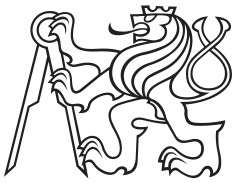
Bachelor Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Tree Detection for UAV Localization

Tereza Zmeškalová

Supervisor: RNDr. Petr Štěpán, Ph.D.
Study program: Open informatics
Specialisation: Artificial Intelligence and Computer Science
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Zmeškalová Tereza** Personal ID number: **499134**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Tree Detection for UAV Localization

Bachelor’s thesis title in Czech:

Detekce stromů pro lokalizaci UAV

Guidelines:

1) Become familiar with the dataset https://github.com/ctu-mrs/slam_datasets/tree/master/forest and how ouster lidar works.
2) Learn about existing lidar data clustering and segmentation methods.
3) Propose a method that detects trees in lidar data and test this method on the above dataset.
4) Create a data structure that contains the location of significant trees from lidar data. Test whether this structure is suitable for drone localization.
5) Compare the localization results with the stored drone locations.

Bibliography / sources:

[1] B. Douillard et al., "On the segmentation of 3D LIDAR point clouds," 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 2798-2805, doi: 10.1109/ICRA.2011.5979818.
[2] V. Bartek, „Improving Detection by Exploiting Dynamics in the Lidar Data“, Bakalářská práce FEL, ČVUT, 2022, https://dspace.cvut.cz/handle/10467/101260.
[3] D. Zermas, I. Izzat and N. Papanikolopoulos, "Fast segmentation of 3D point clouds: A paradigm on LiDAR data for autonomous vehicle applications," 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 5067-5073, doi: 10.1109/ICRA.2017.7989591.
[4] S. W. Chen et al., "SLOAM: Semantic Lidar Odometry and Mapping for Forest Inventory," in IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 612-619, April 2020, doi: 10.1109/LRA.2019.2963823.
[5] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley and C. Stachniss, "SuMa++: Efficient LiDAR-based Semantic SLAM," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 2019, pp. 4530-4537, doi: 10.1109/IROS40897.2019.8967704.

Name and workplace of bachelor’s thesis supervisor:

RNDr. Petr Štěpán, Ph.D. Multi-robot Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **23.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

RNDr. Petr Štěpán, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to express my utmost gratitude to my thesis supervisor RNDr. Petr Štěpán, Ph.D. for his great guidance, advice, endless patience and support while writing my thesis.

I would like to express my immense gratitude to my family, who supported me throughout my studies and allowed me to concentrate fully on my studies. Last but not least, I would like to thank my classmates and friends who have been supportive and helpful throughout my studies and have not allowed me to doubt myself.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 22. May 2023

Abstract

Tree detection from Point cloud (PCL) is crucial for the autonomous movement of drones in the forest as well as for determining its location in a known forest or for approximate localization relative to a takeoff point in an unknown forest. Correct classification and recognition of trees, as well as a suitable algorithm for subsequent localization, is necessary for proper localization. At the same time, using information about all individual trees leads to fewer inaccuracies in long-term localization and the ability to recognize places where the drone has already flown compared to the occupancy grid.

Keywords: point cloud, LiDAR, ground detection, tree detection, RANSAC, localization

Supervisor: RNDr. Petr Štěpán, Ph.D.
room: E-116,
Karlovo náměstí 293/13,
Praha 2

Abstrakt

Detekce stromů z mračna bodů je zásadní pro autonomní pohybování dronů v lese, stejně tak jako pro určení jeho polohy ve známém lese, nebo pro přibližnou lokalizaci vůči místu vzletu v neznámem lese. Pro správnou lokalizaci je nutná správná klasifikace a rozpoznání stromů stejně tak jako vhodný algoritmus pro následné lokalizování. Zároveň použití informace o všech jednotlivých stromech vede ve srovnání s mřížkou obsazenosti k méně nepřesnostem při dlouhodobé lokalizaci a v možnosti rozpoznání míst, kde již dron letěl.

Klíčová slova: mračna bodů, LiDAR, rozpoznání země, detekce stromů, RANSAC, lokalizace

Překlad názvu: Detekce stromů pro lokalizaci UAV

Abbreviations

PCL Point Cloud
LiDAR Light Detection and Ranging
DBSCAN Density-based spatial clustering of applications with noise
IMU Inertial measurement unit
RANSAC Random sample consensus
ICP Iterative Closest Point

Contents

1 Introduction **1**

1.1 Related work 2

2 Ground Removal **5**

2.1 Introduction 5

2.2 Dron points removal 6

2.3 Pillars 6

3 Clustering **9**

3.1 DBSCAN 9

3.1.1 2D approach 11

3.2 Other clustering methods 12

3.2.1 K-Means 12

3.2.2 Meanshift 17

4 Tree classification **19**

4.1 Introduction 19

4.2 Bare trunk classification 19

4.2.1 Cylinder fitting 19

4.2.2 Classification 21

4.3 Full tree classification 22

4.3.1 Classification 22

5 Localization **25**

5.1 Introduction 25

5.2 Data structure 25

5.3 Tree intersection with the ground 26

5.4 Inertial measurement unit (IMU) 28

5.5 Rotation matrix from IMU 28

5.6 Rotation matrix using classified trees 29

5.7 Naive transformations using similarity 31

5.8 Transformations using RANSAC 33

5.9 Iterative Closest Point Algorithm 35

6 Results analysis **39**

7 Conclusions **43**

A Bibliography **45**

B Repository structure **47**

Figures

1.1 Forest dataset 2

2.1 Dron removal 6

2.2 Detected ground 7

2.3 Ground removed 7

3.1 Clustering $\epsilon = 0.45$ 10

3.2 Clustering tree with crown $\epsilon = 0.45$ 11

3.3 Clustering tree with crown $\epsilon = 0.18$ 11

3.4 DBSCAN with 2D projection $\epsilon = 0.12$ 12

3.5 The Elbow method: k = 18 13

3.6 KMeans clustering k = 18 13

3.7 The Silhouette method: k = 2 . . 15

3.8 KMeans clustering k = 2 15

3.9 Kmeans with 2D projection, the elbow method k = 15 15

3.10 Kmeans with 2D projection, the silhouette method, k = 125 16

3.11 Meanshift clustering Bandwith = 10 17

4.1 Tree classification; blue: bare trunk red: contain less than k points; green: satisfies $max_z - min_z < threshold$ and $min_z < 0$; yellow: classified as not tree 21

4.2 Tree classification; blue: tree green: satisfies $max_z - min_z < threshold$ and $min_z < 0$; yellow: classified as not tree 23

4.3 Tree classification; blue: tree green: satisfies $max_z - min_z < threshold$ and $min_z < 0$; yellow: classified as not tree 23

5.1 Tree intersection with the ground 27

5.2 Naive transformation on short . . 32

5.3 Naive transformation on short distance ICP 33

5.4 Naive transformation on long distance 33

5.5 Localization with 12s time difference with drone position highlighted: The original position is marked in red, transformed in yellow 36

5.6 Localization with 5s time difference with drone position highlighted: The original position is marked in red, transformed in yellow 36

5.7 Before ICP 37

5.8 After ICP 37

5.9 Large discrepancy in rotations about the x and y axes before ICP 38

5.10 Large discrepancy in rotations about the x and y axes after ICP . 38

6.1 Dataset part A 40

6.2 Dataset part B 41

6.3 Dataset part C 41

6.4 Visually correct transforms with a larger error 41

6.5 Dataset part A 42

6.6 Dataset part C 42

ctuthesis t1606152353

Chapter 1

Introduction

Computer vision is essential nowadays because it allows us to process data from cameras and sensors. In most industrial applications with autonomous decision-making, the computer must understand its environment and work with it correctly. The Light Detection and Ranging sensor LiDAR, which has become popular in recent years, helps to do this. Lidar has achieved widespread use over other sensors due to its accuracy in measuring distance quickly over large angular ranges.

In this report, we will work with Forest dataset 1.1 from [BP22], which has ground truth odometry for the majority of data but does not include tree detection data. Therefore, providing an exact percentage success rate for detecting individual trees will not be possible. The tree detection results were evaluated on the selected data by examining the point cloud and are documented in detail in the figures. The success of the tree decision method is further evaluated in terms of its use for locating UAVs while moving in the forest. It would be possible to annotate the data manually, but that would be very time-consuming, and it is not the focus of this work.

Localization without GPS is necessary because some places cannot be covered by reliable GPS signal, for example, in the forest. Therefore, the drone must recognize its approximate location compared to the place from which it took off, based only on the data it has measured. In this way, it can be achieved that even if the GPS signal is lost, the UAV can complete the task and return to the starting point. At the same time, if it is an already mapped location, the collected data can be mapped onto already known data, thus achieving even more accurate localization.

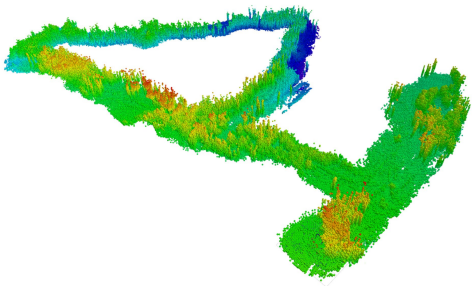


Figure 1.1: Forest dataset

Localization is an important ability that is beneficial to explore and find suitable and robust solutions, especially nowadays when there is a boom in the autonomous activity of drones and other robots.

1.1 Related work

The topic of object segmentation from LIDAR [DUK+11] and localization has already been treated several times in the past with different approaches and on different data.

In this thesis, we are concerned with the detection of trees in the forest. Most of the tree detection works deal with tree detection while flying over a tree canopy. Tusa E. et al. [TMB+21] propose using Mean Shift and crown shape model for segmenting trees that form different types of crowns shows very good results, but in a forest where bare trees without crowns are intermingled it does not achieve such good results.

Another approach for tree segmentation was presented by Qin H. et al. [QFMI15], which focused on segmenting leafy trees from a bird's eye view using LIDAR, ultra-high-resolution RGB data and the watershed algorithm. They achieved perfect results in this direction, but they also focused primarily on deciduous trees and from a different perspective than our intention.

Locating drones in the forest using LIDAR is not addressed in many works. Exciting is the work of Chen Steven W. et al. [CNL+19], which deals with tree and ground detection in forests. They used segmentation and, among other things, Deep Learning to map the forest, which allows them to achieve very good results in this direction.

This work aims not to create an accurate forest map that records all trees accurately but to test the possibilities of robust localization in the forest. For this reason, data segmentation methods for localization in urban environments are relevant to this work [ZIP17, SK19].

Very inspiring is also the bachelor thesis of Vojtěch Bartek [Bar22], which dealt with detecting moving objects in urban environments from Lidar data on a car. In this thesis, we have transferred the ground detection in Lidar data, which has been slightly modified because the ground in a forest environment is not as flat as in an urban environment.

