is computed:

$$\vec{n} = \vec{b}_1 \times \vec{b}_2$$

If the norm of this cross product is nearly zero ($\|\vec{n}\| \approx 0$), it indicates that the segments are parallel, and no intersection is possible.

The cross product is then normalized:

$$\vec{n}_{\text{norm}} = \frac{\vec{n}}{\|\vec{n}\|}$$

To find the intersection, the initial points of both segments are computed:

$$e\vec{1}_1 = \text{tmin}_1 \cdot \vec{b}_1 + \vec{a}_1, \quad e\vec{1}_2 = \text{tmin}_2 \cdot \vec{b}_2 + \vec{a}_2$$

Then, the linear system is solved:

$$\begin{bmatrix} \vec{b}_1 & -\vec{b}_2 & \vec{n}_{\text{norm}} \end{bmatrix} \times \begin{bmatrix} t_1 \\ t_2 \\ d \end{bmatrix} = \left(e\vec{1}_2 - e\vec{1}_1 \right)$$

where $\times$ is a cross product.

Here, t_1 and t_2 are scalar values that locate the potential intersection point along $\vec{b}_1$ and $\vec{b}_2$, respectively, while d represents the distance along $\vec{n}_{\text{norm}}$ from the line defined by $\vec{b}_1$ and $\vec{b}_2$.

If the solution (t_1, t_2, d) exists and d is within the predefined threshold ϵ, then the intersection point $\vec{I}$ can be calculated:

$$\vec{I} = t_1 \cdot \vec{b}_1 + \vec{a}_1 \quad \text{or equivalently,} \quad \vec{I} = t_2 \cdot \vec{b}_2 + \vec{a}_2$$

This intersection point is valid if it lies within the bounds of both segments.

Algorithm 2 Checking Segment Intersections

Require: Updated list of fused segments
  for all pairs of segments segment 1, segment 2 **do**
    Compute cross product $\vec{n} = \vec{b}_1 \times \vec{b}_2$
    if $\|\vec{n}\| \approx 0$ **then**
      Segments are parallel, no intersection
    else
      Normalize $\vec{n}$ to get $\vec{n}_{\text{norm}}$
      Define initial points of both segments
      Solve linear system to find potential intersection
      if Solution exists and distance d within threshold ϵ **then**
        Calculate intersection point $\vec{I}$
        if $\vec{I}$ within bounds of both segments **then**
          Intersection confirmed
        end if
      end if
    end if
  end for

3.2.8 Data Visualisation

Use RViz or similar tools to visualize the segments and intersections in 3D.

4 Experimental setup

This section introduces the practical application and implementation of the point cloud segmentation algorithm within the ROS framework. It details the development of specific ROS nodes created for this project. The section includes descriptions of these nodes' functionalities, their interaction with other quadcopter components, and their integration into the overall system, as illustrated in the provided node graph Figure 6. Additionally, it discusses the adaptation of the Hough-3D-Lines library to fit within the ROS Node and the modifications made to optimize its performance in this context.

4.1 ROS Implementation

The pipeline described in section 3 is implemented in ROS Noetic to interact with other quadropter components such as the auto pilot. ROS nodes `pointcloud_seg`, and `pointcloud_tfbr` were developed in the context of this project.

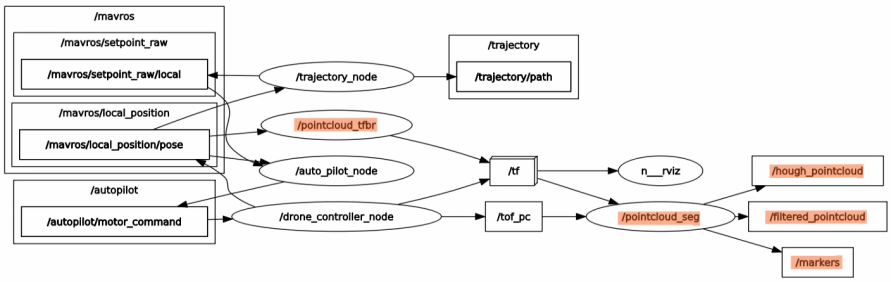


Figure 6: Global Node Graph

The node graph Figure 6 displays all the nodes interacting, both from the `Auto Pilot`, and the `Pointcloud Segmentation` package. The contributed elements are highlighted in orange.

ROS Node `pointcloud_tfbr`

Node `pointcloud_tfbr` is in charge of broadcasting a transformation (TF) from a motion capture frame (mocap) to a world frame (Figure 7), allowing for the visualization of a point-cloud from a drone's perspective in the world frame. This is achieved by subscribing to pose data and broadcasting the corresponding TF transform.

