Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology and Methodology

Catenary segmentation and error detection in LiDAR point clouds

Supervisor:
Máté Cserép
Assistant Lecturer

Author:
Attila Láber
Computer Science for
Autonomous Systems MSc

Budapest, 2023

Contents

Chapter 1

Introduction

With the spreading of ADAS[1] [1], LiDAR[2] sensors have become relatively accessible and well-known in the past decade. Apart from autonomous driving, there are many applications of such Light Detection and Ranging sensors in different fields. The sensors can be mounted on drones or aircraft to create high-quality digital elevation models. They are also used in agriculture for crop mapping, in the video game industry for reproducing real-life scenes with millimetre precision, and of course in land surveying for all sorts of tasks.

Railways — next to many other advantages — provide the most environmentally friendly mode of transportation [2], especially thanks to the electrification of railway lines. With increasing track speeds, there is a great demand for speeding up and improving the accuracy of error detection and inspection on such electrified sections. In Hungary, there are a total of over 8010 kilometres of tracks in active use, out of which 3269 kilometres are electrified [3].

The goal of my work is to provide a set of reliable and efficient algorithms to segment parts of the overhead catenary system, and to detect stagger of the contact cable beyond operational limits. I researched the possibility of using artificial intelligence-supported methods for these tasks, and based on my findings I chose to stick with traditional 3D/2D processing algorithms. I also extended the functionality of the framework in which I implemented my algorithms.

There are two input datasets used, both are dense LiDAR point clouds recorded on Hungarian electrified track sections. One was provided by MÁV[3], the other by

[1] Advanced driver-assistance systems
[2] Light Detection and Ranging
[3] Magyar Államvasutak, Hungarian State Railways

GYSEV[4]. These datasets contain a very dense point-based representation of the recorded areas, with the first set containing over 1.5×10^9 points. The algorithms also rely on seed clouds, which contain only the previously found points corresponding to the tracks and the powerline. These are used to filter out irrelevant points, making the algorithms faster and more robust. The outputs are the points belonging to the segmented masts and cantilevers. In the case of the stagger detection, the outputs are the points exceeding the maximum allowed stagger threshold, or a warning message in case of a lack of stagger in the inspected section.

The implemented methods can robustly segment the mast and cantilever points with over 95% accuracy, and detect stagger of the contact cable beyond the allowed thresholds correctly.

[4]Győr–Sopron–Ebenfurti Vasút, the second largest rail infrastructure operator in Hungary

Chapter 2

Technological background

This chapter provides an overview of the technologies reviewed and used, as well as the most important railway terminologies to help understand the problem better.

2.1 LiDAR

LiDAR is a method that determines ranges to its surroundings by emitting a laser beam and measuring the TOF[1] after having collected the reflected feedback. As the laser beam travels at the speed of light, we can easily deduct the distance from the TOF. Figure 2.1 shows this principle.

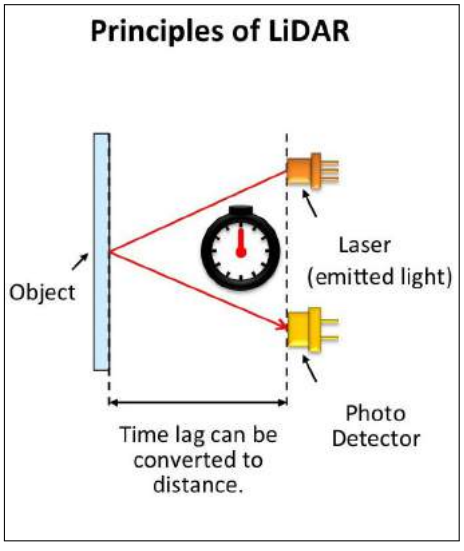


Figure 2.1: *Principles of LiDAR*[2]

[1] time of flight
[2] From Panasonic Newsroom Global: *"Eyes" for Autonomous Mobile Robots*, 2017 [4]

There are different types of LiDAR systems built for specific tasks. Based on the data acquisition method, we distinguish three main types of them:

TLS, or terrestrial laser scanning (often referred to as ground-based LiDAR) is the form of 3D scanning using static, tripod-mounted laser scanners. This technology can reach sub-millimetre accuracy and is mainly used for creating high-resolution geometric data from buildings and various other objects. Another use case is terrain mapping. Its disadvantage is the need for multiple scans for covering larger areas, due to the relatively low range (usually a few hundred metres).

ALS, or airborne laser scanning is the form of 3D scanning using laser scanners directed towards the ground, mounted on aircraft or helicopters. The aircraft can then fly over the surveyed area in a scanning pattern, producing an accurate point cloud of the surface. This technology can be used to cover large areas of interest in short periods of time, at the cost of accuracy. Still, modern ALS systems can reach up to sub-meter accuracy [5], which is mostly sufficient for the main use case of creating digital elevation models.

MLS, or mobile laser scanning is the form of 3D scanning using laser scanners mounted on moving platforms, like boats, trains or road vehicles. This makes it suitable for surveying tasks in both urban and rural environments, where higher accuracy is required than what ALS systems can provide.

The latter two, ALS and MLS also use GNSS[3] and IMU[4] systems for georeferencing the scans.

2.2 Railway catenary systems

The first electrified railways appeared in the 19th century [6]. Since then, electrification of railways has led to a faster and more environmentally friendly way of travelling, beating any other mode of transportation in these regards [2]. These systems are responsible for distributing the electrical current (25 kV 50 Hz in Hungary) along the railway tracks, used by the electric railway vehicles running on such tracks.

[3]Global Navigation Satellite System
[4]Inertial Measurement Unit

Nowadays almost exclusively overhead catenary systems are used in railway transport.

We can categorise the parts of these systems as trackside (masts, wires), or non-trackside equipment (feeder stations, electrical grid). In the following, I will be providing some technical details on the trackside parts of these systems, as the rest are not relevant to my work.

2.2.1 Parts of the catenary

Overhead catenary systems have many different parts (see Figure 2.2), which are constantly being exposed to the weather.

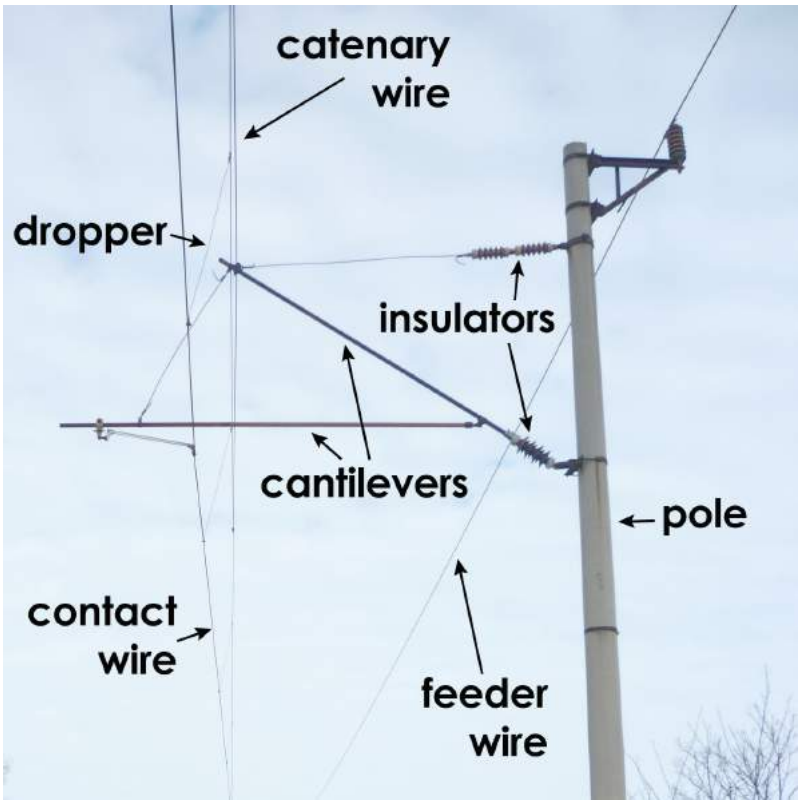


Figure 2.2: *The different parts of overhead catenary systems*[5]

The poles or masts are used to keep the other parts of the system in place. These masts come in many different shapes. There are two main types in use in Hungary today: cylindrical concrete, and steel lattice masts (shown on Figure 2.3). These masts can be installed on either side of railway tracks. The maximum distance between them is 75 metres at installations before 2021, and 69 metres after. They

[5]Adapted from Wikimedia Commons, 2010

are mostly installed orthogonally to the ground plane but can be tilted slightly in curved track sections.

Figure 2.3: *Non-cylindrical steel lattice masts*

The second most important parts are the different wires or cables held in place by these masts. There are three types of such cables:

1. Contact cable: lying the lowest, carries the current for the rail vehicles, being in constant contact with their pantographs[6].

2. Catenary cable: lying in the middle, this cable is responsible for keeping the contact cable in place at higher speeds.

3. Feeder cable: lying the highest, this cable is used on single track lines as a backup to supply current to those sections behind a broken contact cable.

In Hungary, the contact cable generally lies at 6 meter height above rail level. One special property of this cable is its stagger. This is necessary to avoid the cable "carving" a groove in the pantographs. This is shown by Figure 2.4. This stagger is maximally 40 cm at installations before 1995, and 30 cm at newer installations

[6]the device mounted on the roof of electric vehicles to collect power from the overhead wire

since. The allowed discrepancy from these values is 1 cm in the case of European international corridors (category "A") and 3 cm in the case of less important railway lines.

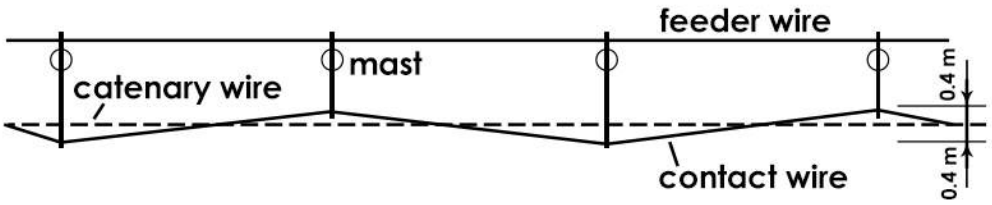


Figure 2.4: *Stagger of the contact wire viewed from the top*[7]

The contact and catenary cables are connected by the droppers. Without them, the contact cable could excessively move around both in the horizontal and vertical directions when vehicles are passing.

The contact and catenary cables are suspended from cantilevers, which are attached to the masts. Between the cantilevers and the masts electrical insulators are placed, so that the current cannot flow through to the masts, potentially causing harm to people and animals.

2.2.2 Diagnostics

According to the Hungarian State Railways' data, more than 68% of pantograph damage and breakage are caused by the failure of the catenary system [7]. The pantographs of the trains are in constant contact with the contact wires. This puts a lot of stress on the system, so naturally, it has to be inspected regularly to check for loose or damaged parts. This visual inspection is usually carried out manually, by railway workers walking along the track, checking for visible defects. This is not only time-consuming and dangerous, but also unreliable for detecting less prominent problems early on.

There are some more sophisticated ways of detecting defects, for instance by using railway vehicles specifically designed for diagnostics of the overhead wire systems (see Figure 2.5). The main drawback of this approach is the low number of such diagnostic units. This also requires running an extra train with a special carriage on the otherwise mostly already busy electrified line networks.

[7]Adapted from Rónai Endre: Vasúti villamos felsővezeték, 1997

To overcome this limitation, some companies provide ready-made catenary monitoring systems, which can be installed directly on the roof of the locomotives pulling the regular passenger/freight trains [8], thus not disturbing traffic flow.