The paper also uses methods based on Density-based spatial clustering of applications with noise (DBSCAN) [EKSX96, Lin72], which provides good results on our dataset. As an alternative approach, K-Means Clustering [LW12] was tested. It is an Unsupervised Learning algorithm which groups the unlabeled dataset into different clusters. In this work, two methods to find the optimal k were tested, The elbow method [SWW+21] and The silhouette method [WFPK+17]].

Chapter 2

Ground Removal

2.1 Introduction

Lidar provides a lot of data; for fast processing, it is necessary to limit the data to relevant points only. To have clean data of trees, we need to remove points that do not belong to the classification, such as ground and drone. It is advantageous to remove them right at the beginning of processing, rapidly reducing the number of points and speeding up the subsequent classification.

Before removing the ground itself, we will start the process by removing the points that belonged to the drone and were also captured during the scan.

Ground point removal is a fairly broad topic with lots of possible approaches. After a more extensive exploration of ground removal methods, we will follow up on Chapter 3 from [Bar22], where V. Bartek compares possible methods of ground removal and will be primarily interested in the Pillars method, which for our purposes, achieves the best results.

2.2 Dron points removal

The drone forms the origin of the coordinate system, i.e. the point (0,0,0). To remove points that belong to the drone, we can work with the distance of the points from the origin of the drone. Our data contains a large amount of directly duplicated points of origin of the drone. Therefore, we first remove all these duplicates located inside the drone diameter and then remove all the points for which apply $p_x^2 + p_y^2 + p_z^2 \leq threshold^2$. For the forest dataset, we set the threshold to 0.35

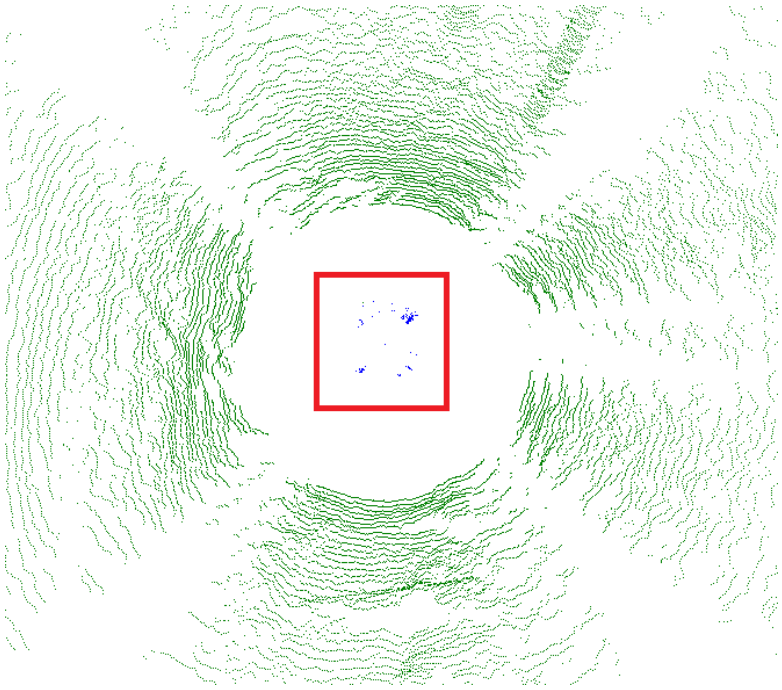


Figure 2.1: Dron removal

2.3 Pillars

The idea of the method of Pillars, proposed by V. Bartek [Bar22], is to divide points into thin pillars and label the lowest point in each pillar as ground. For this purpose, we first neglected the *z-coordinate* of points and made a 2D grid with a specific pillar size. Then we find a minimum on the *z-coordinate* in each pillar. Unlike the work of [Bar22], where the LiDAR moves fixed on a car, the LiDAR on the UAV involves tilts. However the Pillar method is resistant to tilting the drone, so there is no need to consider tilting the LiDAR.

In our case, we choose a 1.8m by 1.8m square pillar size for the grid. Then we use threshold 0.45m and label in each pillar all points for which hold $|p_z - lowest_z| \leq threshold$, where the p_z is a *z-coordinate* of the point to label, and the $lowest_z$ is *z-coordinate* of the lowest point in the pillar.

To improve the results, we add one more stage. Since we have a drone moving in the air at the origin of the coordinate system, i.e. point (0,0,0), we know that the ground will definitely be in negative *z-coordinate*. Therefore we can select from the remaining points in each pillar only those with a negative *z-coordinate*, and if at least for n points, the following applies $p_z - lowest_z \leq threshold$, we will label them as ground points.

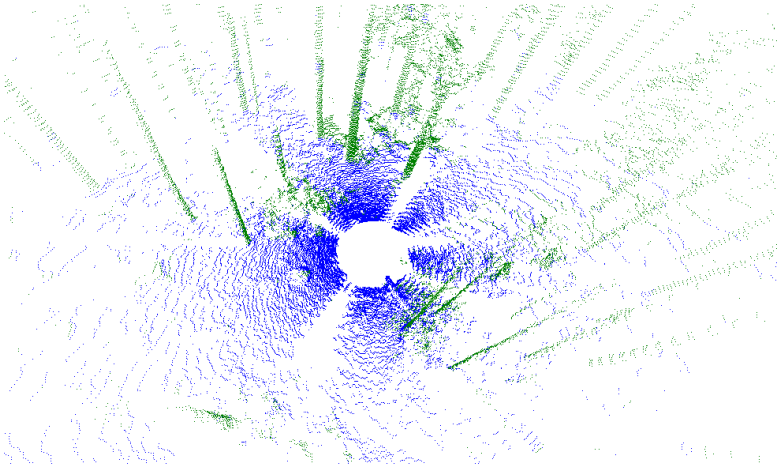


Figure 2.2: Detected ground

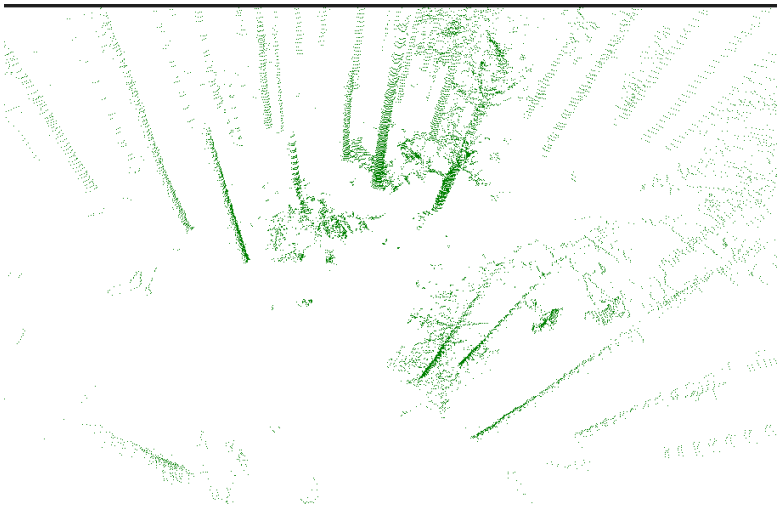


Figure 2.3: Ground removed

Chapter 3

Clustering

After removing the ground points and the drone points, the dataset contains only points that belong to trees and some noise points. Before the actual classification, we need to sort the points into clusters so that we can decide which clusters correspond to trees and which do not.

3.1 DBSCAN

Our data contains noise points that may have been created by the recording itself or by removing the ground. We, therefore, choose Density-based spatial clustering of applications with noise (DBSCAN)[EKSX96, Lin72], which provides the best results on our dataset. DBSCAN takes two hyperparameters:

- eps: specifies how close points should be to each other to be considered a part of a cluster. If the distance between two points is lower or equal to this value, these points are considered neighbours.
- minPoints: the minimum number of points to form a dense region.

For our data, we chose $eps = 0.3m$ and $minPoints = 2$. This setting will generate quite a large number of clusters but mark virtually nothing as non-tree noise points. However, for this setup, we must merge some clusters to have one object in one cluster.

The first step to improving clusters is to check for small clusters. If a cluster containing fewer than k points and is at most a *dist* away from another cluster, we will connect that small cluster to the other one. The distance between two clusters is defined as $min|p_i - p_j| \ \forall p_i \in C_1, \forall p_j \in C_2$, where C_1 is the first cluster, and C_2 is second cluster and $p_{i,j}$ are points from those clusters. For this step, we set the k equal to 50 and the *dist* to 1m.

The second step will be the eventual merger of two larger clusters. Given the relatively small *eps* and *minPoints* parameters, the DBSCAN algorithm may split one tree into multiple parts. So we will check the distance of the two clusters from each other, and if this distance is less than the threshold we set to 0.45 in this step, we will merge the clusters into one.

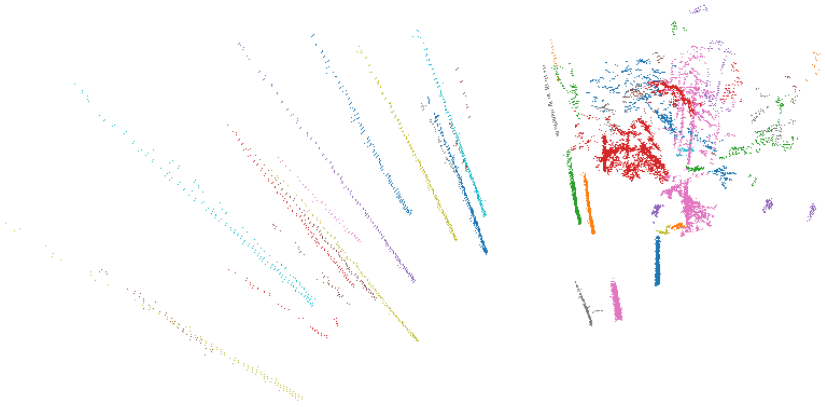


Figure 3.1: Clustering $\epsilon = 0.45$

This setup has proven ideal for classifying bare tree trunks 3.1 but not so ideal for classifying trees with crowns and more branches 3.2. Therefore, after performing this clustering, we evaluate the classification of bare trunks and then perform a modified clustering for the remaining points that were not classified as bare trunks.