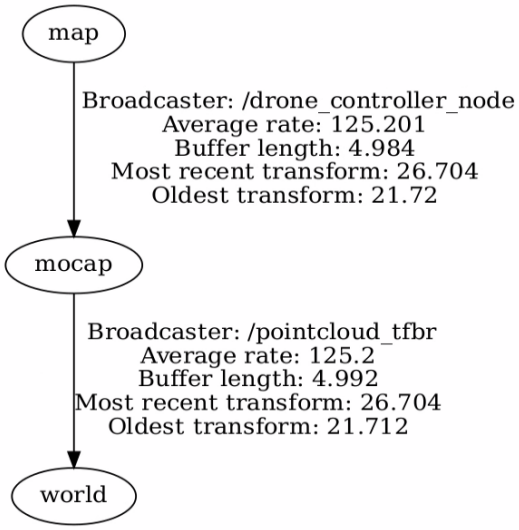


Figure 7: TF2 Frames

For more details, see the implementation in file pointcloud_tfbr.cpp.

ROS Node `pointcloud_seg`

Node `pointcloud_seg` is in charge of the whole point cloud segmentation pipeline. This node is subscribed to topic `tof_pc` (Time of Flight Point Cloud) and launches a callback each time a new point cloud is published. For vizualisation purpose, the node publishes three topics `markers` (cylinder representation and intersections), `filtered_pointcloud` (point cloud filtered) and `hough_pointcloud` (points belonging to the extracted segments). This node is also listening and storing the drone's pose transform in a buffer `tfBuffer` for future segment transformation from drone frame to world frame.

The class `PtCdProcessing` is in charge of the whole segmentation pipeline, and embeds all the functions and variable related to the pipeline into a unique object.

On the node's start, the node user-defined parameters are read from the YAML file `config.yaml` and assigned to their respective variables.

User-defined parameters:

`vebose_level`: 0 (NONE), 1 (INFO), 2 (WARN)

`path_to_output`: Path to the processed output data (segments and intersections).

`floor_trim_height`: Cut off the point cloud below this height in the world frame.

`min_pca_coef`: Minimum PCA coeff to be considered as a line.

`min_weight`: Minimum weight coefficient for the line fusion.

`rad_2_leaf_ratio`: Ratio of radius to leaf size to determine leaf size.

`opt_minvotes`: Minimum number of votes to be considered a line (Iterative Hough Transform).

`granularity`: Granularity of the search between 0 and 6 (Iterative Hough Transform).

`opt_nlines`: Number of lines to be detected (Iterative Hough Transform).

`radius_sizes`: Beam radius sizes (currently only implemented for one size).

Once the node is fully initialized, a `tof_pc` callback can occur. When new point cloud data is received in the `tof_pc` callback function, it is stored in a shared data structure and a signal is sent to notify that new data is available. Meanwhile, a separate thread initialized in the class constructor is running in the background, waiting for this signal. Once notified, this thread processes the new data independently, allowing for optimised and simultaneous execution of data reception and processing. This multithreading approach ensures that the system can handle incoming data in real-time without blocking other operations.

When the signal indicating readiness for data processing is received, the processing of the newly received point cloud begins. The drone pose and orientation, closest in timestamp to the `tof_pc` message, are stored for later use in transforming the segments from the world frame to the drone frame. The point cloud is then filtered and passed to the function responsible for the Iterative Hough Transform.

The retrieved segments are transformed from the drone frame to the world frame using the previously found drone pose. These segments in the world frame are then merged with existing segments or added as new ones, and new intersections are computed on the updated and new segments. The segments in the world frame, along with the intersections, are published for visualization using RViz.

Upon shutdown of the node, the current lists of segments and intersections, along with the processing time data, are saved into three separate files: `segments.csv`, `intersections.csv`, and `processing_time.csv`. These files are stored at the location specified by the `path_to_output` value.

For more details, see the implementation in file pointcloud_segmentation_node.cpp.

4.2 Hough-3D-Lines Library Adaptation

Main file

To integrate the Hough-3D-Lines project within a ROS Node, the main file of the project was modified to include a custom-made, callable function.

This adaptation involved revising the `hough3dlines.cpp` file from the `hough-3d-lines` library, enabling the Hough algorithm's utilization within a ROS node through a callable function. The modified Iterative Hough Transform function takes as parameter the ToF point cloud currently being processed.

The function begins by deleting points with NaNs and Inf values from the point cloud. Tthe Iterative Hough Transform then processes the point cloud, and the newly derived lines are further analyzed. The end points of the segments are determined based on the points at the extremities of the segment's point cloud. Additionally, a Principal Component Analysis (PCA) is performed on the point cloud to assess its distribution.

The segments are then filtered based on several criteria further explained in section 3.2.4:

- The segment must contain a sufficient number of points to be considered valid.
- The computed radius of the segment should align with the predefined radius, indicative of the beam's identity.
- The distance between adjacent points in a segment must not exceed a maximum allowable limit, ensuring the segment's structural integrity and consistency in its point distribution.
- The PCA coefficient value must be above a minimum user-defined threshold.