Deploying such systems only makes sense if the units are installed on multiple vehicles, evenly spread throughout the electrified railway network, which is hard to plan for. Installing and maintaining such units in number is certainly not the most cost-efficient either.

As a more accessible and flexible solution, a mobile laser scanner can be fixed on a flat wagon, which is exactly how the dataset introduced later has been acquired.

Figure 2.5: *Catenary diagnostic wagon of the Hungarian State Railways*[8]

[8]From Czifra Zoltán, 2001 (via www.gigantclub.hu)

Chapter 3

Literature

In this chapter, I'll introduce the reviewed literature which I used to gain insight into the most widely used methods to segment railway LiDAR point clouds and measure the stagger of the contact wire.

3.1 Traditional railway LiDAR data processing

For an easier and more accurate segmentation, separating the catenary-related points from the other parts of the scene (rails, track bed etc.) is generally the first step in the process.

3.1.1 Wires

Segmentation

The simple assumption can be made that the contact wire lies in an almost constant height above the tracks [9]. Due to the geometric placement of the cables, a narrow bounding box can be imagined above the tracks, which contains all the points belonging to the contact and catenary wires [10]. Voxelization is another technique that can be successfully used for segmenting the wires as shown by Geng et al. [11].

One more property that can be exploited is the linear nature of these objects. Fitting a line with RANSAC[1] is a good starting point for figuring out the orientation and location of the different wires, and is used in many applications [12]. Projecting

[1]Random sample consensus

the point cloud into 2D space can also be a good approach as Cserép, Hudoba, and Vincellér showed [13].

Stagger detection

Stagger detection is carried out on the contact cable. For this, the contact cable has to be separated from the rest of the cables, droppers, and cantilevers. Geng et al. and Gutiérrez-Fernández et al. used DBSCAN[2] for clustering the different cables, from which the one lying lowest, horizontally between the rail pairs was selected as the contact cable [11], [14].

The stagger of the wire can then be calculated in relation to the centreline of the track gauge for each contact cable point.

3.1.2 Masts

Once the wires are separated, we are left with the points belonging to masts, cantilevers and droppers. Catenary masts are generally placed at a fixed distance apart from each other. Next to that, their cylinder-like shape and vertical orientation can be helpful for segmentation. Methods like RANSAC can be used for finding their corresponding points.

3.1.3 Cantilevers

The cantilevers can be found attached to the masts, connecting them with the two previously found wire point sets. Using a region-growing algorithm with the XY coordinates of the former, and the Z coordinate of the latter for the starting point can yield reliable results as [9] showed.

Segmenting droppers is often overlooked in publications. If we remove all previously segmented points, we should be left with the points mostly corresponding to the droppers. To eliminate false positives, these points can be projected to 2D space from both top-down and side views. In these projections they would be seen as evenly distributed points from the top view along the contact wire's line, and evenly distributed vertical lines from the side view. Combining these observations, we can easily construct 3D bounding boxes, containing the dropper points.

[2]Density-based spatial clustering of applications with noise

3.1.4 Conclusion

There are many different ways to reliably segment the different parts of railway catenary systems using traditional methods. Most of these techniques rely on the geometric relationship between different parts of the infrastructure, and also the geometrical properties of the objects themselves. With the spreading of artificial intelligence, most recent studies tend to discuss the potential of using machine learning-based methods, with less focus on further improving the accuracy of long-established techniques.

3.2 Artificial intelligence in railway LiDAR data processing

In these few sections, I am discussing the benefits and weaknesses of using such neural network-supported solutions and providing an overview of the most recent findings on this topic.

3.2.1 Benefits

The greatest benefit of artificial intelligence-based segmentation is that a network trained on different environments and data with different quality can successfully segment different inputs without the need for individually tweaking a set of parameters in theory [15]. Another natural benefit is that no complex algorithms have to be implemented, we only need labelled training sets and time to train and tweak the network to reach acceptable results. In many cases, artificial intelligence-based solutions are less computationally expensive than the classic computer vision-based ones.

3.2.2 Weaknesses

As already mentioned, labelled training data is essential for training these neural networks. Either fully or at least partly manual work is needed to label the point clouds, which is very tedious. The quality of the training data is very important, but quantity is almost as essential. As Figure 3.1 shows, this is even more crucial for deep learning methods: traditional machine learning plateaus with millions more

training instances, while deep networks can continue to learn more. With more general problems availability of public labelled training sets is not an issue, however, railway infrastructure is still being treated as sort of a national secret amongst most countries. Thus, there are little to no publicly available labelled railway point clouds at the time of writing these lines.

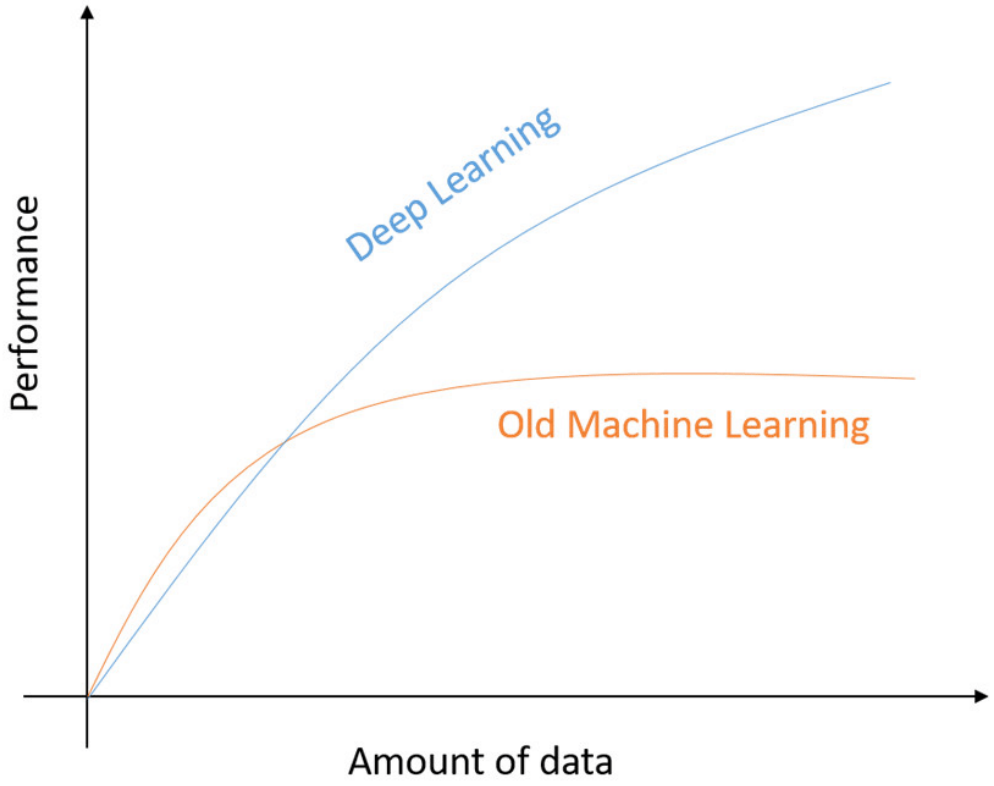


Figure 3.1: *The effect of the amount of training data on old machine learning and deep learning performance.*[3]

3.2.3 Recent findings

In one of the most recent studies Grandio et al. achieved 100% precision for segmenting rails and 98.9% for cables on 32768 input points, after 337 minutes of training [15]. In this paper, the PointNet++ and KPConv architectures were found to yield similar performance. Note that in this study the training data was pre-processed, removing most of the surrounding terrain around the rails. When feeding the network a sparser point cloud also containing the surroundings, the network

[3]From Alom et al.: "A State-of-the-Art Survey on Deep Learning Theory and Architectures", 2019 [16]

struggled to provide usable results (see Figure 3.2), highlighting the importance of more varied training data.

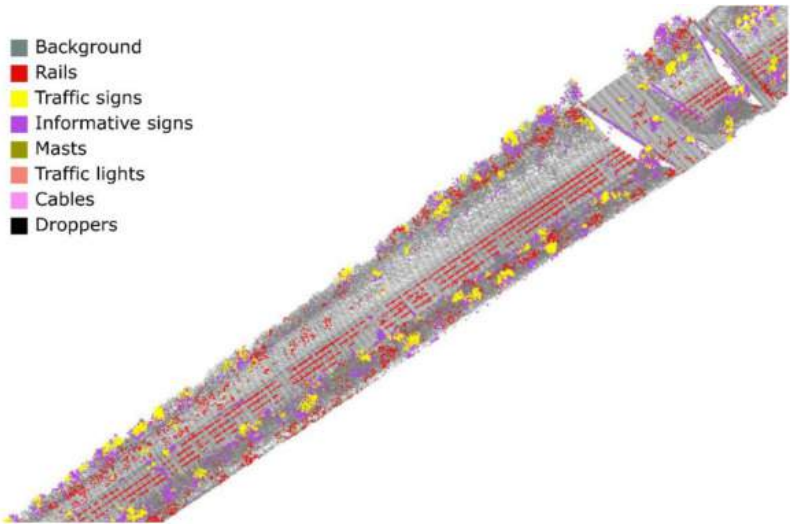


Figure 3.2: *Feeding unseen data to the network can lead to weak results, like ground points getting falsely classified as rails*[4]

Manier et al. tried to take advantage of specific characteristics of the railway scenes, specifically the strong vertical component of objects of interest [17]. An axially symmetrical transformation was defined by them to conserve relative elevation information. With this addition, a mean IoU[5] of 83.4% was achieved on their dataset of the Paris-Lyon high-speed line, compared to the ConvPoint (78.1%) and ShellNet (66.1%) methods.

Yu et al. focused on segmenting rails, and tried to propose a neural network solution which can be used in real-time [18]. As Pointnet and RandLA-Net only reached a maximum of around 83% IoU, they hand-crafted their own network architecture, which was somewhat slower, but still able to process data in real-time and also reach an IoU of 95.14%.

As discussed earlier, one of the main drawbacks of artificial intelligence methods is the need for labelled datasets. In the paper by Guinard et al., an attempt was made to address this by creating a weakly supervised method for speeding up the pointwise classification of LiDAR acquisitions [19]. They reached an overall F1-score[6] of 96.13%, however, for this, the point clouds had to be pre-segmented manually.

[4]From Grandio et al.: "Point cloud semantic segmentation of complex railway environments using deep learning", 2022 [15]

[5]Intersection over Union, an evaluation metric

[6]an evaluation metric that measures accuracy, mainly used in machine learning

3.2.4 Conclusion

Neural network-based solutions have gotten quicker, more advanced and more accurate over the last few years. Public training data is still virtually unavailable, and after having reached out to multiple researchers, it seems that the national railways are not willing to share their acquisitions publicly. There is research in the direction of automating the creation of such labelled datasets [19], which can drive more interest towards neural network-based methods in the field of railway point cloud segmentation tasks.

Chapter 4

Datasets

In this chapter, I will introduce the datasets which were used during developing and testing my algorithms. One of them was recorded in Central Hungary between **Szabadszállás and Csengőd** stations, the other in Western Hungary, around **Szentgotthárd** station.

4.1 Data recording

The two datasets were recorded with the same method. This consisted of loading a car with an MLS system on an open wagon, which was then pulled along the rails by a locomotive at a maximum of 60 km/h speed, illustrated by Figure 4.1. The MLS system used was a Riegl VMX-450 MMS 3.1 unit, which apart from supplying LiDAR data was also taking pictures every five metres. These are mainly used to supply colour information for the 3D point cloud. The RGB attributes were not used in my work, as other systems might not provide them. The sensor uses two spinning mirror laser scanners for mapping out the 3D point clouds, recording 1.1 million points every second, operating at 12000 RPM.