For clustering full-size trees, the parameter $eps = 0.18$ is most beneficial. Again, we will have to apply to cluster, but we will have to modify the parameters. We will apply the first step without changes. However, the merging of large clusters will occur if the distance condition from the second step is satisfied and one of the following conditions is also satisfied 3.3:

- $|mean_{x_1} - mean_{x_2}| < threshold_{mean}$ and $|mean_{y_1} - mean_{y_2}| < threshold_{mean}$
- $height_{c_1} < threshold_{height}$

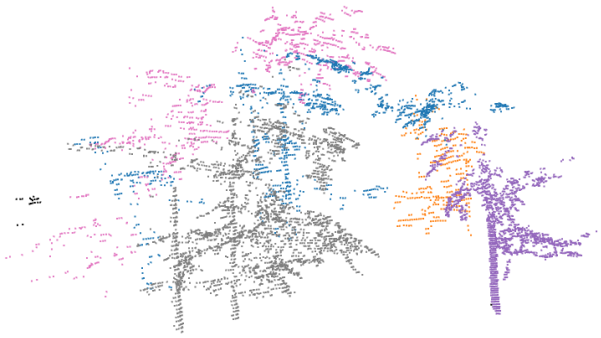


Figure 3.2: Clustering tree with crown $\epsilon = 0.45$

- $mean_{z_1} < mean_{z_2}$ and $|max(x_1) - min(x_1)| < threshold_{clusterSize}$
- $|height_{c_1} - height_{c_2}| < threshold_{heightDifference}$

Where c_1, c_2 are cluster1 resp. cluster2, $mean_x$ are means of x coordinates from cluster1 resp. cluster2, $mean_y$ are means of y coordinates from cluster1 resp. cluster2, $mean_z$ are means of z coordinates from cluster1 resp. cluster2, for heights, the following applies $|max(z_d) - min(z_d)|$ for d=1,2 and thresholds are set $threshold_{mean}, threshold_{height} = 1$, $threshold_{clusterSize} = 2.5$ and $threshold_{heightDifference} = 1.5$

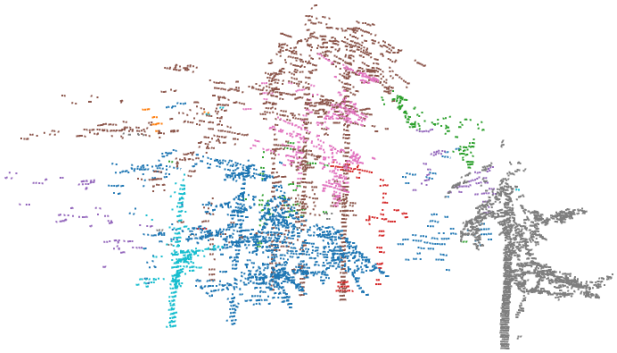


Figure 3.3: Clustering tree with crown $\epsilon = 0.18$

3.1.1 2D approach

In this work, we also tried a new approach of projecting the points into 2D space by omitting the *z-coordinate*. With this procedure, we achieved a

significant improvement in speed, efficiency and even correctness of DBSCAN clustering.

In this case, we can omit the cluster joining algorithms because they are no longer needed. The only problem is merging some full trees whose branches intersect into a single cluster.

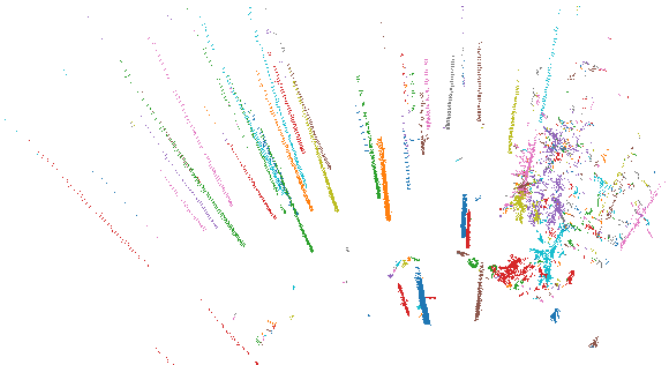


Figure 3.4: DBSCAN with 2D projection $\epsilon = 0.12$

3.2 Other clustering methods

3.2.1 K-Means

K-Means Clustering [LW12] is an Unsupervised Learning algorithm which groups the unlabeled dataset into different clusters. It takes k as a parameter and divides the dataset into k clusters. The disadvantage of this method is the need to know the number of clusters into which we want to divide the points. In our case, it is not possible to know in advance. However, at least two methods exist to find the optimal k. These are The elbow method [SWW+21] and The silhouette method [WFPK+17].

The elbow method

This method is probably one of the best-known and most popular, but its approach is slightly naive.

The idea of this method is straightforward, for a sequence of $k = 1..n$, it

executes the K-Means algorithm and continuously calculates the sum of the distances of a point from the cluster centre. As k increases, the individual distances will be smaller. The ideal k is the one where the most significant break occurs, i.e. where for the previous k, the given sum was much larger, and for the next k, it is only slightly smaller 3.5. However, this method failed

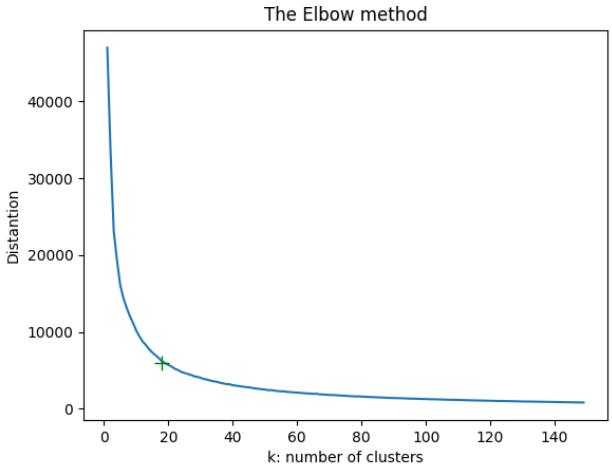


Figure 3.5: The Elbow method: k = 18

to find a good fit for our data such that K-Means clustering performed well, as seen in figure 3.6.

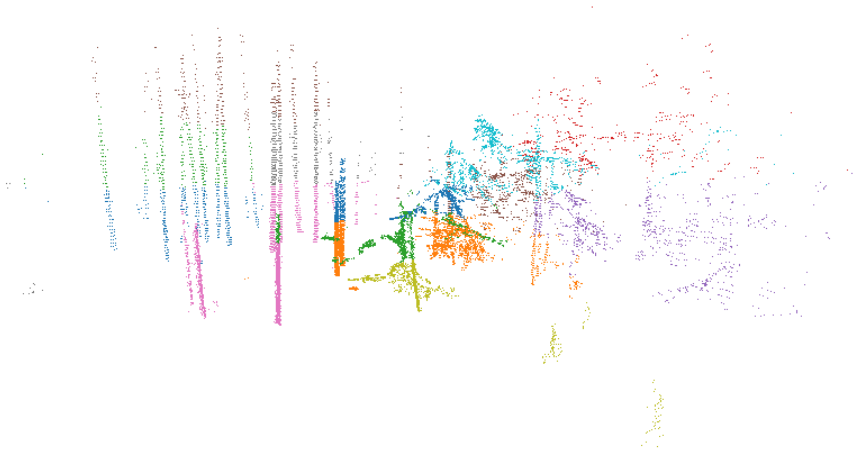


Figure 3.6: KMeans clustering k = 18

The silhouette method

This method is a bit more complex than The elbow method. It uses the silhouette coefficient S. Defined as

$$S(x_i) = \frac{b_{x_i} - a_{x_i}}{\max a_{x_i}, b_{x_i}}$$

Where x_i is a point from the dataset, b_{x_i} is the average distance from x_i to all clusters to which x_i does not belong and a_{x_i} is the average distance between x_i and all the other data points in the cluster to which x_i belongs.

$b_{x_i} = min_{i \neq j} \frac{1}{|X_j|} \sum_{j \in X_j} dist(x_i, x_j)$ where dist is euclidian distance and X_j is cluster j.

$a_{x_i} = \frac{1}{1 - |X_i|} \sum_{j \in X_i, j \neq i} dist(x_i, x_j)$ where dist is euclidian distance and X_i is cluster i.

The silhouette coefficient measures how similar a data point is within-cluster compared to other clusters.

As for The elbow method for the sequence $k = 1..n$, the K-Means algorithm evaluates and computes the average of the silhouette coefficient for all points in the dataset. The k with the highest average coefficient is then chosen as the optimal k.

However, this method did not generate suitable k for adequate clustering on our dataset, as seen in figure 3.8.

2D approach

As for DBSCAN, we achieved better results for kmeans after 2D projection than in the original 3D space. While the elbow method did not give us acceptable results, as seen in figure 3.9, the silhouette method gave us very satisfactory results, as seen in figure 3.10. Still, it significantly increased the program's running time compared to DBSCAN.