For more details, see the implementation in file hough_3d_lines.h.

Hough Library Modification

In this project, the Hough transform is executed repeatedly with identical parameters. To optimize performance, certain variables that were initially allocated every time the Hough transform started can be allocated just once.

The modifications are centered around optimizing the use of the Sphere object within the Hough transform implementation (hough.cpp, hough.h). By initializing the Sphere object globally and sharing it across instances of the Hough class, the modified code reduces computational redundancy. This change improves the performance, especially when multiple instances of the Hough transform are used.

1. **Initialization of Hough Space**: The most important addition is the `initHoughSpace` function. This function globally initializes the `Sphere` object used by the Hough transform. This change allows the direction vectors to be only computed once and shared, rather than being recomputed for each instance of the Hough class.
2. **Hough Constructor and Destructor**: The constructor of the Hough class has been modified to use the globally initialized `Sphere` object. This change reduces redundancy and improves the performance of the algorithm by avoiding repeated calculations of the same data. The destructor no longer deletes the `Sphere` object, as it is now managed globally.

5 Testing and Results

This section delves into the evaluation of the point cloud segmentation algorithm using a simulation setup in Webots. The testing environment including different simulation scenarios to thoroughly assess the algorithm's performance is described. The section then proceeds to detail the results, presenting both quantitative and qualitative analyses. Quantitative results include a comparison of extracted segments with ground truth data in various testing environments, while qualitative results assess the overall effectiveness of the algorithm in more general, real-world-like simulations. This section aims to validate the algorithm's effectiveness and identify areas for improvement based on the simulation outcomes.

5.1 Simulation Setup in Webots

The algorithm was tested on a Webots simulation of the drone. Three main simulation environments were used.

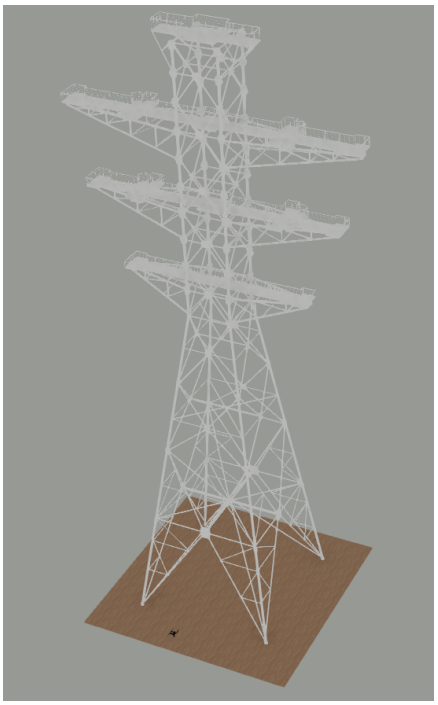


Figure 8: Simulation Environment flying_arena_ros_tower

In the `flying_arena_ros_tower.wbt` simulation environment (Figure 8), waypoints defined in `wp_tower.csv` are strategically positioned close to the structure. They are designed to explore all corners of the structure, and to

maximize the information gathered.

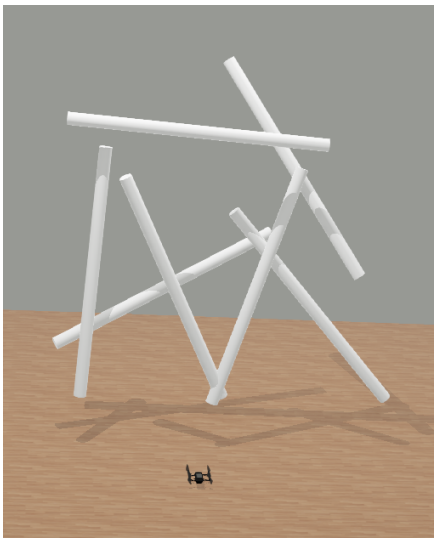


Figure 9: Simulation Environment flying_arena_ros_obs_tests

The `flying_arena_ros_obs_tests.wbt` simulation environment (Figure 9) is dedicated to experimental uses, such as testing and result quantification. The associated waypoints for these activities are contained in `wp_tests.csv`.