Figure 4.1: *The setup used for collecting the data*

4.2 Szabadszállás – Csengőd dataset

This dataset was provided by MÁV, the Hungarian State Railways. It was recorded in 2016 between the stations of Szabadszállás and Csengőd on railway line number 150 (see Figure 4.2), which is undergoing complete reconstruction at the time of writing this thesis.

The dataset contains 18.5 kilometres of railway infrastructure with rural surroundings, extending to about 65 metres orthogonally in both directions from the centre of the electrified single-track railway line. The section does not contain bridges or tunnels.

Figure 4.2: *Covered area of the Szabadszállás – Csengőd dataset*

In total, there are over 1.5×10^9 points in the dense point cloud, which has an average precision of 3 mm and a maximal error of 7 mm. The points are georeferenced in the hungarian EOV[1] coordinate system with a positional accuracy of 3–5 centimetres.

[1]short for uniform national projection (egységes országos vetület)

4.2.1 Sample: MÁV-simple

The 100 metres long section contains around 7.3 million points. It is cropped between the EOV Y coordinates of 159100 and 159200. It contains two cylindrical masts to detect, and no interfering trackside objects (see Figure 4.3).

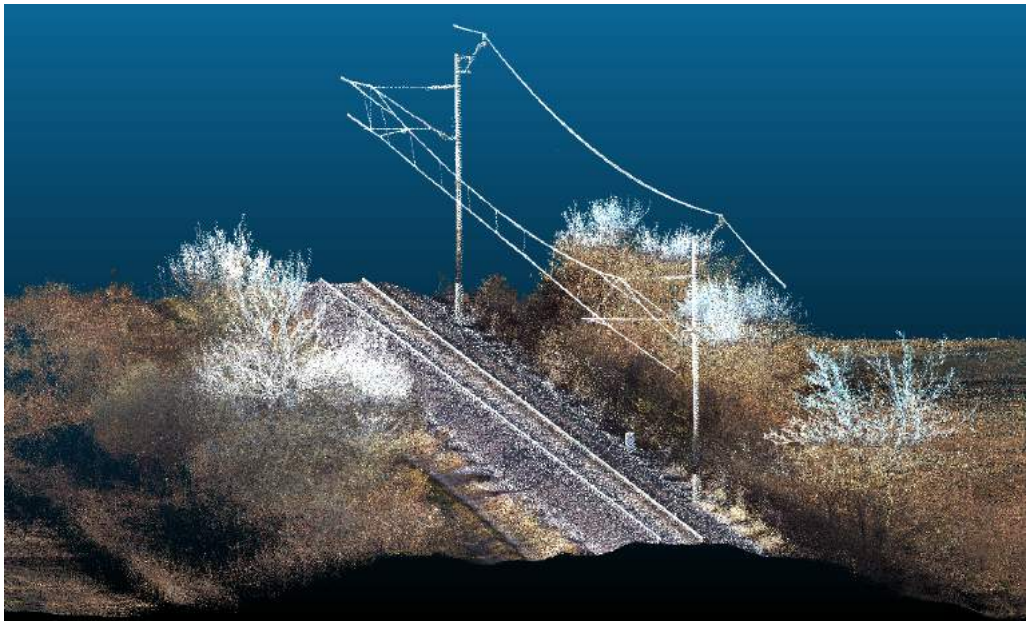


Figure 4.3: *The **MÁV-simple** sample visualized in 3D*

4.2.2 Sample: MÁV-long

The 560 metres long section contains around 40.3 million points. It is cropped between the EOV coordinates of (154707,666893) and (155266,667988). It contains eight cylindrical masts to detect and also has multiple interfering mast-shaped objects near the tracks (see Figure 4.4).

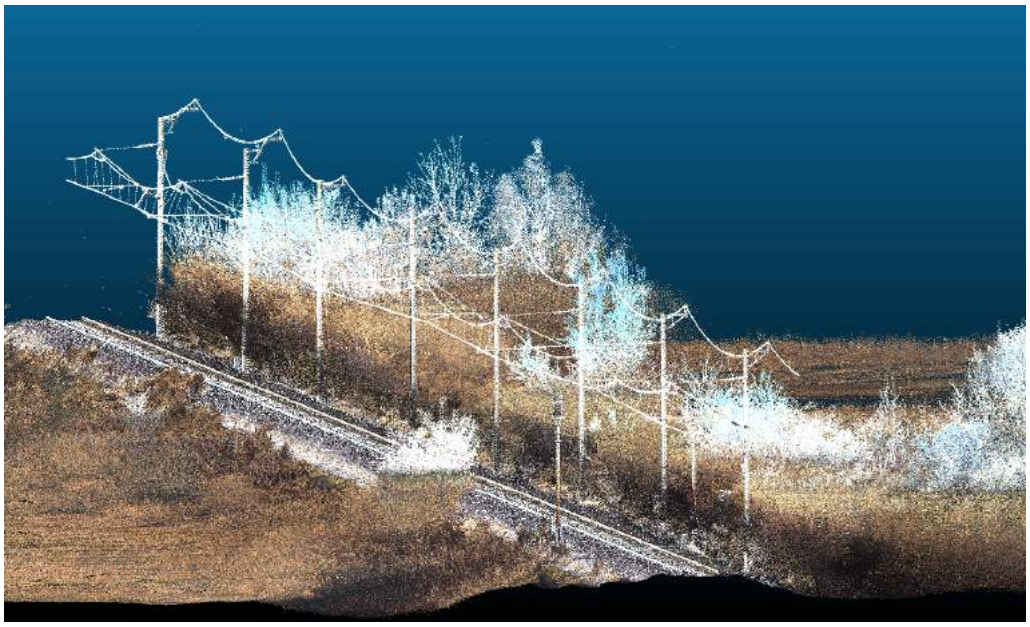
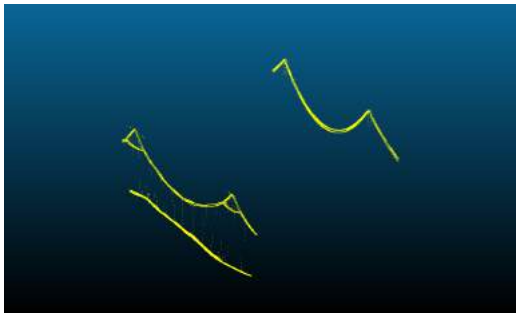


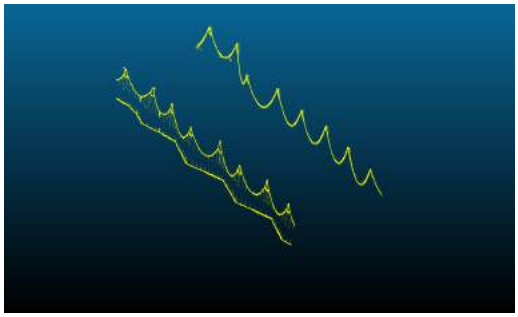
Figure 4.4: *The **MÁV-long** sample visualized in 3D*

4.2.3 Cable stagger detection and cantilever segmentation samples

As there is little variation in the structure of cables and cantilevers between the **Szabadszállás – Csengőd** and **Szentgotthárd** datasets, only samples from the first were used to evaluate these algorithms. Thus, the cantilever detection was run on the above two samples, and for contact cable stagger detection the already detected cables [12] from these same samples were used (see Figure 4.5).



(a) *Detected cables from the **MÁV-simple** sample*



(b) *Detected cables from the **MÁV-long** sample*

Figure 4.5: *Cable samples from the **Szabadszállás – Csengőd dataset***

4.3 Szentgotthárd dataset

This dataset was provided by GYSEV/Raaberbahn which is the second-largest railway company and railway infrastructure owner in Hungary. It was recorded in 2017 around Szentgotthárd station (see Figure 4.6) and contains mostly urban surroundings around the electrified single-track line. It contains over 800 million points.

The dataset was recorded on a 4.7 kilometres long section in a width of about 90 metres. The section does not contain bridges or tunnels.

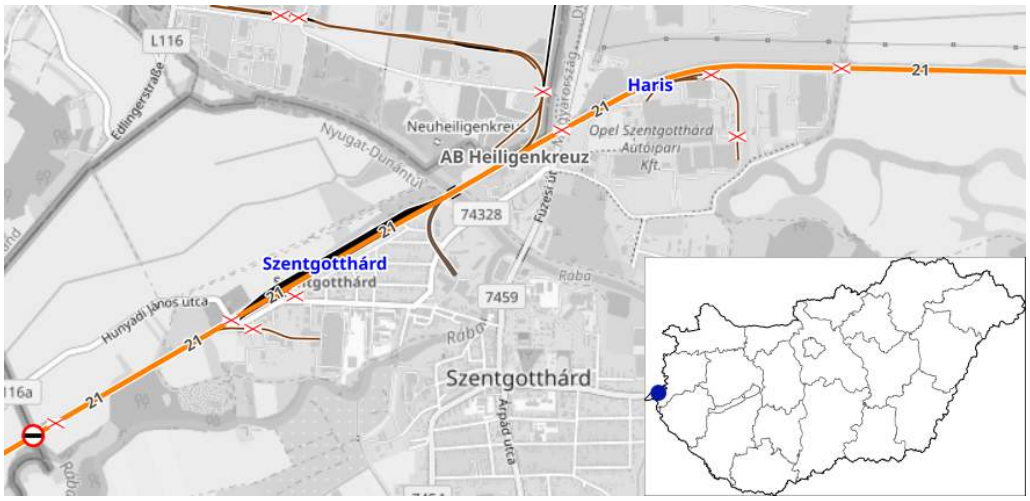


Figure 4.6: *Covered area of the Szentgotthárd dataset*

4.3.1 Sample: GYSEV-standard

The 185 metres long section contains around 7.4 million points. It is cropped between the EOV coordinates of (183790,440290) and (183845,440478). It contains three cylindrical masts to detect and also some interfering mast-shaped objects near the tracks (see Figure 4.7).

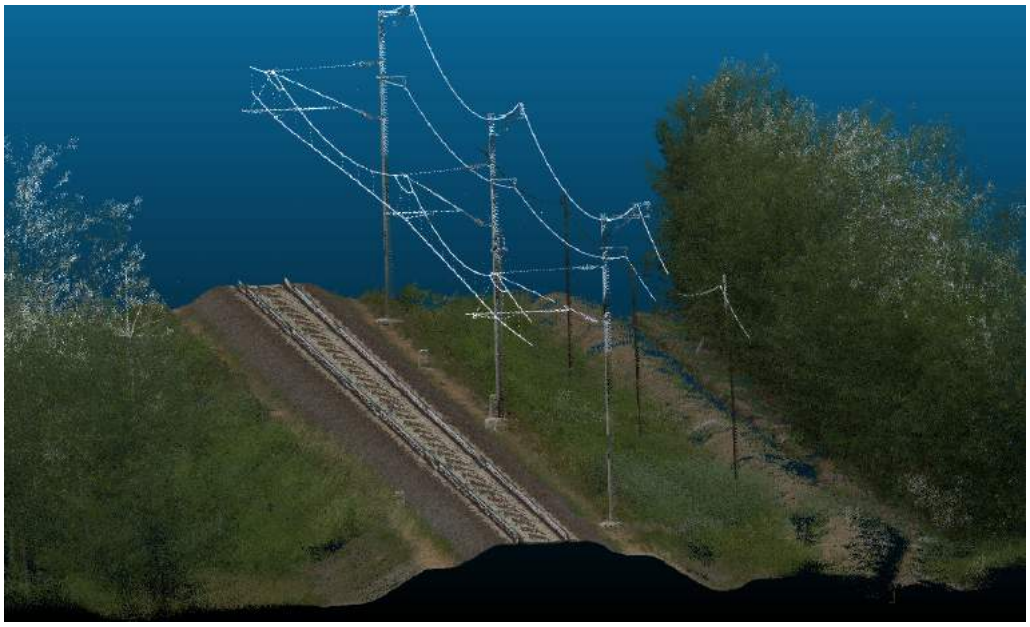


Figure 4.7: *The **GYSEV-standard** sample visualized in 3D*

4.3.2 Sample: GYSEV-mixed

The 188 metres long section contains around 7.3 million points. It is cropped between the EOV coordinates of (183765,440619) and (183850,440807). It contains three masts to detect (two lattice, one cylindrical), and no interfering trackside objects (see Figure 4.8).