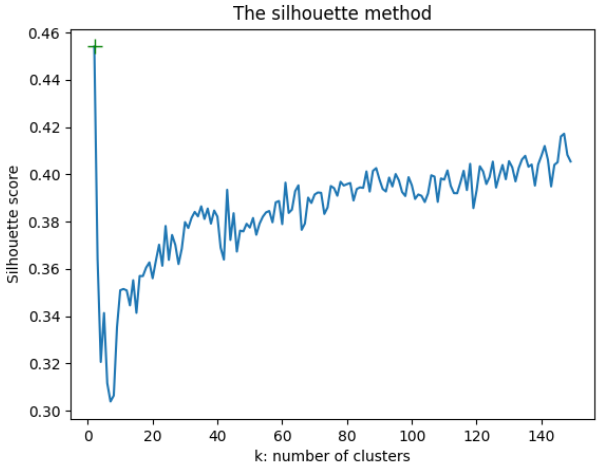


Figure 3.7: The Silhouette method: k = 2

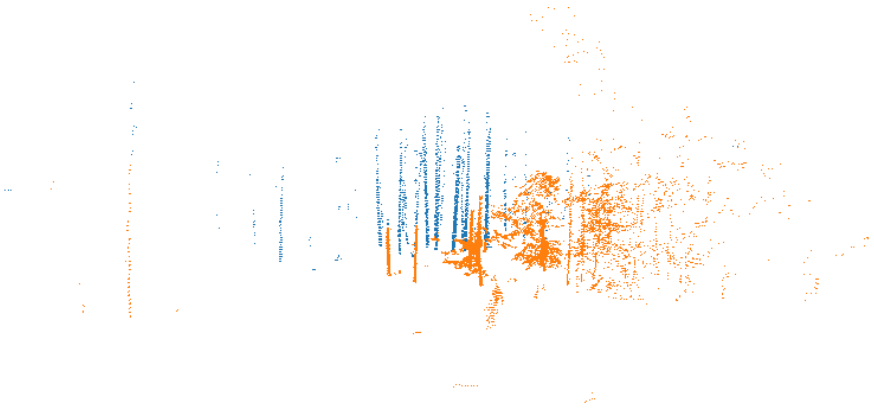


Figure 3.8: KMeans clustering k = 2

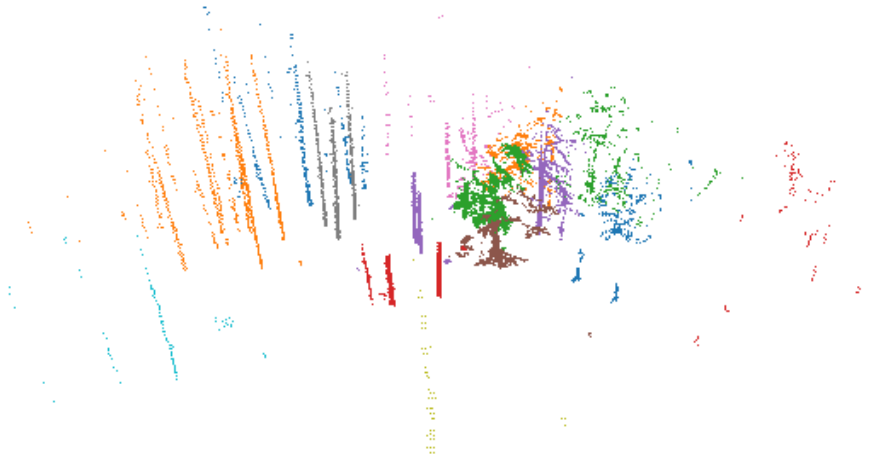


Figure 3.9: Kmeans with 2D projection, the elbow method k = 15

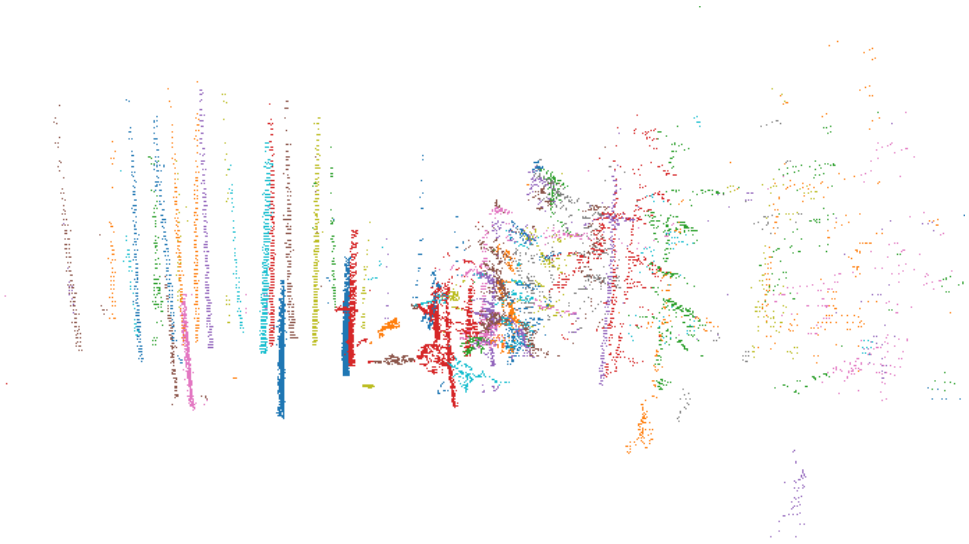


Figure 3.10: Kmeans with 2D projection, the silhouette method, k = 125

3.2.2 Meanshift

Meanshift[Cha22] is a centroid-based algorithm that helps in various use cases of unsupervised learning. It implements mass-centre drift using Gaussian kernel functions. Unlike K-means, it does not require specifying the number of clusters but takes *bandwidth* as a parameter, i.e. the radius of the centre of the mass circle.

This algorithm calculates for each point a weighted mean M_w within the radius from the tested point, and if M_w is not within the $\epsilon - distance$ from the tested point, mean shifting occurs. The whole process is repeated with a new point, M_w until the distance between the point and its weighted mean is less than ϵ, i.e. it no longer converges.

$$M_w = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i} \text{ where } w(d) = e^{-\frac{d}{2\sigma^2}}$$

Although this algorithm does not need to know the number of clusters in advance, the *bandwidth* dramatically affects the final result and is not entirely easy to choose this parameter correctly. Fortunately, some libraries have functions to calculate the optimal bandwidth value. Still, even for the optimal *bandwidth* found, which for our data was set to 10, this algorithm did not provide satisfactory enough results to be used on our dataset, as seen in Figure 3.11.

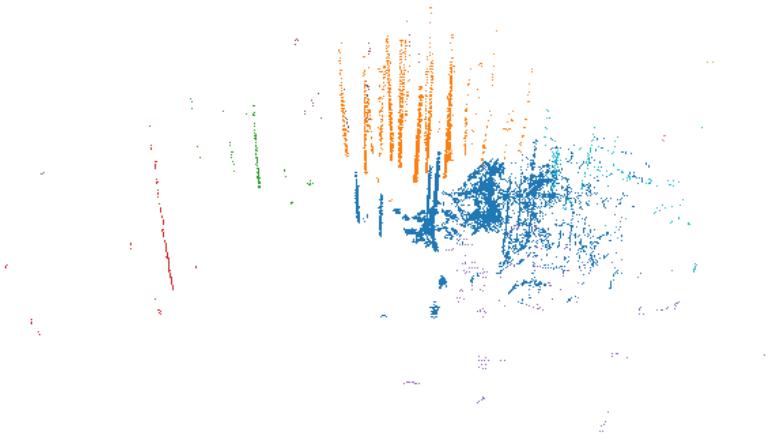


Figure 3.11: Meanshift clustering Bandwith = 10

Chapter 4

Tree classification

4.1 Introduction

As with clustering, there is a significant difference in classification between a bare trunk and a full tree with crowns and branches. While a bare tree has a cylindrical shape, which is great for classification, the classification of a full-sized tree cannot be approached so straightforwardly.

4.2 Bare trunk classification

As already mentioned, the classification of bare trunks differs from that of full-sized trees and is more straightforward and simpler. Since the bare trunk is cylindrical, to classify it, We will use cylinder fitting[Pan17].

4.2.1 Cylinder fitting

Since a bare trunk is essentially a cylinder for most trees in the dataset, we can try to fit a cylinder to the cluster data, and if the error of this fit is less than our defined threshold, we can label the cluster as a bare trunk. The

algorithm[Pan17] is described below. First, let's shift the data's centre of mass to the coordinate system's origin. Then we choose some initial angles and define the G function as: $\sum_i (\langle Y_i, Y_i \rangle - u - 2 * \langle Y_i, v \rangle^2)$

$Y = \langle P, X_i \rangle$ where P is the projection matrix of direction w, and X is our data points,

$u = \sum_i \frac{\langle Y_i, Y_i \rangle}{n}$ where n is a number of data points

$$v = \frac{\langle \hat{A}, \sum \langle Y_i, Y_i \rangle \cdot Y_i \rangle}{\sum_{i=0}^{k} d_i}$$

where d is the main diagonal of $\langle \hat{A}, A \rangle$

$A = \sum_i \langle Y_i', Y_i \rangle$

$\hat{A} = \langle S, \langle A, S' \rangle \rangle$

$$S = \begin{pmatrix} 0 & -w[2] & w[1] \\ w[2] & 0 & -w[0] \\ -w[1] & w[0] & 0 \end{pmatrix}$$

We try to minimize this *G function* by using our data as parameters and our predefined initial angles as direction w. As a result, we take the direction with the minimum value of the *G function* and calculate the other parameters of the ideal cylinder for our data. The centre of the cylinder is calculated as

$c = \frac{\langle \hat{A}, \sum \langle Y_i, Y_i \rangle \cdot Y_i \rangle}{\sum_{i=0}^{k} d_i} + t$, where t is the initial shift of the centre of mass.

Finally, the *radius* is calculated as

$$r = \sqrt{\sum_i \frac{\langle c - X_i, \langle P, c - X_i \rangle \rangle}{n}}$$

4.2.2 Classification

With the ability to calculate the cylinder parameters, including the *fit error*, we can move on to decision-making. We will not consider clusters that contain less than $limit_k$ points; in our case, $limit_k$ is set to 15. Such clusters cannot be a tree because the number of points is too small, and we can skip them. Furthermore, we can also neglect the clusters for which the condition specified below applies because they are too low clusters, and therefore, it is likely that it is not a tree but rather a bush or a person.

$$max_z - min_z < threshold_{limitHeight} \text{ and } min_z < 0$$

After pruning out the unnecessary clusters, we attempt to fit a cylinder to each remaining cluster using the algorithm described above. If the fit error does not exceed the $threshold_{limitHeight}$, we label the cluster as a tree.

However, as mentioned, this method only classifies bare trunks, so for the rest of the clusters, where some still contain full trees, it is advisable to run a modified clustering and use other methods to classify them.

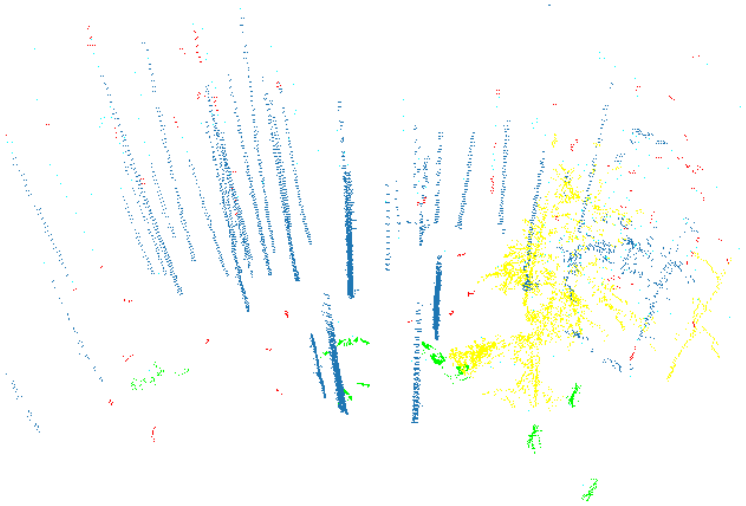


Figure 4.1: Tree classification; blue: bare trunk red: contain less than k points; green: satisfies $max_z - min_z < threshold$ and $min_z < 0$; yellow: classified as not tree

4.3 Full tree classification

The classification of full-size trees is a bit more complicated since the tree has many branches and thus does not form a clear pattern that could be used for identification. Moreover, a similar situation often occurs when we join several nearby tree clusters together. However, even though this problem can be solved by cylinder fitting described in Section 4.2.1, the decision conditions will have to be more complex.

4.3.1 Classification

As mentioned above, we also use cylinder fitting to classify the full-size trees. In the first stage, we marked all those clusters with a fit error less than the threshold and labelled them as bare-trunk trees. These classified trees will no longer be subjected to further classification. Therefore, we are left with clusters containing full trees and other objects that are not trees at all for this stage of classification.

With the parameters we receive after applying the cylinder fitting, we can reconstruct the fitted cylinder and look at the points that satisfy the following:

$$d_x < r + threshold_{cylinderCutout} \text{ and } d_x > r - threshold_{cylinderCutout},$$

where d_x is the distance of the point x from the centre line, r is the radius of the cylinder and the $threshold_{cylinderCutout}$ we set to *0,5* according to the results of the experiments.

Suppose the number of points that fell into the above-specified cylinder cutout is within our specified percentage of original points. In that case, we label the cluster as a tree because a tree trunk fit has probably occurred.

As can be seen in the figures 4.2 and 4.3, this method is quite efficient except for clusters where multiple trees have been merged, and thus the fit of the cylinder cannot be correct.