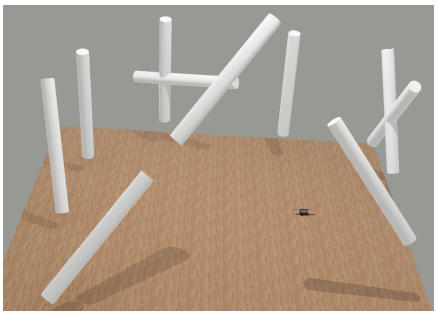
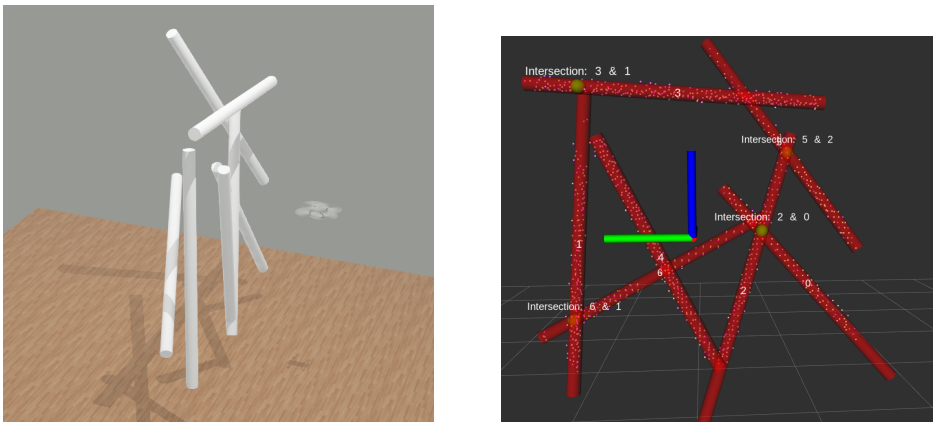


Figure 10: Simulation Environment flying_arena_ros_obs

Lastly, the `flying_arena_ros_obs.wbt` simulation environment (Figure 10) was utilized during the early stages of algorithm development and debugging. Here, the drone follows a figure-eight trajectory to facilitate these processes.

5.2 Results

Performance Quantification



(a) Testing Environment (b) Segments Output

Figure 11: Environment and Algorithm Output

Quantitative testing was conducted in the `flying_arena_ros_obs_tests.wbt` environment, as shown in Figure 11a. This setup allowed for a comparison between the segments present in the testing environment and those extracted by the segmentation node, depicted in Figure 11b.

This environment is composed of seven beams, each having a radius of $0.05m$. To test the algorithm's capability to detect intersections, several beams in the simulation are intentionally positioned to cross each other. A specific set of parameters were chosen for this test:

`floor_trim_height`: 0.3 m

`min_pca_coef`: 0.99

`min_weight`: 0.01

`rad_2_leaf_ratio`: 1.5

`opt_minvotes`: 12

`granularity`: 6

`opt_nlines`: 10

`radius_sizes`: 0.05 m

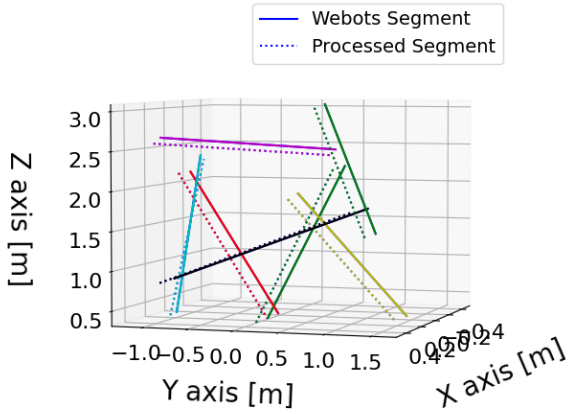


Figure 12: Segment Comparison

Figure 12 presents a comparative analysis of both the ground truth segments (labeled as 'Webots Segment') and the extracted segments (labeled as 'Processed segments'). The segments were obtained after multiple measurements. The color-coding of the segments helps visualize the correspondence between the extracted segments and their respective ground truth counterparts. Although a slight offset is noticeable between the segments (further discussed in section 6.3), the extracted segments are accurately located and provide a reasonable representation of the ground truth environment in the simulation.

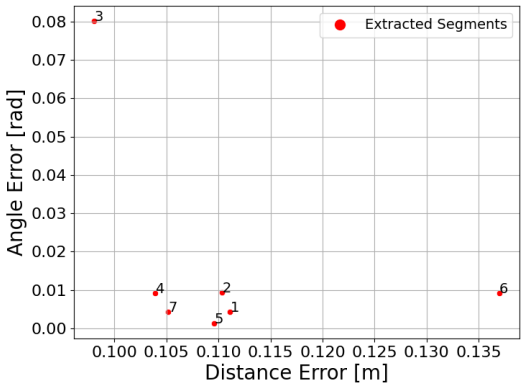
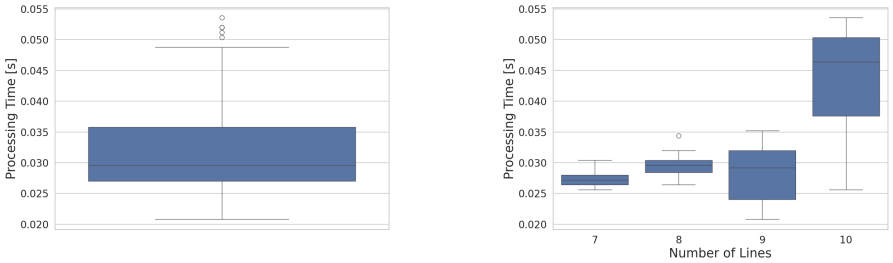