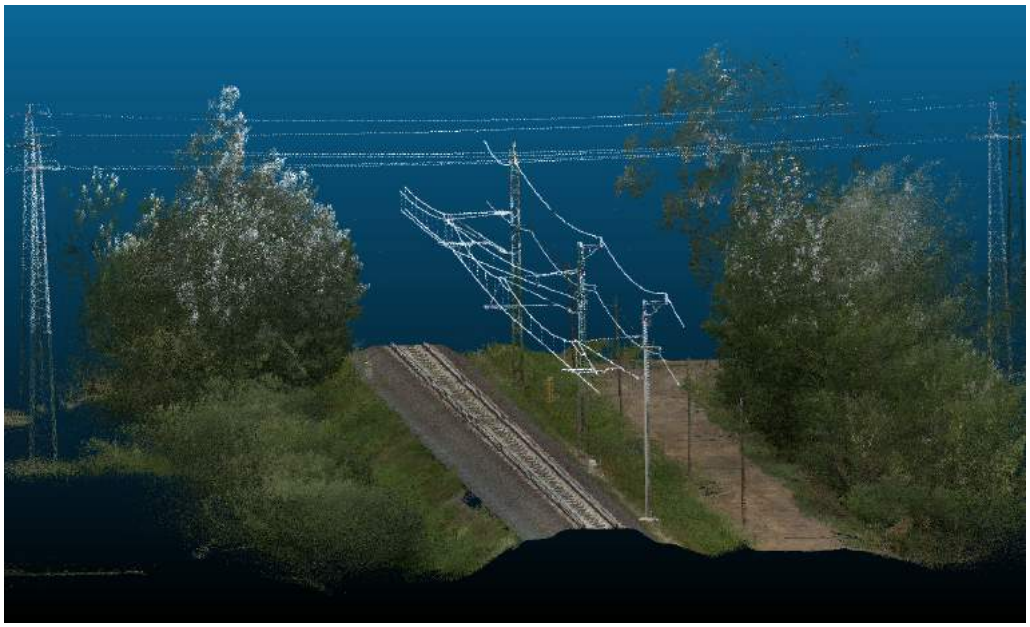


Figure 4.8: *The **GYSEV-mixed** sample visualized in 3D*

Chapter 5

Methodology

In this chapter, I will introduce the three algorithms I developed for segmenting masts, cantilevers and finally for detecting contact wire stagger above operational limits. The assumption was made that the inputs of these methods are preprocessed in the sense that they contain straight rail track segments. In the case of curved sections, they can be cut at the detected curves as shown by Tábori yielding multiple, virtually straight sections [20].

5.1 Mast segmentation

In most publications, the detection of masts is done by using pattern-matching [9] or voxel-based methods [11]. My proposed method is different in that it utilises certain characteristics of the railway point clouds to segment the points belonging to masts. I focused on accurately segmenting cylindrical masts, as most of the masts found in the two datasets were of this type. For the lattice masts a fallback mechanism was implemented, which still provides usable results.
The method is divisible into three main parts:

1. Removing unnecessary points.
2. Finding possible mast centroids.
3. Segmenting masts.

Figure 5.1 shows the algorithm's flowchart.

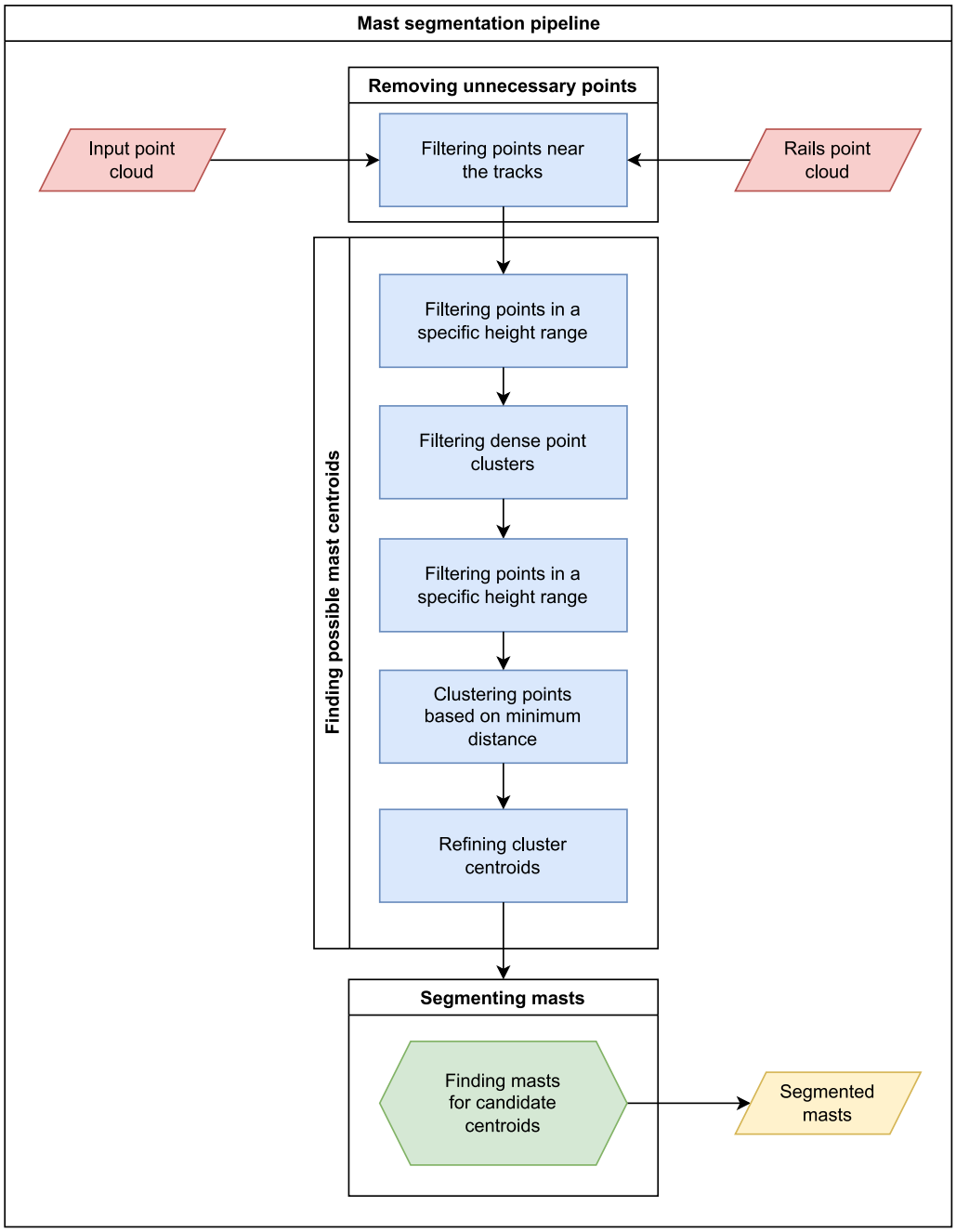


Figure 5.1: *Overview of the mast segmentation pipeline*

5.1.1 Removing unnecessary points

The catenary masts lie within a few metres distance from the tracks, so a large portion of the input data outside this region can be disregarded. For this, the already detected rails [21] are used as a reference. Only the points within a maximum 2.2 metres distance on each side of the tracks are kept and passed on to the next stage of processing. The steps required for this are shown by Algorithm 1 [12].

Algorithm 1 Width filtering with seed

1. Fit a line on seed cloud S with RANSAC.
2. Calculate the angle α the fitted line closes with the Y-axis.
3. Create copies of input P and S clouds as Q and S_{rot}.
4. Rotate both Q and S_{rot} on the Z-axis by α.
5. Find minimum and maximum values x_{min}, x_{max} on the X-axis of S_{rot}.
6. Remove points p_i from P where:
   $q_i \in Q : q_i.x < x_{min} - \delta$ *or* $q_i.x > x_{max} + \delta$.
7. Return P.

The remaining points after running width filtering on sample **MÁV-long** are visible in Figure 5.2.

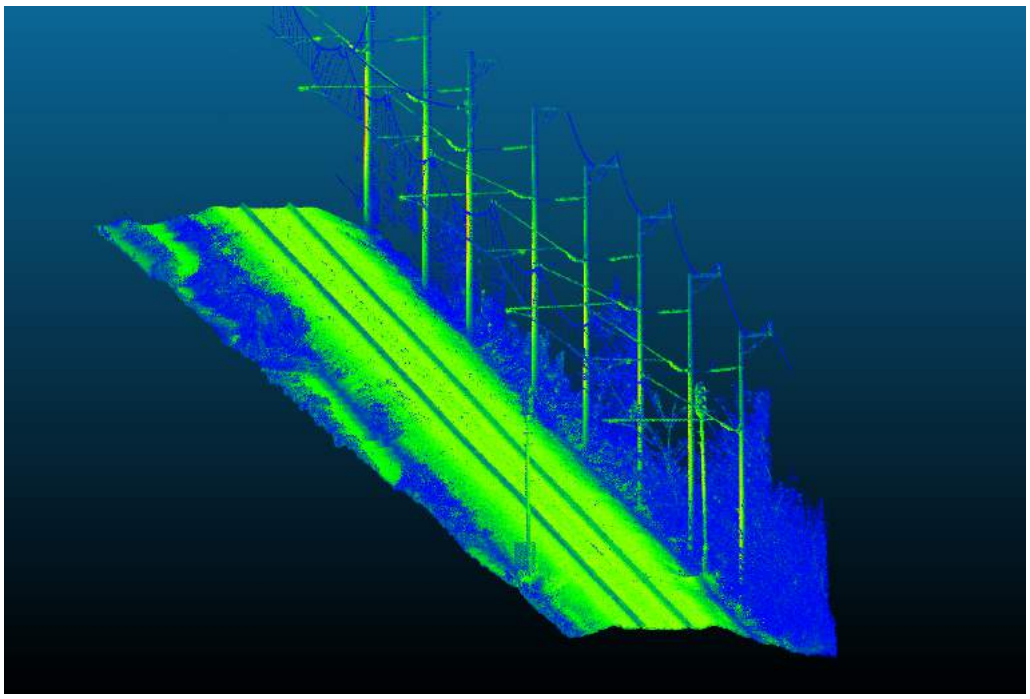


Figure 5.2: *Point cloud after width filtering*

5.1.2 Finding possible mast centroids

One of the unique characteristics of the input data utilisable for finding potential mast points is the distribution of points based on their density. Point density is a metric that can be calculated for a point by counting how many neighbouring points we can find within a sphere with a given radius centred at our point. Equation 5.1

shows the calculation of this property for point p in the set of points $P \subset \mathbb{R}^3$ with δ being the radius. Function $\delta(p, q)$ is the Euclidean distance formula ($\delta : P\ x\ P \to \mathbb{R}$).

$$density(p) = |\ \{\ \delta(p, q) < \gamma\ \}\ |\ \ where\ p, q \in P,\ p \neq q\ and\ \gamma \in \mathbb{R} \tag{5.1}$$

Figure 5.3 shows sample **MÁV-simple** coloured by the calculated density values. Red and orange points have a dense, while blue ones have a sparse neighbourhood.