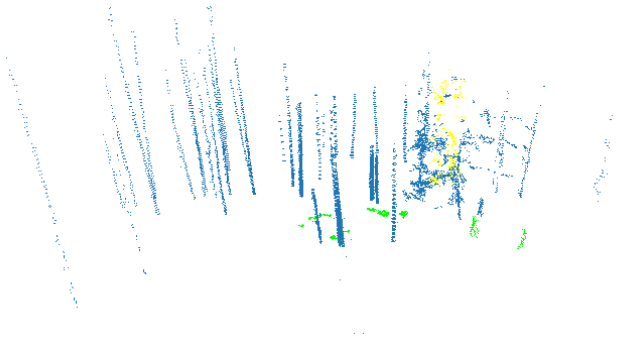


Figure 4.2: Tree classification; blue: tree green: satisfies $max_z - min_z < threshold$ and $min_z < 0$; yellow: classified as not tree

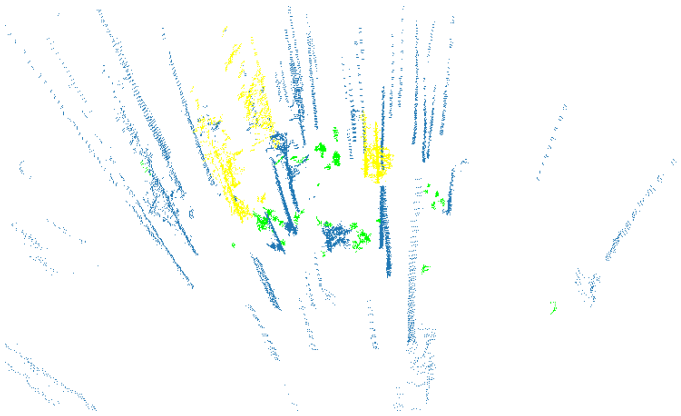


Figure 4.3: Tree classification; blue: tree green: satisfies $max_z - min_z < threshold$ and $min_z < 0$; yellow: classified as not tree

Chapter 5

Localization

5.1 Introduction

Classified trees can be used for accurate and robust drone localization. When the trees are classified, it is possible to start processing the localization of the drone. For localization, we need to have a data structure to store the forest and individual trees with data that characterize the trees or the forest. Then we can get rotation matrices and translation matrices that characterize the new position of the drone based on the tree position information from previous LiDAR data.

Some transformations can be computed using an inertial measurement unit (IMU), but these do not provide complete accuracy, so the data itself and the classification must be used.

5.2 Data structure

For convenient localization, we first need to create a suitable data structure. Since this work is written in Python, Forest and Tree classes were created to store all information about the drone's environment.

Each tree consists of the index that uniquely characterizes it within the forest, the array of points that form it, the classification accuracy, the proportion of points falling within the cutout of the fitted cylinder, the distance from the drone, the distance to the k nearest trees and the approximate point of intersection with the ground.

```
class Tree:
    def __init__(self,points,index, number_of_closest, accuracy
    ↪  = np.inf, ratio = np.inf, distance = np.inf):
        self.index = index
        self.points = points
        self.accuracy = accuracy
        self.ratio = ratio
        self.distance = distance
        self.closest = np.zeros(number_of_closest)
        self.intersection = [0,0,0]
```

The forest consists of an array of individual trees, an array of lines that represent the axes of the fitted cylinders and an array of planes defining the ground around the trees, an array of individual intersections of these lines and planes for each tree, a matrix of attributes (The *i-th* row represents the distances from the k nearest trees to the *i-th* tree of the forest), and a matrix of reciprocal attribute values.

```
class Forest:
    def __init__(self):
        self.trees = []
        self.lines = []
        self.planes = []
        self.intersections = []
        self.sim_matrix = np.matrix([])
        self.sim_matrix_rec =np.matrix([])
```

5.3 Tree intersection with the ground

For some calculations it is worth having only one point that exactly defines the given tree instead of all the points in the tree or instead a vector representation of the tree. The ideal point turned out to be the intersection of the tree with the ground, specifically, the intersection of the cylinder's axis fitted to the tree.

The cylinder's axis is easily obtained by a cylinder fit that returns the centre of the cylinder and the direction vector of the axis, which are sufficient data to get a line representing this axis.

At least three points are needed to represent the plane that represents the ground. However, these points should not be ground points only from the pillar where the tree is located because they would not respect possible changes in the terrain.

We recorded two points for each pillar that best represented the ground surface during the grid creation. These two points are the initial two points that will define the plane. To these, we add one point from each of the four pillars adjacent to the pillar with the given tree. The pillars further away from the drone sometimes contain no ground points or even no points overall, therefore, the final set of points does not always have to contain six points. The problem arises when the tree is surrounded only by these empty pillars; thus, only the original pair of points remains. In that case, outside of our points, we use the tree vector as a perpendicular to the search plane. We can afford to do this because we have no more specific information about the terrain in the vicinity.

When we find these points characterizing the ground, in most cases, they do not lie in the same plane. Therefore, it is necessary to fit the plane through them with the method of least squares distances of points from the plane.

The plane representing the ground and a straight line representing the tree is enough to find their intersection, and thus we get the intersection of the tree and the ground. These intersections are shown in figure 5.1, with a red dot.

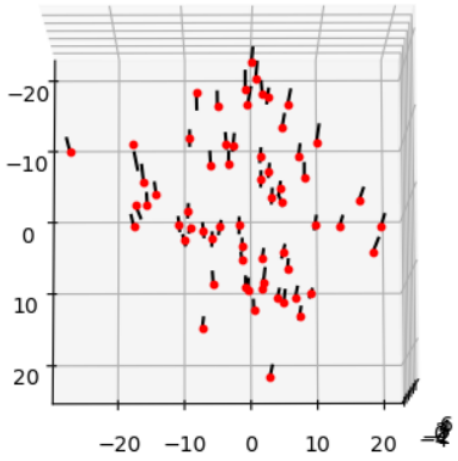


Figure 5.1: Tree intersection with the ground

5.4 Inertial measurement unit (IMU)

An IMU is a device that can measure and report motion data in a time-series format of an object to which it is attached. It contains an accelerometer, gyroscope, and, optionally, a magnetometer or barometer. The accelerometers are responsible for acceleration measurements, and the gyroscopes are responsible for angular velocity measurements. Each one of the measures is represented in a three-axis coordinate system, so generally speaking, they both together yield a 6-dimension measurement time series stream. [BO21]

In addition to the linear acceleration and angular acceleration, our IMU data contain the drone orientation expressed in terms of quaternions, i.e., x, y, z, and w coordinates. There are algorithms to create a rotation matrix directly from the quaternion without transforming them into Eulerian coordinates. Still, for our further calculations, we need to know the rotations with respect to each of the x,y and z axes, so we must first convert quaternion representation to the Eulerian angles. This relatively straightforward transformation was provided in the article *"How To Convert a Quaternion Into Euler Angles in Python"* [aut20]. It transforms quaternions into angles rotated around the x(roll) , y(pitch), z(yaw) axes in radians:

$$roll_0 = 2.0 \cdot (w \cdot x + y \cdot z)$$
$$roll_1 = 1.0 - 2.0 \cdot (x \cdot x + y \cdot y)$$
$$roll_x = atan2(roll_0, roll_1)$$

$$pitch = 2.0 \cdot (w \cdot y - z \cdot x)$$
if $pitch \in < -1; 1 >$, than $pitch$ stays the same
else if $pitch < -1$, than $pitch = -1$
else if $pitch > 1$, than $pitch = 1$
$$pitch_y = asin(pitch)$$

$$yaw_0 = 2.0 \cdot (w \cdot z + x \cdot y)$$
$$yaw_1 = 1.0 - 2.0 \cdot (y \cdot y + z \cdot z)$$
$$yaw_z = atan2(yaw_0, yaw_1)$$

5.5 Rotation matrix from IMU

Since the data from IMU unit are available in the dataset, the initial rotation can be precomputed, which makes the translation matrix and the final

rotation better searched. The primary problem with this approach is the time difference in the recording because the LIDAR scan records roughly half the number of recordings compared to the IMU. It is, therefore, necessary to select the record from the IMU unit that was taken at the closest moment in time to the acquisition of the record from the LIDAR.

After resolving the time discrepancy, we can use the above-mentioned algorithm to convert the obtained quaternions into an Eulerian representation, thereby obtaining the rotation states for each point cloud along individual axes. From this information, we obtain the difference in angles, and these angles give us the rotations along individual axes. It is enough to construct the overall rotation matrix as R, where the order of the individual rotations is according to x(γ),y(β) and finally, z(α)

$$R = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}$$

Due to minor discrepancies in time and due to integration of noisy data from the IMU unit, even the rotation calculated from the IMU can not be completely accurate. In the time between the acquisition of the IMU and the LIDAR recording, a more significant rotation could have occurred, which was not recorded or, on the contrary, was recorded and did not correspond to the orientation at the moment when LIDAR data was captured. Therefore, these rotations are rather approximate, especially for the z-axis.

5.6 Rotation matrix using classified trees

As mentioned in the previous chapter, the rotation from the IMU unit may not always be correct. Therefore, it is necessary to calculate the own rotation matrix using the obtained LIDAR data.

The most important is the rotation around the z-axis; it is usually impossible to correctly calculate this from the IMU unit. Rotations around the x- and y-axis are not so significant, and we are usually able to calculate them correctly from the IMU unit, and if there is a more substantial deviation, we can calculate these rotations correctly, assuming the correct displacement is found, using the ICP algorithm mentioned in a later chapter 5.9.

The final rotation matrix R corresponds to the multiplication of the rotation matrices around individual axes. $R = R_z(\alpha)R_y(\beta)R_x(\gamma)$, where α is angle around z-axis, β angle around y-axis and γ around x-axis.

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}$$

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}$$

For the rotation matrix around the z-axis, we will use the projection into 2D and calculate the rotation matrix between the two corresponding vectors in 2D. Having 4 points forming these two vectors $(A1_x, A1_y), (A2_x, A2_y), (B1_x, B1_y)$ and $(B2_x, B2_y)$, where A1 correspond to B1 and A2 to B2. We can calculate values of a_x, a_y, b_x and b_y that are components of unit vectors for A2A1 resp. B2B1. The required rotation matrix R_{z2D} can be expressed from these values according to formulas 5.1 and 5.2.

$$cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|} \tag{5.1}$$

$$sin(\theta) = \frac{\|\vec{a} \times \vec{b}\|}{\|\vec{a}\|\|\vec{b}\|} \tag{5.2}$$

$$a_x = \frac{(A2_x - A1_x)}{\sqrt{(A2_x - A1_x)^2 + (A2_y - A1_y)^2}}$$

$$a_y = \frac{(A2_y - A1_y)}{\sqrt{(A2_x - A1_x)^2 + (A2_y - A1_y)^2}}$$

$$b_x = \frac{(B2_x - B1_x)}{\sqrt{(B2_x - B1_x)^2 + (B2_y - B1_y)^2}}$$

$$b_y = \frac{(B2_y - B1_y)}{\sqrt{(B2_x - B1_x)^2 + (B2_y - B1_y)^2}}$$

$$R_{z2D} = \begin{pmatrix} a_x \cdot b_x + a_y \cdot b_y & b_y \cdot a_x - b_x \cdot a_y \\ b_x \cdot a_y - b_y \cdot a_x & a_x \cdot b_x + a_y \cdot b_y \end{pmatrix}$$

The last step is to extend this rotation matrix for the 3D dimension.

$$R_z = \begin{pmatrix} R_{z2D} & 0 \\ 0 & 1 \end{pmatrix}$$

5.7 Naive transformations using similarity

This method aims to find two pairs of matching trees in two LiDAR data. These pairs will already provide us with the computation of the translation and rotation of the drone between the data. The trees are selected based on the attributes assigned to each tree. The attributes are the distance to k nearest trees.