Figure 13: Distance VS Angle Error

Figure 13 presents more quantitative results, comparing the middle posi-

tions and relative angles of the ground truth segments with those extracted by the algorithm. The distance error varies and is attributed to the offset of the points relative to the center of the beam. This issue will be further addressed in Section 6. The angle error is generally minimal, except for Segment 3, which appears as an outlier. This deviation could be a result of sudden drone movements, such as during takeoff, affecting point processing. Overall, the estimation closely aligns with reality, although there is room for improvement.



(a) Overall Processing Time

(b) Processing Time Distribution by Number of Lines

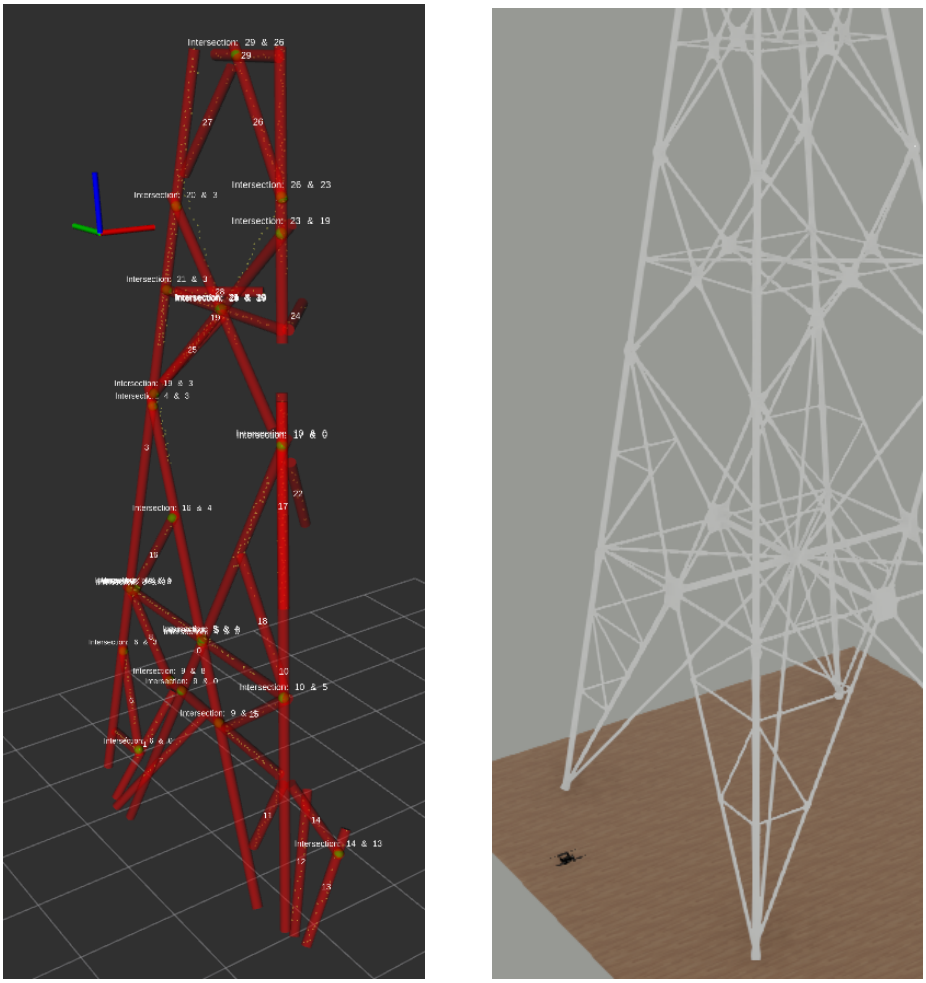
Figure 14: Time Related Measurements

Figure 14a and 14b illustrates the distribution of processing times for each point cloud evaluated in this test, providing a statistical overview of the algorithm's performance across different executions and number of lines extracted.

General Results

In a more general experiment in the `flying_arena_ros_tower.wbt` simulation environment, tests on a model structure were carried out to assess the overall effectiveness of the system. The outcomes were generally positive, showing that the drone was capable of detecting and mapping the main structure.

The algorithm still encountered some challenges. It occasionally had segments failing fusion, and thus superposition of two similar segments (further discussed in section 6.3). Additionally, the system sometimes struggled with variations in the structure, such as changes in its configuration or when beams of different radii were present in the structure.