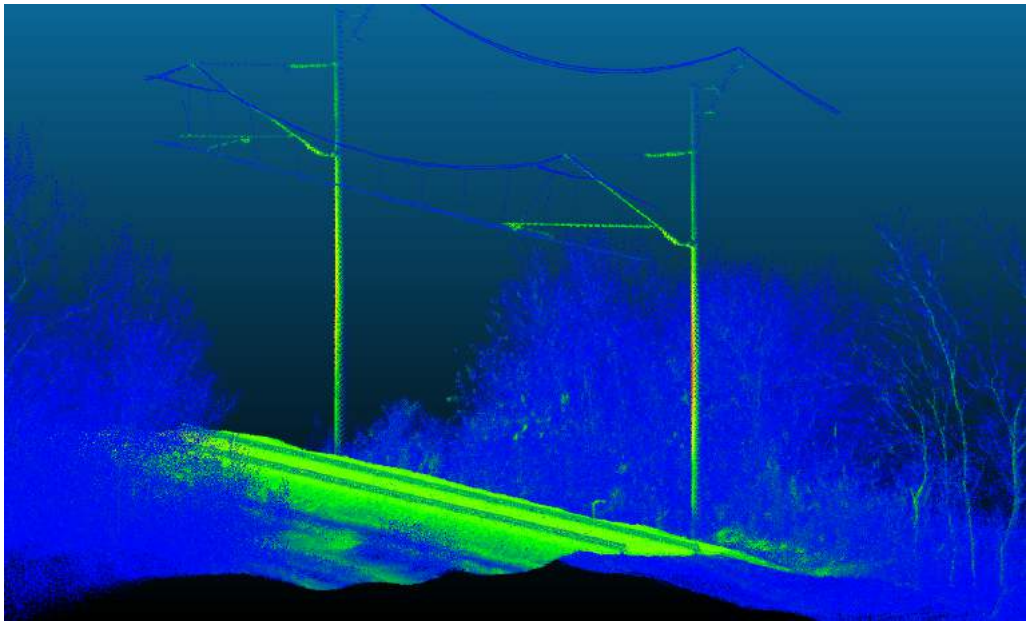


Figure 5.3: *Point cloud coloured by density values*

As visible from the figure, points near the vertical centre of the masts have a high density value, while points belonging to vegetation and other points out of interest have a low density value. Thus, by filtering points above a certain density value, we can find points belonging to masts. This filtering is shown by Equation 5.2, where λ is the minimum number of neighbours to keep points with. Calculating density values is a computationally demanding task, with an $O(n^2)$ time complexity when implemented on non-optimal spatial data structures.

$$Q = \{\ p \in P :\ density(p) > \lambda\ \}\ where\ \lambda \in \mathbb{N} \tag{5.2}$$

Thankfully, it is known that the points we are looking for lie above the tracks. Thus, taking a vertical slice a few metres above the tracks, we get a much smaller set of points for which the density values can be calculated significantly faster. This bandpass filtering method is shown by Equation 5.3, where S contains the sliced points which lie within a δ vertical distance from the *midpoint*. After evaluating

multiple sample inputs, the values 0.5 metres and the level of the rails plus 3 metres were set for δ and *midpoint* accordingly.

$$S = \{\, p \in P : \; midpoint - \delta < p.z < midpoint + \delta \} \; where \; midpoint, \delta \in \mathbb{R} \quad (5.3)$$

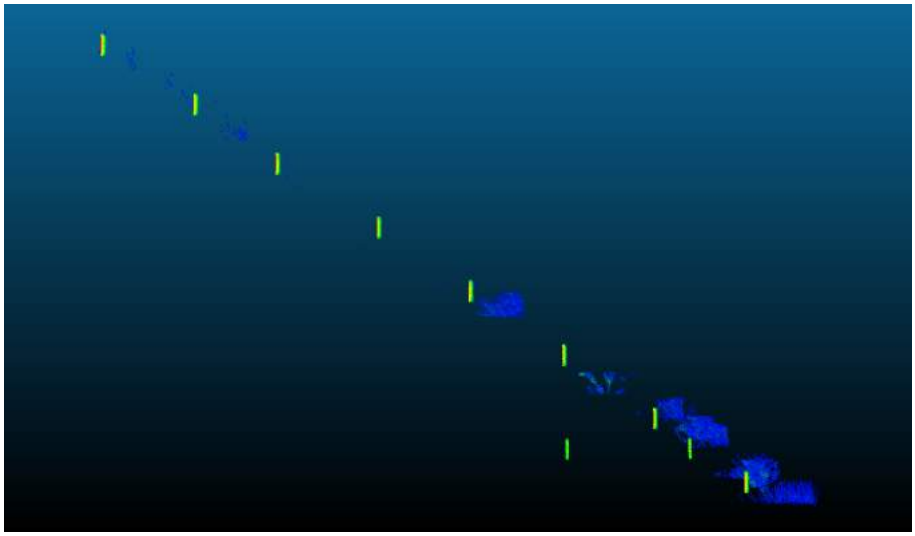
The result of running the bandpass filter after width filtering on sample **MÁV-long** is visible in Figure 5.4a.

Finding the optimal radius and neighbour count for the density filtering is not a straightforward process. A radius of 0.16 metres and a minimum neighbour count of 150 points were found through trial and error. With these values, only the points belonging to mast-like trackside objects are kept. Next to the desired catenary masts, this also includes signal masts (Figure 5.4b, **MÁV-long** sample), which will need to be eliminated later.

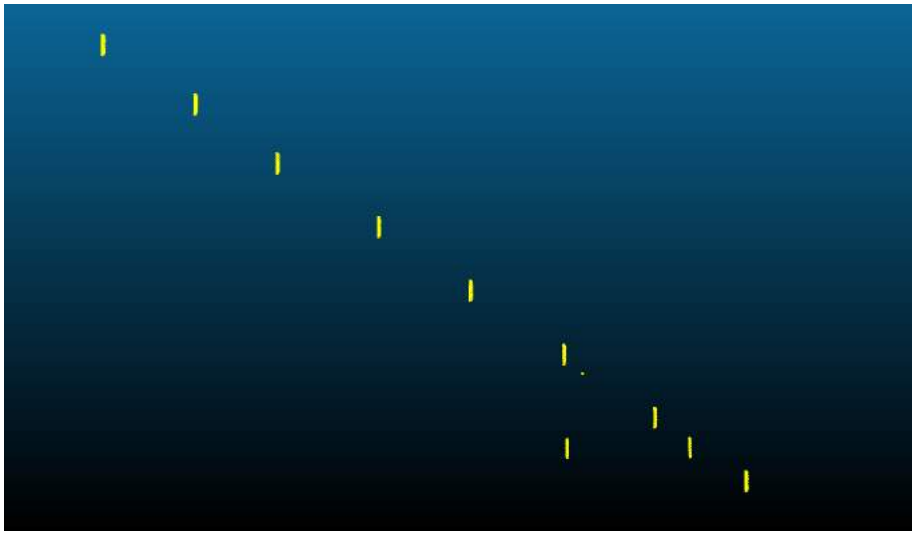
To speed up processing time, a second bandpass filter pass is run on these points. This only keeps points within Z values of $+/-$ 0.1 metres of the midpoint, reducing the number of points by 80%. The result of running this second pass is visible in Figure 5.4c.

The next step is to cluster points belonging to different masts. For this, we can then group them into clusters based on their minimum Euclidean distance from each other. Using a Kd-tree[1] structure this clustering can be carried out as shown by Algorithm 2 [22].

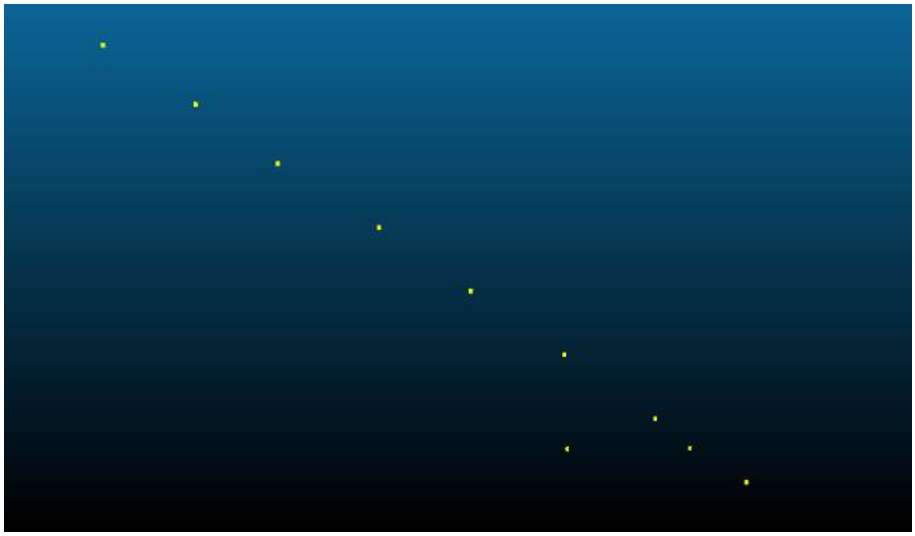
[1] k-dimensional tree, a space-partitioning data structure



(a) *Result of first bandpass filter*



(b) *Density filter*



(c) *Second bandpass filter*

Figure 5.4: *The first three steps of finding possible mast centroids*

Algorithm 2 Euclidean clustering

1. Create a Kd-tree representation for the input point cloud dataset P.
2. Set up an empty list of clusters C, and a queue of the points that need to be checked Q.
3. Then for every point $p_i \in P$, perform the following steps:
   - Add p_i to the current queue Q.
   - For every point $p_i \in Q$ do:
     - Search for the set P_i^k of point neighbors of p_i in a sphere with radius $r < d_{th}$.
     - For every neighbor $p_i^k \in P_i^k$, check if the point has already been processed, and if not add it to Q.
   - When the list of all points in Q has been processed, add Q to the list of clusters C, and reset Q to an empty list.
4. The algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point clusters C.

By calculating the centroids of these clusters, we then get the centroids of possible masts. These centroids however do not accurately reflect the true centroids of the masts, as some mast points with lower density values might not have been taken into account when calculating them. For this reason, the centroids are being recalculated for the points within a cylindrical bounding box of the original centroids. These bounding boxes, which are centred on the original centroids, have a 0.5 metres radius and a height of 0.2 metres. These values were found to be sufficient for enclosing all possible mast points in the height level used without introducing points from nearby vegetation and other objects. Figure 5.5 illustrates this from a top-down perspective. The original centroid (calculated from dense points in yellow) is shown in orange, while the refined centroid (recalculated from the dense, and additionally the light blue points) is coloured green.