In Chapter 5.2, we stated that for each forest, we have a matrix of individual attributes and a matrix with reciprocal values of those attributes. With these matrices available, we can multiply the matrix with the standard values of one forest and the matrix with the reciprocal values of the other forest. In this way, for each tree from the first forest, we will find out which tree from the second forest has the most similar attributes because the closer the attribute values of trees t_i and t_j are (when t_i is a tree from the first forest and t_j is a tree from the second), the value at indices i,j in the resulting matrix approaching the value of k (corresponding to k closest trees defined in the chapter about data structure 5.2). For our testing, we choose $k = 3$.

With the matrix prepared in this way, we can find two trees from the first forest to which we could assign trees from the second forest with the smallest possible difference in attributes. After selecting these trees, we find the vectors between the two trees in the respective forests and calculate the rotation matrix around the z-axis. Since the rotation around the x and y axes tends to be minor, the angles calculated from the IMU can be used, and only the z-axis angle of rotation can be replaced with the newly calculated one.

Before the translation matrix is calculated, the points of the second forest must be rotated with the obtained rotation matrix because the rotation affects the direction and size of the translation, especially for trees that are close to the drone. After rotation, we need to take the means of each pair of the

trees represented by their intersection with the ground (the intersection of the trees from the second forest must also be rotated), express the vector between these points, which will define the translation, and create the translation matrix T, where T_x, T_y and T_z correspond to the individual components of the obtained vector.

$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

This approach usually only works for a limited distance when the drone can still see similar trees and is able to find a pair of trees in both old and new data. As you can see in the figure 5.2, the transformations are not 100% correct for all trees, but the ICP algorithm will solve this, as you can see in the figure 5.3.

Since we select only two pairs of trees with the closest match without further examination, discovering a new occluded tree can easily result in complete failure, as the new tree may cause a closer match in symptoms at a non-matching location. At that moment, when we choose trees that do not match, the entire rotation and displacement do not correspond to reality. Therefore, this method is unsuitable for longer distances and rapidly changing environments, as shown in figure 5.4.

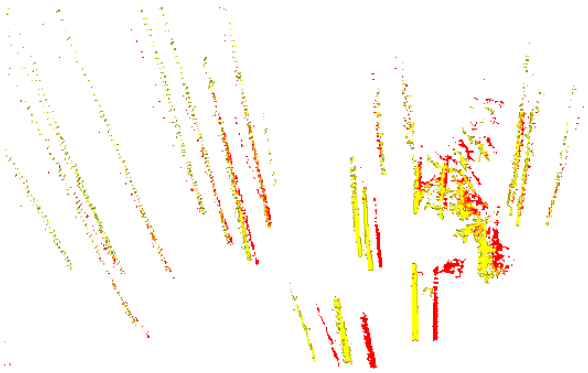


Figure 5.2: Naive transformation on short

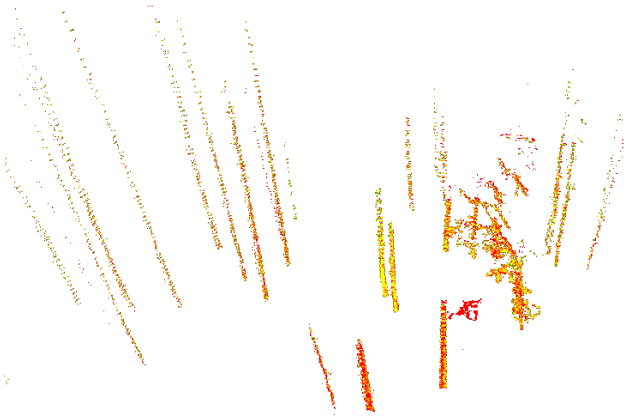


Figure 5.3: Naive transformation on short distance ICP

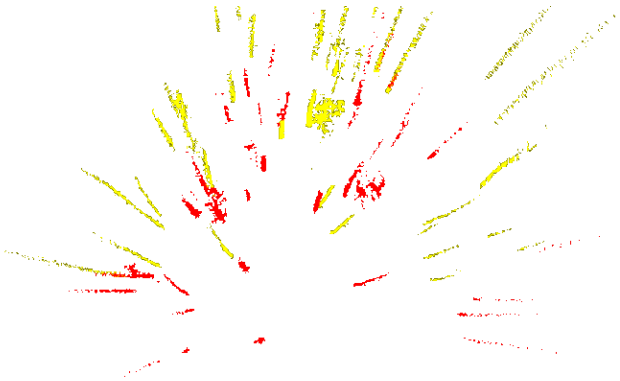


Figure 5.4: Naive transformation on long distance

5.8 Transformations using RANSAC

In this chapter, we will focus on a more complex localization method that already works with more conditions and does not rely only on the best similarity.

As in the previous chapter 5.7, we will count how similar the trees from the first forest are to those in the second forest. This time, for each tree from the first forest, we select three trees from the second forest with which it has the most similar attributes. This will expand the choice, and if a tree that was not visible was revealed to us in the record of the second forest, and thanks to it, a better match in attributes was created, the right pair will also be available for selection.

Using RANSAC, we always randomly select two from this set of pairs. The problem, however, was choosing whether the new pair was better than the old one. It was necessary to add control conditions that would be able to decide on quality. The first factor that contributes to the decision is the similarity in attributes. However, as seen in the previous chapter 5.7, attribute similarity alone is insufficient. Therefore, another factor will be the difference in the distance between the trees. For a pair of trees within their forest, we will calculate the distance between them and then compare these distances, and they will serve as another decision criterion. Despite rotation and displacement, this distance should remain as similar as possible.

However, even the combination of these two conditions is insufficient because a situation may arise when the attributes are almost identical, and the distances correspond. Still, each pair applies an entirely different translation. Therefore, it is also necessary to arrange the match in this attribute. This can be achieved using cosine similarity.

"Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis."[HKP12]

The vectors used to verify the cosine similarity will be created as translation vectors between individual pairs of trees. And then, we calculate the similarity using the formula 5.3 where $\vec{a}$ is the translation vector between the first pair of trees and $\vec{b}$ between the second pair.

$$sim = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|} \tag{5.3}$$

These three values proved almost sufficient to distinguish which pairs are more suitable. We create one score from the similarity in attributes (more precisely, difference in attributes) and difference in distance between the trees within their forests by multiplying them so that both values participate in the main decision criterion. Since we combine the value by multiplication, both will contribute equally to the score. As both represent the difference that we want to minimize, the smaller both values are, the smaller the resulting score will be.

The new pair will be considered better if their score is better (lower) than the original one. At the same time, the distance difference is not bigger

than 1,5m (this condition is necessary because sometimes the difference in attributes is so slight that it overcomes a larger distance difference), and also, the cosine similarity is greater than 0.97.

We also allow a new pair to be selected as better if the score is at most 0.005 less than the current best pairs but only if the distance difference is at most 0.2m and the cosine similarity is greater than 0.99. In this way, we will make it possible to enforce a pair that has almost the same score as the best but probably a better cosine similarity, which plays a significant role in the result because the larger the angle between the individual vectors, the worse the final translation.

All these calculations were performed on the data rotated by the rotation matrix obtained from the IMU, but as already mentioned, this rotation may not always be very accurate, so it is necessary to verify that the cosine similarity remains greater than 0.97.

Having selected two pairs of trees, the procedure for finding the rotation and translation matrix is identical to that of the naive approach in the previous chapter 5.7.

After finding suitable conditions for selecting the best pairs, this approach turned out to be very good, thanks to the random selection of individual pairs from a larger number than in the naive approach and a sufficient number of repetitions of these selections.

On data, when the drone flies more and not too many new trees appear around the drone, it can locate itself even after 12 seconds of drone flight 5.5. In situations where more new trees appear or if the drone flies faster, it can locate itself after about 5 seconds of flight 5.6. These achieved results, given the complexity of data processing, give the assumption applicability for real-time localization.

5.9 Iterative Closest Point Algorithm

Although the results achieved by our transformations are very good, there is still a little room for improvement that the Iterative Closest Point (ICP) algorithm[1] can provide us.

[1] https://github.com/pglira/simpleICP/tree/master

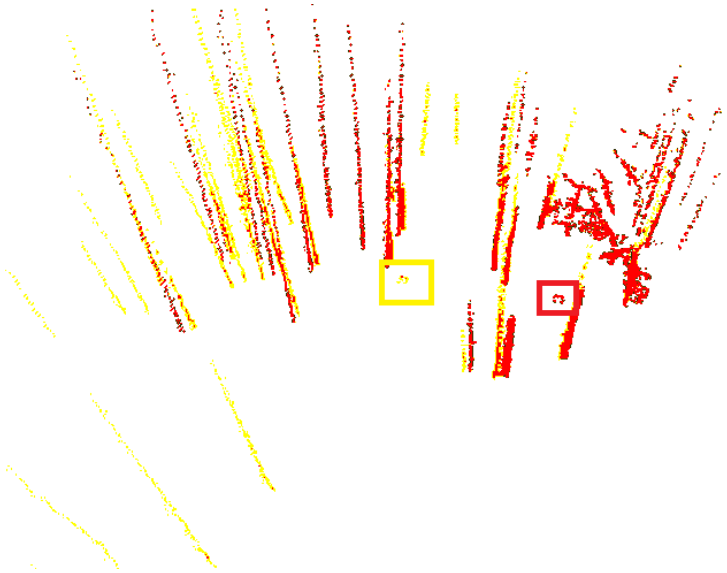


Figure 5.5: Localization with 12s time difference with drone position highlighted: The original position is marked in red, transformed in yellow

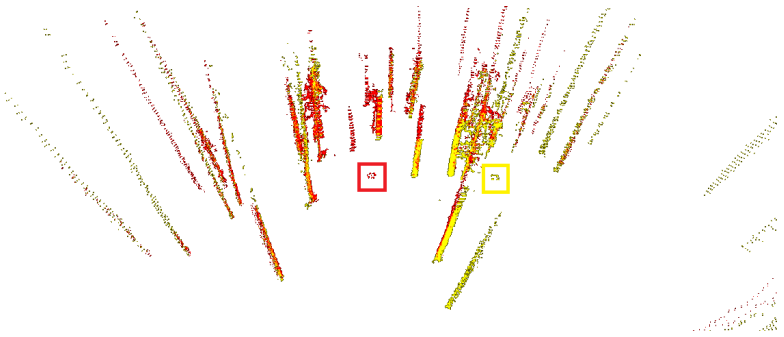


Figure 5.6: Localization with 5s time difference with drone position highlighted: The original position is marked in red, transformed in yellow

"ICP algorithm principle: Given a reference point set P and a data point set Q (at a provided initial estimate R, T), the algorithm finds the corresponding nearest point in P for each point in Q to form a matching point pair. Then, it uses the sum of the Euclidean distances of all matching point pairs as the value of the error objective function error"[ZSY+22]

This algorithm provides good results if there are not too many differences between two clusters of points and if they do not contain too many noise points. Therefore, using the ICP algorithm from the beginning was not advisable because when the point clouds are far from each other and overlap only partially, ICP cannot find suitable transformations. But at the moment, when we already have the points rotated and shifted by our transformations, the parts that overlap are already very close to each other. At that moment, ICP can beautifully match minor discrepancies even though the point clouds

do not fully overlap. As shown in the figures below, where figure 5.7 show transformations before ICP and figure 5.8 after ICP.