(a) Extracted Structure (b) Simulation Structure

Figure 15: Structure Extracted VS Ground Truth

6 Discussion

This section evaluates the algorithm's complexity and addresses technical challenges, such as segment location accuracy, similarity detection, and fusion methods. It also considers the algorithm's ability to handle various beam radii and the integrity of segment point distributions. Key limitations like handling flat-angle intersections, segment fusion issues, beam diversity, and detection of multiple radius segments are discussed. Future enhancements include integrating GTSAM for structural model optimization and conducting real-world tests to assess the algorithm's robustness in diverse environments.

6.1 Algorithm Complexity

Upon analyzing the algorithm's main components, the complexities for each point cloud callback were deduced as follows:

- Point cloud pre-processing involves iterating through the cloud, leading to a $\mathcal{O}(n)$ complexity, where n is the number of points
- The Iterative Hough Transform complexity is $\mathcal{O}(2nN_1 + 3n^2n_{min})$, with N_1 as the number of directions, n the point count, and n_{min} the minimum votes for a line [1]
- PCA, applied to all valid segment, has for a reduced point cloud of size $n_{red} \geq n_{min}$, and number of newly extracted segment s_{ext} a complexity of $\mathcal{O}(s_{ext} \cdot \min\left(n_{red}^3, 3^3\right))$ [5]
- The segment processing, a quadratic operation, has a complexity of $\mathcal{O}(s_{all}^2)$, where s_{all} is the overall segment count

The algorithm's runtime is primarily influenced by the Iterative Hough Transform and PCA, with the overall complexity being either $\mathcal{O}(2nN_1 + 3n^2n_{min})$ or $\mathcal{O}(s_{ext} \cdot \min\left(n_{red}^3, 3^3\right))$, depending on N_1, n, n_{red}, s_{ext} and n_{min} values.

6.2 Technical Challenges & Solutions Implemented

Segment's Location

Significant point location variability was observed, mainly due to changes in the drone's position. This issue occurred because the drone's position while processing detected segments into the world frame was not consistent with its position at the time of detection. To improve accuracy, the point cloud's timestamp is now carefully matched with the drone's position timestamp. This approach enhances the precision of segment position detection

and reduces variability.

For more details, see the implementation in function **closestDronePose** in file pointcloud_segmentation_node.cpp.

Segment's Similarity Detection

The challenge in detecting segment similarity was multifaceted, focusing not just on segment positions but also on aligning their directions and placements. Various methods were explored, including "Middle Position & Segment Parameters Matching," "Middle Position & Direction Check," and "Segment Projection" (elaborated in 3.2.6). The most effective method proved to be "Segment Projection," which involved projecting a new segment's endpoints onto an existing world segment and verifying if the projection length was within a dynamic threshold related to the beam's radius.

The "Middle Position & Segment Parameters Matching" method was found unreliable due to significant variations in line coefficients $\vec{a}_{\text{seg}}$ and $\vec{b}_{\text{seg}}$, even among closely positioned segments.

Similarly, the "Middle Position & Direction Check" method was ineffective. It relied on comparing segment midpoints, which varied greatly with segment length, leading to inaccuracies, especially for similar segments with different lengths.

For a detailed understanding of the chosen approach, refer to the function **checkSimilarity** in the file pointcloud_segmentation_node.cpp.

Segment's Fusion

When two segments are identified as similar, their information is merged. Several strategies were explored (elaborated in 3.2.6):

- Mean Segment

- Weighted Fusion

Initially, the mean of the two segments was considered, assigning equal weight to both the new and old segments. This solution worked, however this method was improved to take into account the quality of segments. The segments were fused considering their point distribution and the number of points. In this approach, a threshold was set to ensure information fusion between segments, even when one segment had significantly more points than the other.

For more details, see the implementation in function **checkSimilarity** in file pointcloud_segmentation_node.cpp.

Generalization to all Radius Sizes

To accommodate various radius sizes in the algorithm, the desired radius size must be input before execution. This facilitates the adaptation of certain parameters, like `leaf_size` in the voxel grid filter, based on the size of the beam. The ratio of the radius to the leaf size, denoted as `rad_2_leaf_ratio`, is a critical user-defined parameter. Parameter `diag_voxel` is also used for various threshold including in the algorithm for beam similarity detection.

`opt_dx` corresponding to the step width in the x'y'-plane for the Hough Transform influences the accuracy of beam detection due to its role in Hough space resolution. The value of `opt_dx` is linked to the leaf size, which is derived based on the radius specified by the user.

The selected values for `opt_dx` and `diag_voxel` were determined through experimental testing, and user defined radius. These are not optimal values for every case, and adjustment might be required for extreme cases.

Segment's Integrity Check

The integrity and consistency of a segment is verified through the analysis of its point distribution. This involves ensuring that the distance between neighboring points within a segment does not exceed a predefined threshold. If two adjacent points are separated by a distance greater than this threshold, the segment is deemed non-integral. The threshold is set to twice the leaf size multiplied by the square root of three. This value corresponds to the maximum distance between two points in diagonally adjacent voxels.