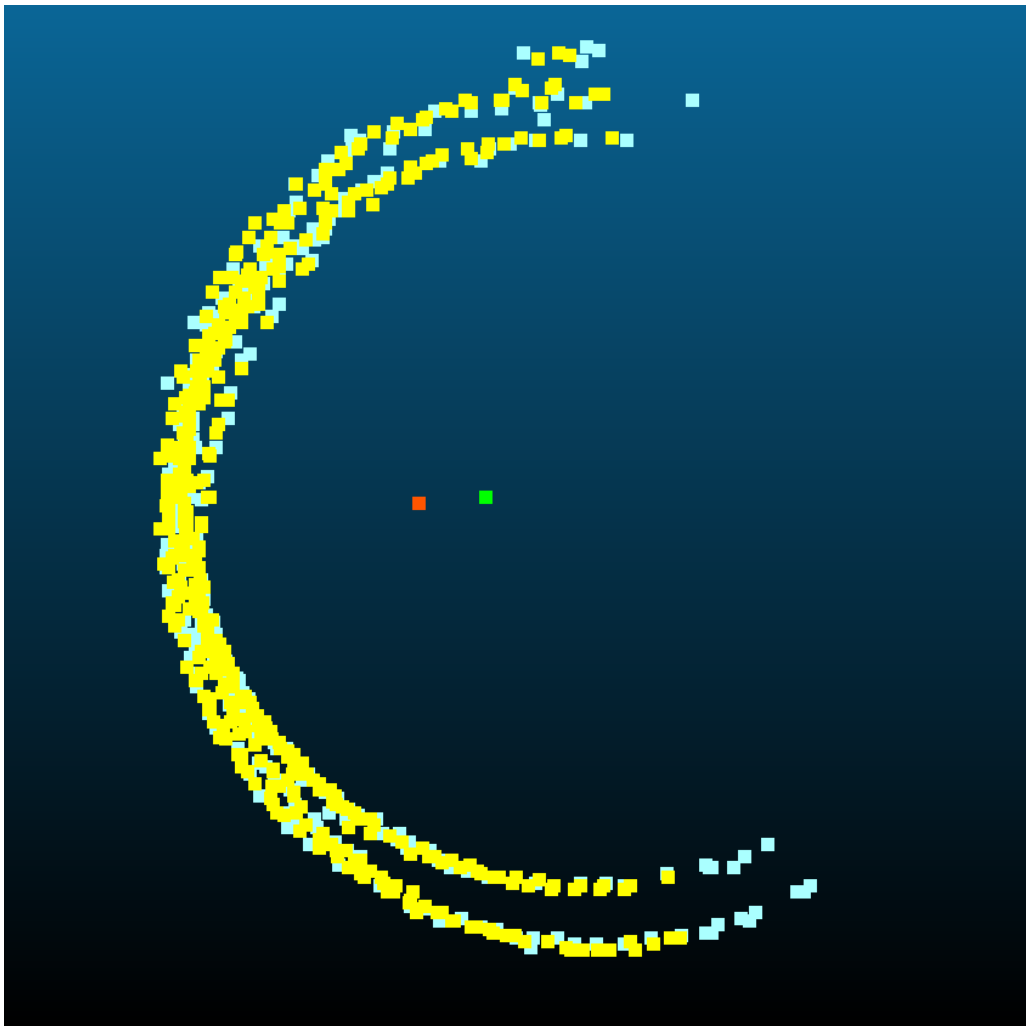


Figure 5.5: *Centroid recalculation*

5.1.3 Segmenting masts

After the candidate mast centroids have been found, the final step is to check if there is a valid mast object centred around them, and if so, their corresponding points have to be filtered. Figure 5.6 shows how these steps are carried out.

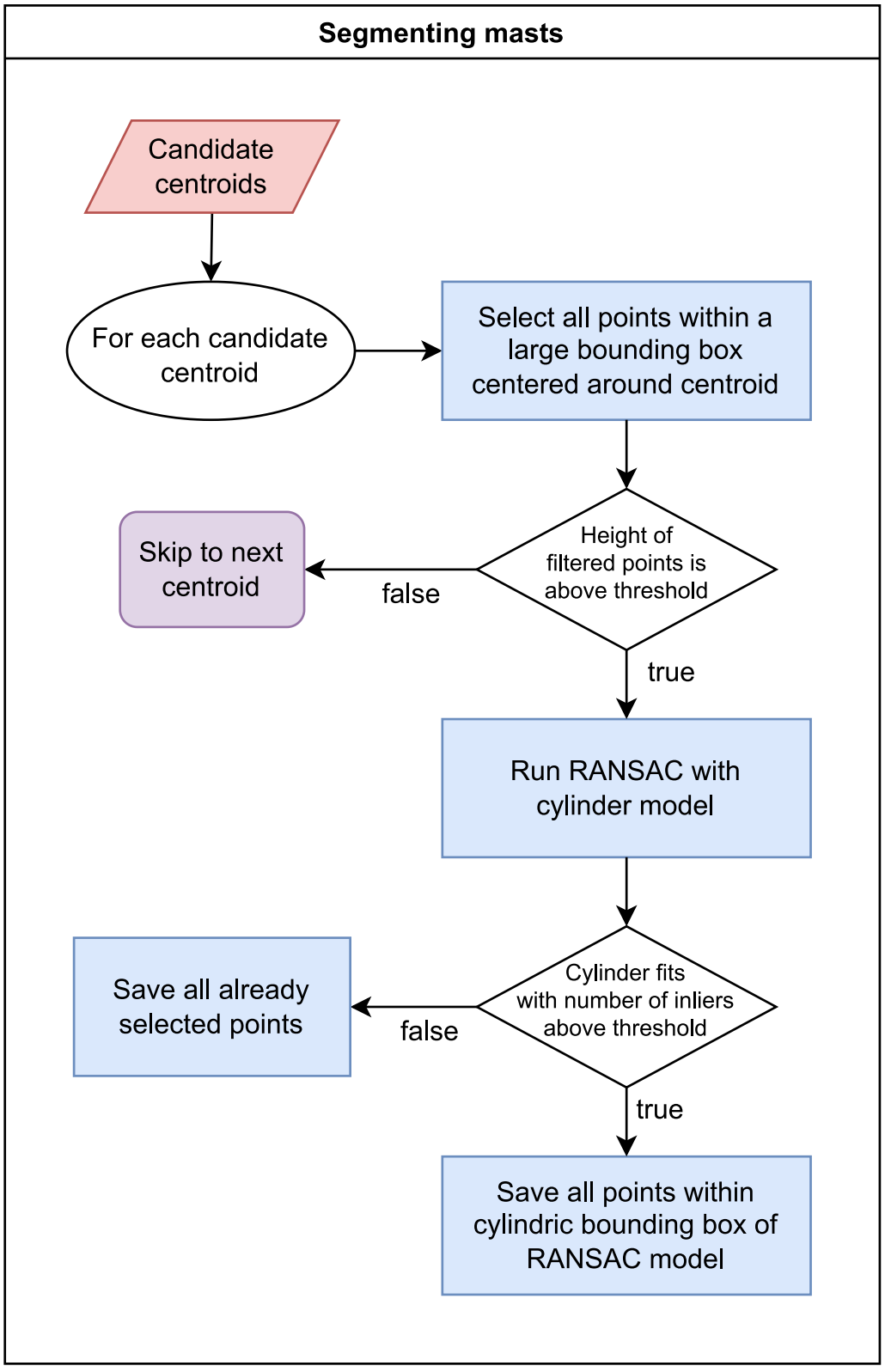
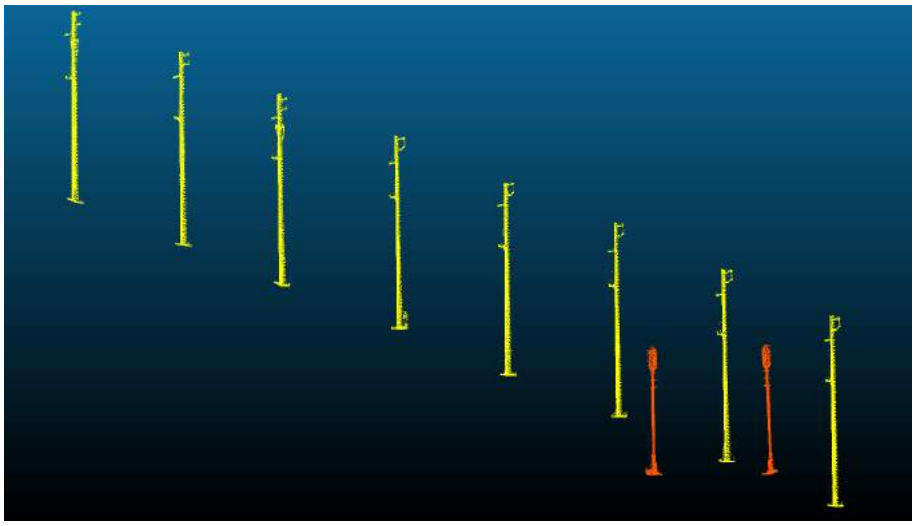


Figure 5.6: *The actual mast segmentation algorithm*

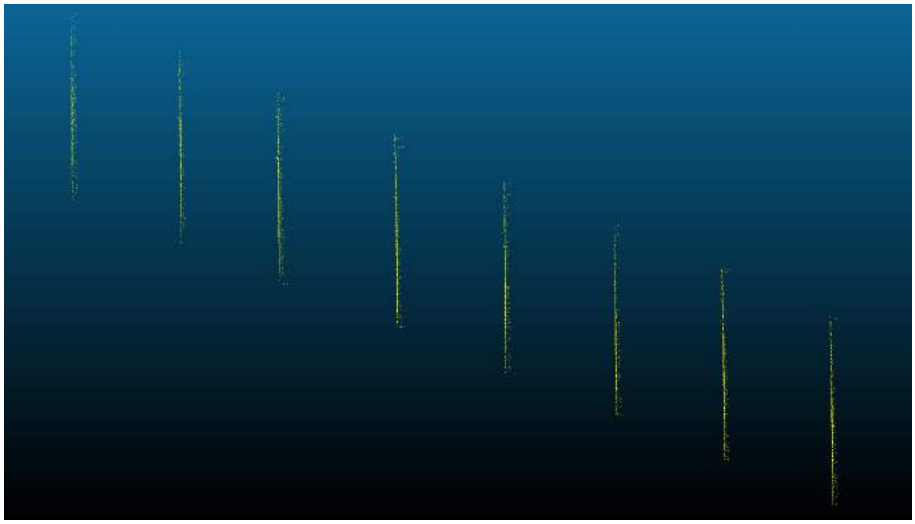
For each centroid point, the points within a large cylindrical bounding box are selected first. This bounding box horizontally centred on the centroid has a radius of 0.4 metres. Its height is 13 metres, reaching from 5 metres below to 8 metres

above the Z value of the centroid. The points after this step are demonstrated by Figure 5.7a in the case of the **MÁV-long** sample. Next, the height of the points inside this bounding box is calculated by taking the difference between the largest and smallest point values on the Z-axis. If this height value is below 8 metres, the algorithm moves on to the next candidate centroid, as the catenary masts have a general height of 9–10 metres. An example of this behaviour is visible in Figure 5.7a, with the points to disregard belonging to non-catenary mast objects shown in orange.