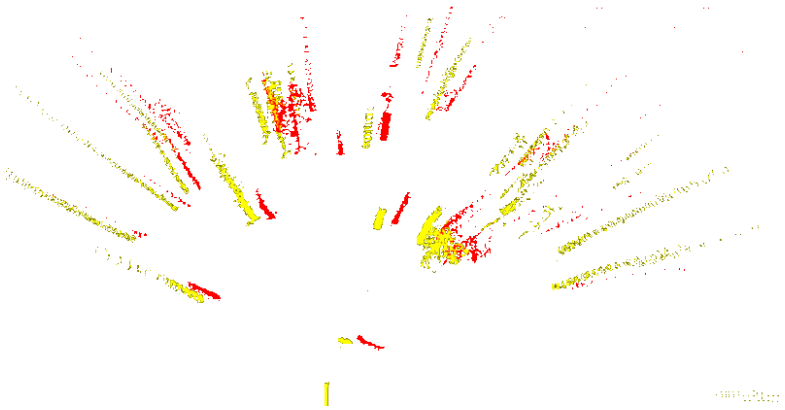


Figure 5.7: Before ICP

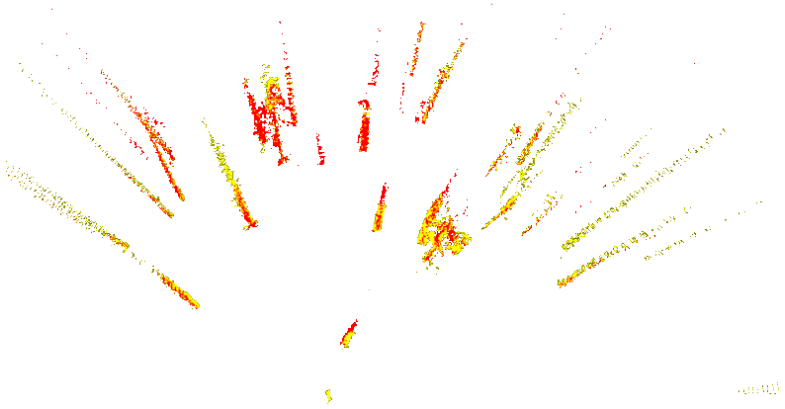


Figure 5.8: After ICP

At the same time, ICP handles the situation well when the overall translation and rotation around the z-axis are adequate, but the rotations around the x and y axes calculated from IMU must be fine-tuned. ICP manages this alignment even in the case of the need to align larger rotations around these axes, as shown in figures 5.9 and 5.10. This confirms that we can use the rotations around the x and y axes from the IMU unit, and any discrepancies can be aligned by the ICP algorithm.

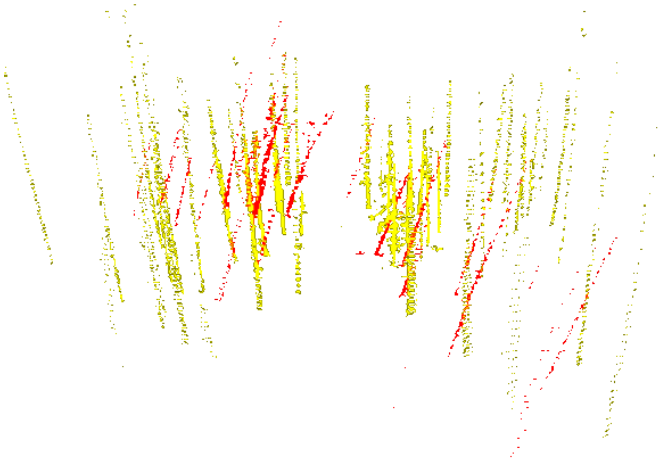


Figure 5.9: Large discrepancy in rotations about the x and y axes before ICP

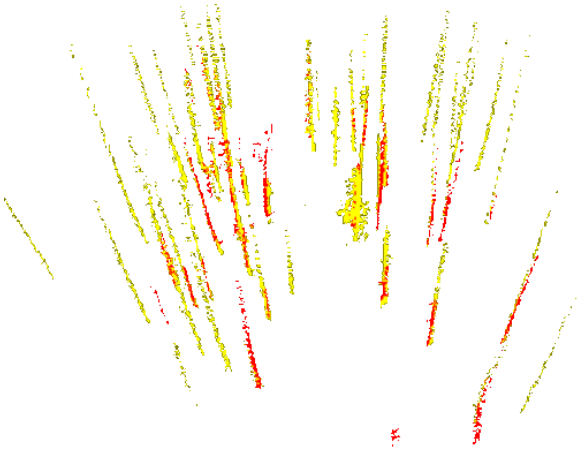


Figure 5.10: Large discrepancy in rotations about the x and y axes after ICP

Chapter 6

Results analysis

In this chapter, we will evaluate the achieved results and compare the proposed methods. We will use data from the drone's RTK (Real Time Kinematic) unit to evaluate the results. Because the dataset is huge and difficult to work with, testing was done on three sections, representing different types of forest, hereinafter referred to as Parts A, B and C. Since we do not have the necessary rotation matrices to be able to calculate the correct flight direction relative to the initial take-off point and the fact that the drone can move to the right and left without the need for rotation, to evaluate the localization method we will compare only the distance that the drone flew (i.e. the magnitude of the shift).

The direction of the shift can then be evaluated based only on the figures by humans; if the direction is incorrect, it would be visible in the figures that the shift was incorrect even in the case of identical shift sizes. The accuracy can be evaluated according to how the distance that the drone flew according to RTK and the distance calculated by our method differ. Of course, it should be considered that since the drone is moving in the forest, even the RTK data can be inaccurate in some cases.

In Figures 6.1 to 6.3 you can see the result of localization of the data measured at different distances from the original data. Localization is always performed only on the first data to determine how often localization needs to be performed. As seen in Figures 6.1 to 6.3, using the method with RANSAC achieves perfect results on average up to a distance of 8m, and the average distance difference is about 0.2m. It is not possible to determine from the data whether this error is an RTK error or an error in out algorithm. As seen

on the individual graphs, the estimated accuracy of the ICP algorithm copies the curve of the difference in distances quite reliably and thus agrees with the achieved results.

The only problem can be seen in Figure 6.2, where there was a fluctuation in error at a distance of about 2m. One of the possible reasons why this happened is inaccurate data from RTK. This is when the drone performs many greater rotations simultaneously, and inaccuracies could therefore occur from this point of view. Indeed, if we look at figure 6.4, we can see that the transformations were successful; an error of almost 1 m in the shift would undoubtedly have a significant effect, which doesn't occur.

To be more sure whether this situation is really an error on the side of the RTK or on the side of the algorithm, we tested all possible transformations that correspond to the translation obtained from the RTK with a difference in distance of at most 0.3m and none of these results were satisfactory, indicating that the error occurred with more likely on the RTK side. However, even if this error was caused by an insufficient condition in the algorithm, it can be seen that the algorithm managed to recover from this problem, subsequently reduce its error, and worked without a problem on the remaining tested parts of the dataset.