For more details, see the implementation in function **hough3dlines** in file hough_3d_lines.h.

PCA Filtering

To make sure that the detected lines are actual segments and not floor or random point clouds, a Principal Component Analysis (PCA) is executed on the point cloud of the given segment. This analysis is crucial for assessing the segment's overall shape. A segment is considered valid if its PCA coefficient, denoted as $\text{pca\_coeff}_{\text{seg}}$, exceeds a predefined value specified by the user parameter `min_pca_coeff`.

For more details, see the implementation in function **hough3dlines** in file hough_3d_lines.h.

6.3 Limitations of the Current System

Intersections with Flat Angles

The algorithm encounters difficulties in distinguishing two segments intersecting at a shallow angle as separate entities. This limitation is demonstrated in Figure 16, where segment 5 is inaccurately identified as a single segment, although it comprises two distinct segments, one before and another after the intersection. The challenge stems from the algorithm's similarity criterion, which merges two similar segments into one, failing to recognize minor angular variations between intersecting segments.

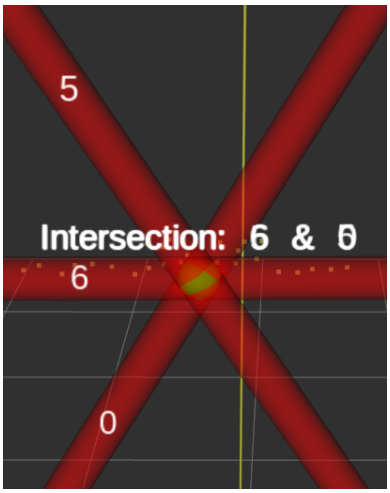


Figure 16: Intersection Example

Intersection Fusion

The algorithm currently lacks the capability to identify intersections involving more than two segments. As depicted in Figure 16, this limitation results in unclear intersection naming due to name superposition, primarily because there are at least three intersecting segments at that point.

Beam Diversity

The algorithm assumes cylindrical beams and is not designed for complex beam shapes, presenting challenges in detecting non-cylindrical beams. However, it might work for beams with small enough radius. Addressing this could involve a multi-stage filtering process, starting with basic line segment identification and refining into complex shapes. Developing custom features to recognize specific geometrical attributes of different beam types, like I-beams, is another approach. Alternatively, reducing the point cloud's resolution might simplify the detection to one circular beam.

Inexact Drone Pose

The algorithm assumes the drone position to be exact. This however is not the case in the real world and has to be addressed. Inaccurate pose information can lead to errors in the transformation of point cloud data from the drone frame to the world frame. This misalignment can cause inaccuracies in segment detection, fusion, and intersection identification. These errors could result in an incorrect representation of the physical environment, potentially leading to false detections or missed structural elements. Consequently, it is essential to consider these uncertainties and implement methods to mitigate their impact, such as more robust data fusion techniques or error correction algorithms.

Multiple Radius Segment Detection

The algorithm is currently limited to detecting segments with a single radius, but it has the potential to handle multiple radii with further development. To detect various radii, the algorithm would need to process the same point cloud repeatedly, each time using a different radius. After processing for one radius, points that are part of the extracted segment should be removed before analyzing the next radius. For optimal functionality, segments should be processed in descending order of their radius sizes.

Fusion Fails

The segment fusion process occasionally encounters challenges, particularly when it mistakenly identifies the ends of two segments as distinct, resulting in segments that overlap. While this issue requires attention, its impact on the overall performance is limited due to its infrequent occurrence.

This problem occurs in instances where the drone captures only the two ends of a segment but misses the middle part. Such situations are uncommon and typically result from abrupt drone movements, leading to gaps in data between two consecutive Time of Flight (ToF) point cloud readings. To mitigate this issue, operating the drone at a slower speed and with smoother movements can be an effective solution.

Ground Truth Offset

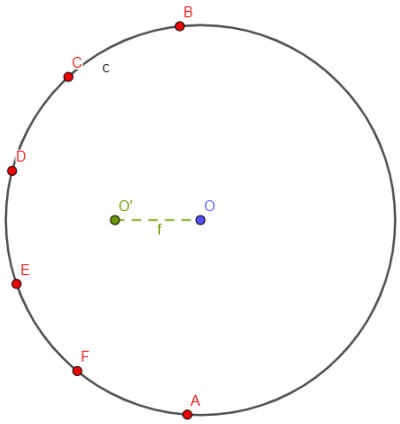


Figure 17: Ground Truth Offset, Beam Cross Section

In Figure 17, the discrepancy between the computed and actual beam positions is evident. The figure illustrates this with red points representing the point cloud, green point indicating the computed segment axis based on the points' mean, and blue points showing the ground truth. A noticeable offset between the computed points and the ground truth can be observed.