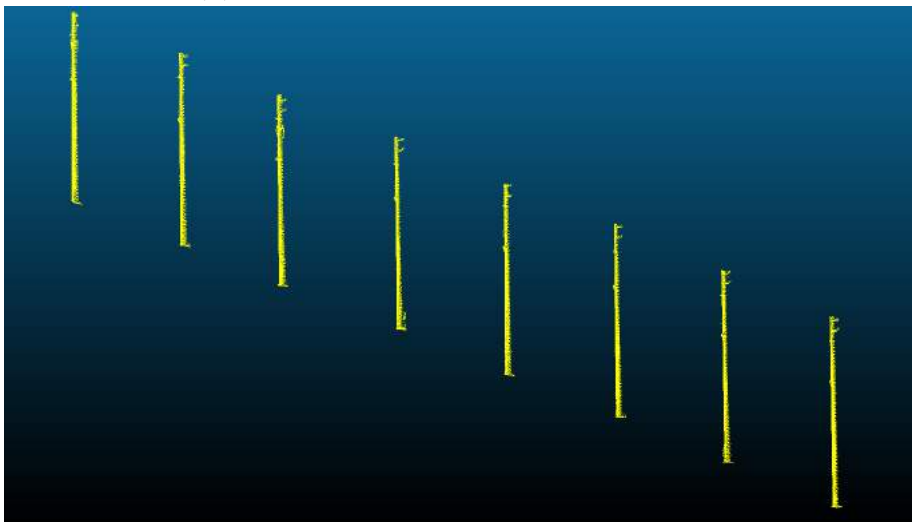
If the height value is above 8 metres, the next step of processing is fitting a cylinder to the selected points with the RANSAC algorithm. If we don't find a cylinder fitting the model, all the previously selected points are saved for the currently processed centroid as masts points. If a fitting cylinder is found, the points inside the cylindrical bounding box constructed with the model's coefficients are saved as mast points for the currently processed centroid (see Figure 5.7c). This extra step is necessary because of the slightly tapered nature of the cylindrical masts, due to which RANSAC is not able to correctly classify all the mast points as inliers, shown by Figure 5.7b.



(a) *Filtering points within large cylindrical bounding boxes of centroids*



(b) *Inliers after RANSAC cylinder fitting*



(c) *Filtering points with coefficients of fitted cylinder*

Figure 5.7: *Finding the masts points for the centroids*

5.2 Cantilever segmentation

For segmenting the cantilevers, the already detected wires and masts are supplied to the pipeline next to the input point cloud. Figure 5.8 shows the general overview of the pipeline.

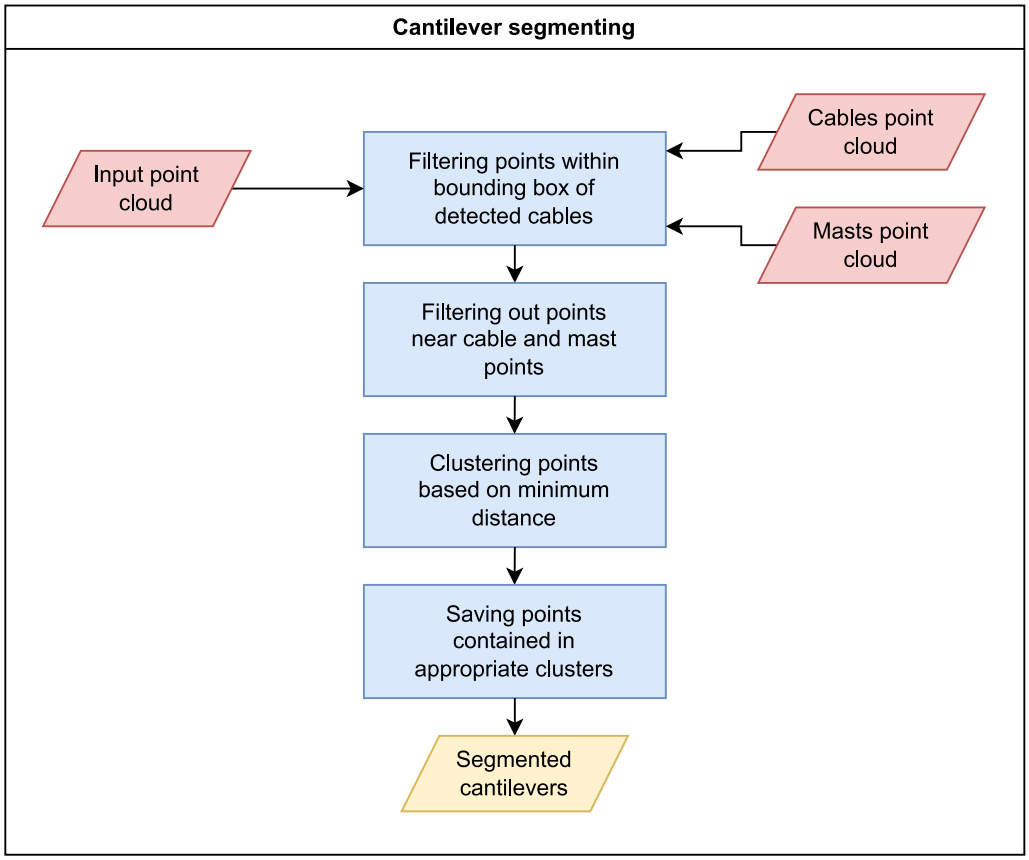


Figure 5.8: *Cantilever segmentation pipeline*

First, the points lying outside of the region where the cantilevers are expected to be found are removed from the data. This is done in two passes, first in the horizontal, then in the vertical direction. For the horizontal pass, the previously introduced Algorithm 1 (Page 34) is used with the already detected wires supplied as a seed. For the vertical pass all points above the highest, and below the lowest wire points are removed. The result of these steps is shown in Figure 5.9a, on sample **MÁV-simple**.

After these steps, we are left with points in the region of interest, but next to the cantilever points, all the points belonging to wires and some belonging to the masts are also contained in this area. Thus, the next step is to remove said points. For this, all the points within a small radius of the already detected mast- and cable points

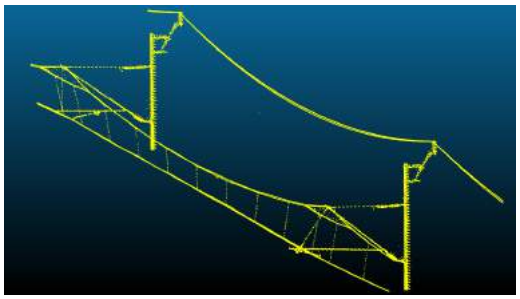
are removed (see Figure 5.9b). This radius is 8 and 5 centimetres for the mast- and cable points respectively. To speed up this process an Octree[2] representation of the points is used. This process is shown by Algorithm 3.

Algorithm 3 Cantilever filter

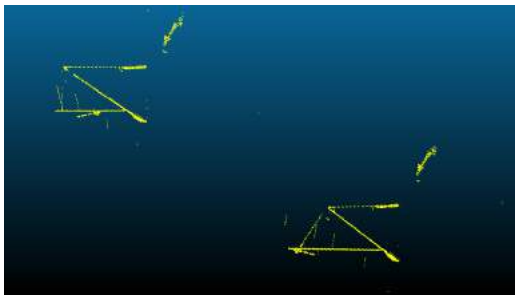
1. Construct Octree representation O of input cloud P.
2. Initialize empty cloud *Result*.
3. For both cable and pole seed clouds S do:
   - Initialize empty set of indexes I to be removed.
   - For each point $s \in S$ do:
     - Using O find all i for which $p_i \in P$ falls within δ radius of s.
     - Add all i to I.
   - Add all $p_i \in P$ to *Result* where $i \notin I$.
4. Return *Result*.

Next, the remaining points are clustered using the previously introduced Euclidean clustering algorithm (Algorithm 2). This clustering is being executed on the remaining points, but the Kd-tree used for searching nearby points is built from a voxel grid[3] of the points with a leaf-size of 35 centimetres. The points of the constructed voxel grid are shown in Figure 5.9c. The radius is set at 4 centimetres for this clustering step, after which we get all the points of individual cantilevers contained in a different cluster, along with several additional clusters containing irrelevant points.

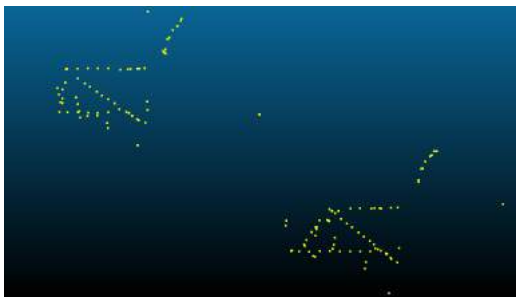
[2] a tree data structure where each internal node has exactly eight children
[3] a sparse 3D spatial data structure



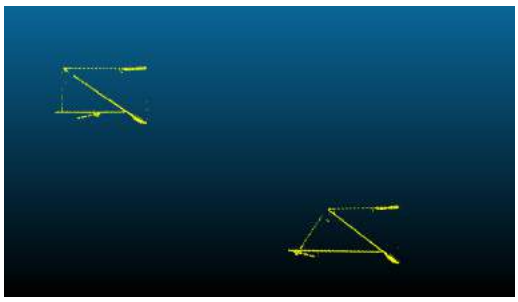
(a) *Filtering points within the range of interest*



(b) *Removing points near cables and masts*



(c) *The constructed voxel grid used for clustering*



(d) *Result after clustering and selecting appropriate clusters*

Figure 5.9: *Finding the cantilever points*

The final step is to select the appropriate clusters and save the points contained by them as cantilever points. To accomplish this, only the clusters containing a reasonable number of points are selected for further analysis. This threshold was set at 2000 points after evaluating the different sample inputs. Then, for each cluster passing this condition PCA[4] is used to determine the orientation of the points. The points in clusters oriented orthogonally to the cables are saved as cantilever points, as seen in Figure 5.9d.

5.3 Contact cable stagger checking

For contact cable stagger checking, the already detected cables [12] and additionally the already detected rail tracks [21] are used. Figure 5.10 shows the outline of this pipeline.

[4]principal component analysis

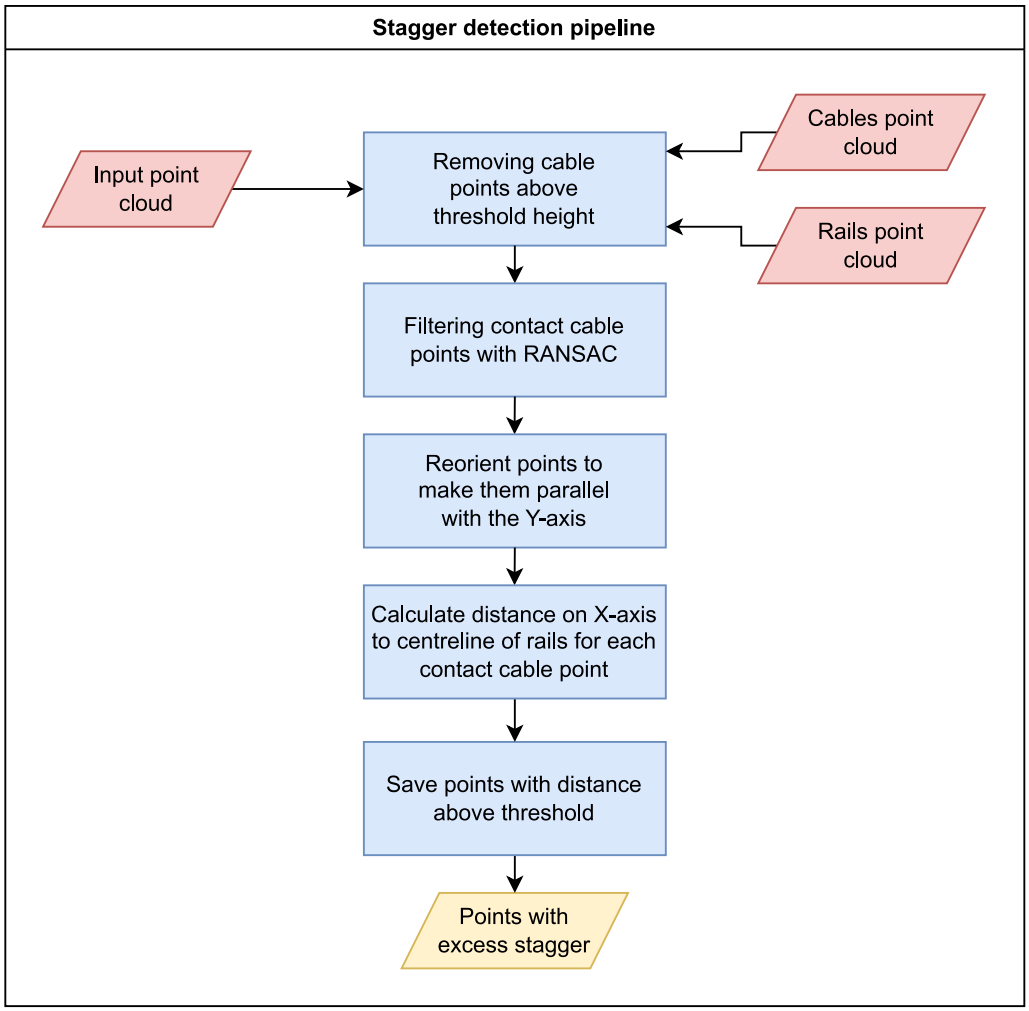


Figure 5.10: *Stagger checking pipeline*

First, the contact cable has to be filtered from the rest of the cables. For this, all the points 0.18 metres or above the lowest point of the cable input cloud are removed. After this step, the RANSAC algorithm is used to fit multiple straight lines to the horizontally staggering contact cable, eliminating potential noise like falsely detected cable points belonging to cantilevers or droppers. The result of these steps is demonstrated by Figure 5.11 on the **MÁV-long** sample.