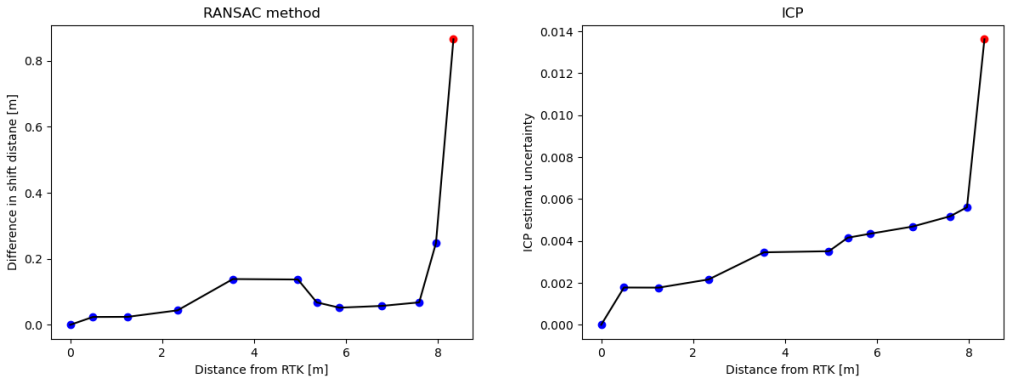


Figure 6.1: Dataset part A

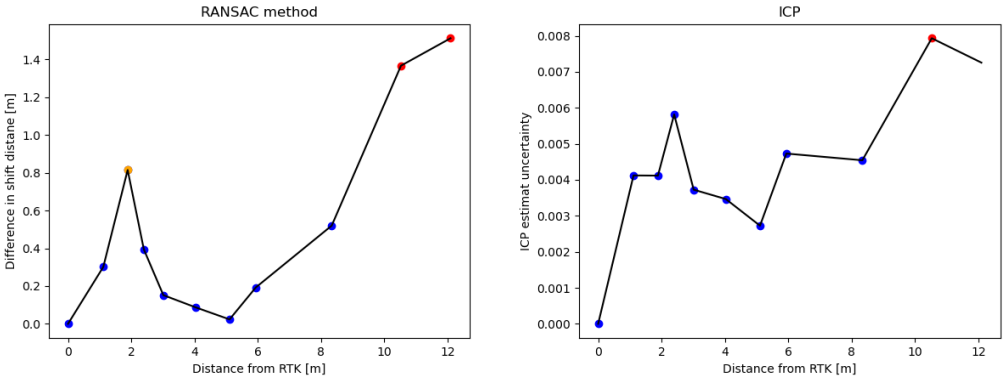


Figure 6.2: Dataset part B

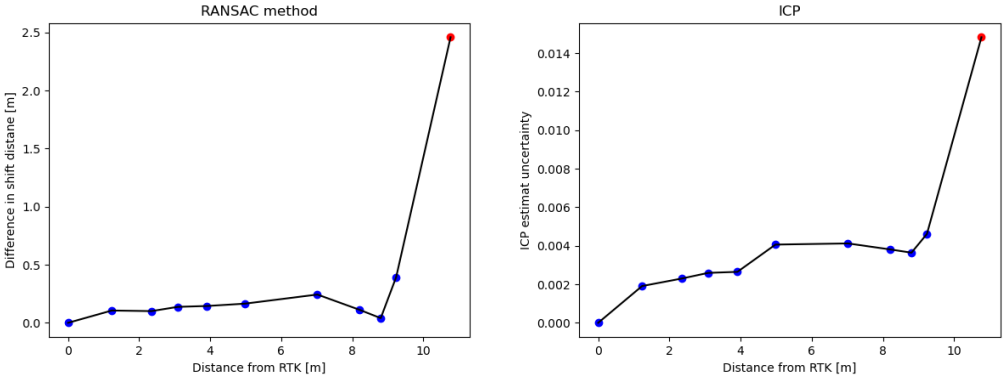


Figure 6.3: Dataset part C

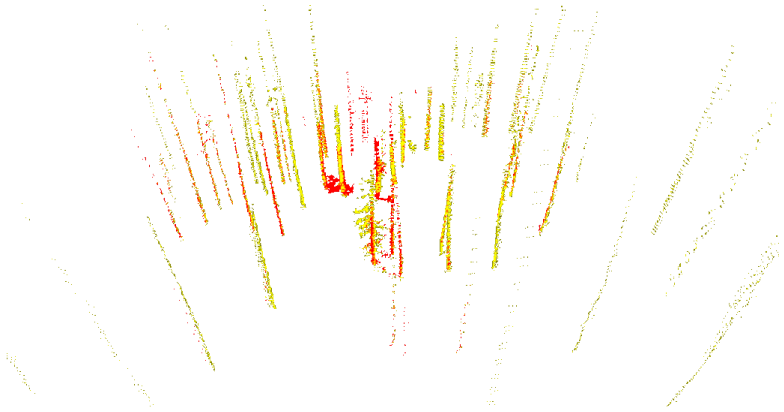


Figure 6.4: Visually correct transforms with a larger error

On the other hand, we also tested a naive approach that only selects two pairs of trees with the slightest difference in symptoms. Excellent results were achieved here, in some cases even a few centimetres better than with the RANSAC method, but only at shorter distances, where there are no significant changes in the number of trees. For longer distances with more

significant changes, the method started to provide quite random results, as can be seen in Figures 6.5 and 6.6, where the graphs of both methods are shown for comparison. It can be seen that the technique works quite well for short distances, but for longer distances or rapidly changing environments, it is unsuitable. On the other hand, it is suitable using RANSAC method.

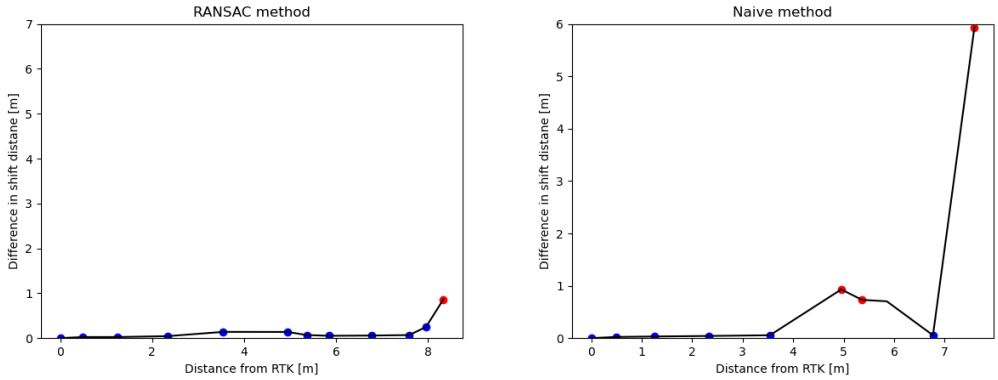


Figure 6.5: Dataset part A

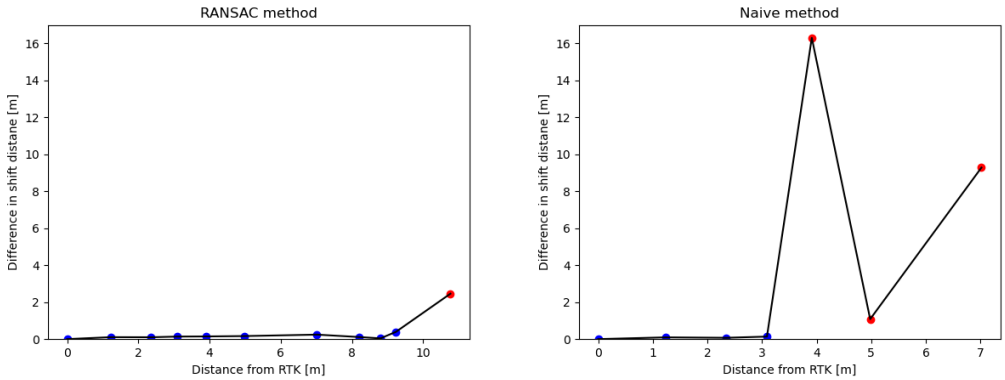


Figure 6.6: Dataset part C

Chapter 7

Conclusions

In the first part of the work, we tested removing the ground using the pillar method. Although this approach was designed for an urban environment with flat terrain, this approach proved to be the most appropriate for our data, where we must work with more uneven terrain.

The bigger problem was clustering, which is necessary for the whole classification. As we have shown, much more accurate results can be achieved by clustering on data with 2D projection. There is also a significant speedup compared to the original 3D clustering, where it was necessary to perform subsequent cluster merging to achieve the desired results. In 3D space, only the DBSCAN method was applicable; however, after 2D projection, the Kmeans method also achieved very decent results using the Silhouette method to find the optimal k. However, finding the optimal k is very time-consuming, so it is not entirely suitable for this problem.

Even in this way, absolutely accurate clustering does not occur, which subsequently affects the result of the classifier because sometimes we merge multiple trees into one cluster.

For classification, fitting the cylinder to the data achieves very good results. We divide the whole classification into two phases. First, we classify the bare trunks, where we need the fit of a cylinder and decide whether the error of this fit is small enough to say that it is a tree. The second phase is then the classification of full trees, where we cut points satisfying $d_x < r + threshold$ and $d_x > r - threshold$ from the fit cylinder. If the number of points remaining is at least, in our case, 50% of the original points, we classify the cluster as a

tree.

However, this classification fails in clusters with multiple trees in one cluster because it is impossible to fit corectly the trunk of one tree when there are multiple trees.

In terms of localisation, we managed to achieve respectable results. We have presented two methods where; one of them works with excellent accuracy at short distances, and a little-changing environment, and the other is capable of localization at distances up to about 8m with very good precision.

Still, there is room for improvement here as well. It is possible that a better criterion could be found for the RANSAC method such that there would be no need for possible fine-tuning using the ICP algorithm.

The most considerable improvement for this work would be to improve the classification for multiple trees in a single cluster, as this shortcoming subsequently affects the localization negatively, as from a different position, these trees may be divided into various clusters and thus figure differently in the localization.

In summary, we get very good results, especially in terms of ground removal and localization. This work can be improved in the areas of clustering. A new algorithm can be designed to detect multiple trees in a single cluster and an even better criterion could be found for selecting suitable pairs of trees in the RANSAC method for localization.

Appendix A

Bibliography

[aut20] automaticaddison, *How To Convert a Quaternion Into Euler Angles in Python*, `https://automaticaddison.com/how-to-convert-a-quaternion-into-euler-angles-in-python/`, 2020.

[Bar22] V. Bartek, *Improving detection by exploiting dynamics in the lidar data*, 1–38.

[BO21] PhD Barak Or, *What is imu?*, Towards Data Science (2021).

[BP22] Tomáš Báča and Pavel Petráček, *Slam dataset*, `https://github.com/ctu-mrs/slam_datasets`, 2022.

[Cha22] Amit Chauhan, *Understanding mean shift clustering and implementation with python*, Towards Data Science (2022).

[CNL+19] Steven W. Chen, Guilherme V. Nardari, Elijah S. Lee, Chao Qu, Xu Liu, Roseli A. F. Romero, and Vijay Kumar, *SLOAM: semantic lidar odometry and mapping for forest inventory*, CoRR **abs/1912.12726** (2019).

[DUK+11] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, *On the segmentation of 3d lidar point clouds*, 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 2798–2805.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, and Xiaowei Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, KDD-96 Proceedings (1996).

[HKP12] Jiawei Han, Micheline Kamber, and Jian Pei, *Data mining concepts and techniques, third edition*, 2012, pp. 77–78.

[Lin72] R. F. Ling, *On the theory and construction of k-clusters*, The Computer Journal **2** (1972), 326–332.

[LW12] Youguo Li and Haiyan Wu, *A clustering method based on k-means algorithm*, Physics Procedia **25** (2012), 1104–1109.

[Pan17] Xingjie Pan, *cylinder fitting*, `https://github.com/xingjiepan/cylinder_fitting`, 2017.

[QFMI15] Y. Qin, António Ferraz, Clément Mallet, and Corina Iovan, *Individual tree segmentation over large areas using airborne lidar point cloud and very high resolution optical imagery.*

[SK19] Muhammad Sualeh and Gon-Woo Kim, *Dynamic multi-lidar based multiple object detection and tracking*, Sensors **19** (2019), no. 6.

[SWW+21] Congming Shi, Bingtao Wei, Shoulin Wei, Wen Wang, Hai Liu, and Jialei Liu, *A quantitative discriminant method of elbow point for the optimal number of clusters in clustering algorithm*, EURASIP Journal on Wireless Communications and Networking **31** (2021), 1–16.

[TMB+21] Eduardo Tusa, Jean-Matthieu Monnet, Jean-Baptiste Barré, Mauro Dalla Mura, Michele Dalponte, and Jocelyn Chanussot, *Individual tree segmentation based on mean shift and crown shape model for temperate forest*, IEEE Geoscience and Remote Sensing Letters **18** (2021), no. 12, 2052–2056.

[WFPK+17] Fei Wang, Hector-Hugo Franco-Penya, John Kelleher, John Pugh, and Robert Ross, *An analysis of the application of simplified silhouette to the evaluation of k-means clustering validity*, 07 2017.

[ZIP17] Dimitris Zermas, Izzat Izzat, and Nikolaos Papanikolopoulos, *Fast segmentation of 3d point clouds: A paradigm on lidar data for autonomous vehicle applications*, 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 5067–5073.

[ZSY+22] Qingguo Zhou, Zebang Shen, Binbin Yong, Rui Zhao, and Peng Zhiand, *Theories and practices of self-driving vehicles*, Elsevier, 2022.

Appendix B

Repository structure

```
zmeskter_bp_repository.zip
├── Origin PDCs
│   ├── 1651738565.019267170.pcd
│   └── 1651738569.626969595.pcd
├── PCDs
│   ├── afterICP.pcd
│   ├── class1.pcd
│   ├── class2.pcd
│   ├── clustering1.pcd
│   └── clustering2.pcd
├── classify.py
├── clustering.py
├── examples.ipybn
├── pcdIO.py
├── ploting.py
├── README.md
├── remove_ground_and_dron.py
├── runs.py
└── transformations.py
```