The algorithm inaccurately identifies the central axis of beams, as shown in the beam cross-section in Figure 17 and further observed in Section 5, Figure 12. This issue arises from the point cloud's distribution on the beam. One possible correction involves using the circle equation $(x - x_c)^2 + (y + y_c)^2 - r^2 = 0$ and minimizing a system of equations with all beam points to find the nearest solution to the beam's center. However, this approach is viable only for cylindrical segments, necessitating alternative solutions for non-cylindrical beams like I-beams.

Segment Division upon Intersection

Upon finding intersections, it would make sense for segments to be divided corresponding to the intersection distance. This however is not an evident task as it has to be updated each iterations. Only segments that are newly intersecting or have modified intersections need to be divided. This leads to an increased number of segments to process, each with its own set of dependencies and interactions.

6.4 Future Directions and Improvements

After tackling the mentioned limitations, the algorithm offers significant potential for further enhancements and development.

GTSAM optimisation

GTSAM can enhance the structural model obtained using segments and intersections [6]. It can represent each segment as a factor in a graph, with intersections acting as constraints. GTSAM's optimization algorithms can be utilized to refine segment positions and orientations, ensuring alignment with intersection constraints. This improves overall accuracy and ensures consistency, especially in structures with multiple connecting paths.

Real World Testing

For future research, real-world testing is key to evaluating the algorithm's effectiveness in varied environments. This practical assessment will reveal its robustness and areas for improvement, guiding enhancements for reliable real-world application.

7 Conclusion

This section wraps up the report, highlighting the project's successes in developing a 3D point cloud data segmentation pipeline for Micro Aerial Vehicles and outlining future enhancements.

7.1 Key Achievements and Overall Results

The project's primary achievement is the development of an advanced, real-time segmentation pipeline for 3D point-cloud data analysis from Micro Aerial Vehicles (MAVs), enhancing the precision and efficiency of structural inspections. Key achievements include the implementation of a dynamic threshold strategy for detecting segment similarity and a fusion technique that considers segment quality. Additionally, real-time processing capabilities have been realized.

7.2 Potential Future Directions

Future enhancements of the project include refining the algorithm to better handle intersections at flat angles and complex junctions with multiple segments. Expanding the system's capabilities to recognize diverse beam shapes and segments of varying radii will enhance its versatility. Addressing ground truth offset issues will improve accuracy, while implementing segment division at intersections will offer a more detailed structural analysis. Additionally, making the algorithm more resilient to positional noise from the drone and integrating GTSAM optimization will further refine structural model accuracy. Finally, real-world testing is essential to validate the system's effectiveness in various environments and guide further refinements. These improvements should generate a material improvement on the current system.

7.3 Final Thoughts

This investigation into the Hough transform-based structure segmentation pipeline, now integrated within a ROS node, has revealed important insights about the algorithm's performance in extracting steel structures. It effectively showcases the Hough transform's ability to identify diverse structural configurations, but it also illustrates some challenging aspects, particularly in the post-processing of segments and the accurate integration of historical data with new inputs. While there is room for improvement, the use of 3D Hough lines in detecting structures turns out to be very promising, opening up new possibilities in the realm of autonomous infrastructural inspection.

References

[1] C. Dalitz, T. Schramke, and M. Jeltsch, "Iterative Hough Transform for Line Detection in 3D Point Clouds," *Image Processing On Line*, vol. 7, pp. 184–196, 2017. https://doi.org/10.5201/ipol.2017.208.

[2] C. Dalitz, T. Schramke, and M. Jeltsch, "Iterative hough transform for 3d line detection." https://github.com/cdalitz/hough-3d-lines, 2017.

[3] ROS Contributors, "Ros noetic ninjemys." https://wiki.ros.org/noetic, 2020.

[4] Point Cloud Library Contributors, "Point cloud library (pcl)." https://pointclouds.org/, 2011.

[5] I. M. Johnstone and A. Y. Lu, "imjohnstone.su.domains." https://imjohnstone.su.domains//WEBLIST/AsYetUnpub/sparse.pdf, 2004.

[6] Georgia Tech Robotics and Intelligent Machines, "Gtsam: Georgia tech smoothing and mapping library." https://gtsam.org/.

[7] Cyberbotics Ltd., "Webots: Open source robot simulator." https://cyberbotics.com/, 2020.

[8] T.-T. Tran, V.-T. Cao, and D. Laurendeau, "Extraction of cylinders and estimation of their parameters from point clouds — sciencedirect.com." https://www.sciencedirect.com/science/article/pii/S0097849314001150?via%3Dihub.

[9] Wikipedia contributors, "Hough transform — wikipedia, the free encyclopedia," 2023.