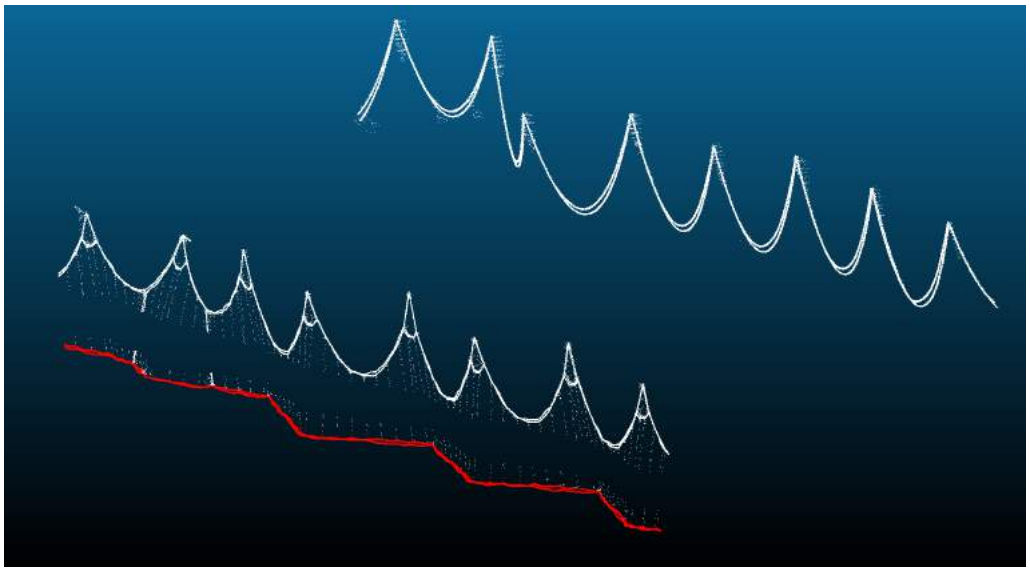


Figure 5.11: *The filtered contact cable points (red) with the rest of the input points*

The stagger of the contact cable is checked relative to the centreline of the rails. To make this easier, both the contact cable and the rails are rotated parallel to the Y-axis with the help of PCA. This way the centreline can be computed much easier, by calculating the mean of the minimum and maximum X coordinates of the rail points.

Next, the distance of each point from the centreline is calculated on the X-axis. Then, the below steps follow:

- If this distance is larger than a prescribed stagger value plus a threshold value, the point is saved as an excessively staggering point.
- If this distance is smaller than a minimum threshold, the point is saved as a point which lacks stagger.

The prescribed stagger and threshold values differ based on the year of construction and the class of the railway line as discussed in Chapter 2, Page 9. Thus, these values can be set as a parameter of the pipeline, as well as the threshold for lack of stagger. After looping through all contact cable points, the ratio of points saved as lacking stagger to all the contact cable points is calculated. If this is larger than 45%, a warning is issued indicating a potential lack of stagger. Next to this, the excessively staggering points — if any — get saved as the output of the pipeline.

Chapter 6

Implementation

In this chapter, I will provide some details about the implementation of my solution.

6.1 Code Availability

The source code is attached next to the thesis. It is also publicly available online at the `https://github.com/GISLab-ELTE/railroad/` repository.

6.2 About the railroad framework

My algorithms were implemented inside the Robust Railroad Infrastructure Detection Framework, or railroad tool for short. This open-source framework was built by Cserép, Hudoba, and Vincellér to extract various railroad infrastructure from dense LiDAR point clouds, with a primary focus on cable and railway track detection [13].

It is written in the C++ programming language and mainly relies on the PCL[1] and OpenCV libraries for extracting important data from the input point clouds.

In the framework, the various algorithms are defined as filters, arranged in a pipeline. The first filter in line gets the supplied input point cloud as an input. Then the filters get executed one after the other, passing only the filtered points to the next filter in line.

[1] Point Cloud Library

6.3 New and modified filters

Next to using and modifying some already implemented filters to use in my processing pipelines, I also implemented some new ones providing the necessary additional functionalities. Most of these filters take parameters and could be used for other tasks in the future, due to their general nature. The two tables (6.1, 6.2) below show an overview of these filters and their purpose.

New filters	
Filter name	**Purpose**
BandPassFilter	Filters points in a selected vertical range.
CantileverFilter	Finds cantilever points in region of interest.
CorrigateCentroidsFilter	Refines centroids by moving them to the true centroid.
MinDistanceClusterFilter	Implements the Euclidean clustering algorithm.
MinHeightFilter	Keeps points above or below specified height.
RansacCylinderFilter	Finds mast points for centroids.
StaggerFilter	Finds contact cable points with excess stagger and displays warning for lack of stagger.

Table 6.1: *The newly implemented filters*

Modified filters	
Filter name	**Modifications**
HeightFilter	Added multiple seed support and parameter for height limit.
OutlierFilter	Added parameters for radius and min. number of neighbours.
RansacFilter	Added parameter for distance threshold of RANSAC.
WidthFilter	Added multiple seed support.

Table 6.2: *The modified filters*

6.4 Modifications in the framework

To make implementation easier I extended some of the default functionalities of the railroad project, such as seed handling, the piping mechanism and point cloud reading.

6.4.1 Handling multiple seed clouds

By default, the program was only capable of handling a single input point cloud as a seed, but the cantilever segmentation required both the already detected tracks

and wires as input next to the base point cloud.

For this, I created a new class called SeedHelper, which encapsulates loading, checking and storing the list of required seed point clouds. Currently, there are four types of infrastructure for which the user can load seeds: rail, pole, cable and ties.

The user can supply the list of file paths for the different seed point clouds and another list for their corresponding types with two named arguments.

There are various error-checking mechanisms implemented to handle incorrect parameterization cases, like the number of supplied paths and types not matching, or missing type of seed required by one of the filters.

An instance of the SeedHelper class is created at runtime and is passed down the processing pipes, making it possible to query and use any of the loaded seed clouds in the different filters.

6.4.2 Keeping a reference to the original input cloud

Due to the architecture of the piping mechanism and filters, only the filtered points and the seed cloud were passed down to the next filter originally. This meant, that if for example, we ran a clustering filter, resulting in the centroids of each cluster found, we would not have access to the original input point cloud anymore, only these centroid points.

As a workaround, the input point cloud could be supplied as a seed cloud too, but this was neither practical nor possible in case a different point cloud was required as a seed by one of the filters.

To overcome this limitation, I modified the CloudProcessor class and the main entry points of the program, so that a reference is held to the original input cloud, easily reachable from any filter at any point. This reference is constant, preventing the programmer from unintentionally modifying the original point cloud.

6.4.3 Point cloud demeaning

Upon reading the las/laz format input and seed clouds, they were originally scaled and repositioned by the corresponding values found in the headers.

As our datasets were georeferenced in the EOV coordinate system, this meant that each X coordinate got shifted to a value between 200 000 and 400 000, with the Y coordinates getting remapped to range from 400 000 up to 1 000 000.

Most of the time this did not pose any problems, but when estimating normal vectors during mast segmentation, unlikely bad results started to show. The normal vectors get calculated by using PCA, during which these extremely large values represented as floating point numbers get multiplied together. This is where a lot of precision was lost, due to the sampled points being in close proximity to each other, and the large mantissa.

Equation 6.1 shows the calculation of the covariance matrix, where k is the number of points considered in the neighbourhood of p_i and $\bar{p}$ represents the 3D centroid of the nearest neighbours.

$$C = \frac{1}{k} \sum_{i=1}^{k} \cdot (p_i - \bar{p}) \cdot (p_i - \bar{p})^T \tag{6.1}$$

A naive solution to this problem is to disregard any kind of position offset from the point cloud headers. However, this introduces another problem, where if the seed clouds don't share the exact same positional offset with the input cloud, they become misaligned. So instead, a global demean value is getting stored based on the XYZ offset of the base input cloud, which then gets subtracted from each loaded point cloud's offset. This new offset then gets added to the corresponding clouds XYZ coordinates, essentially leaving the input point cloud in its original position, and shifting the rest to its coordinate system.

Before writing the results after the processing has been finished, the demeaning steps are applied again in reverse, shifting back the points to their original locations, with the original offset values written to the header, keeping them comparable with the untouched input point clouds.

Chapter 7

Results

In this chapter, I'll demonstrate the results achieved with my methods through the measured running times and accuracy. The running times were measured in the following environment:

- Ubuntu 20.04 operating system running in a VirtualBox VM[1],
- Intel i5-6600K CPU running at 3.5 GHz,
- 32 GB of RAM (8 GB assigned to VM).

For each measurement, an average of four consecutive runs are shown. The running times were negatively affected by the virtualized environment, but they are sufficient to compare the running times across different kinds of input data.

For the calculation of the precision and accuracy measurements equations 7.1 and 7.2 were used, where TP = true positive, TN = true negative, FP = false positive and FN = false negative count.

$$Precision = \frac{TP}{TP + FP} \tag{7.1}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.2}$$

7.0.1 Mast detection

Table 7.1 shows the measured running times of the mast detection pipeline on the different samples. As the data shows, the running time of the detection scales in a linear fashion with regards to the length of the section fed to the pipeline.

[1] Virtual Machine

	Mast detection run times		
	Number of points	Length of section	Running time
MÁV-simple	7.3 million	100 metres	3.8524 s
MÁV-long	40.3 million	560 metres	39.3928 s
GYSEV-standard	7.4 million	185 metres	6.6212 s
GYSEV-mixed	7.3 million	188 metres	6.5092 s

Table 7.1: *Mast detection pipeline running times on the sample inputs*

The next table (7.2) shows the running times of the different stages of the mast detection pipeline. As visible from these measurements, the first and last stages of removing unnecessary points and the actual segmentation also scale in a linear fashion with the size of the inputs. Relative to these, however, the stage for finding possible mast centroids can take an unexpectedly long time (mainly as seen with the last two samples).

	Detailed run times of mast detection		
	Removing unneccessary points	Finding possible mast centroids	Segmenting masts
MÁV-simple	0.9040 s	0.5451 s	2.3295 s
MÁV-long	5.8372 s	13.2667 s	19.6883 s
GYSEV-standard	1.2664 s	4.0845 s	1.1420 s
GYSEV-mixed	1.2919 s	3.8400 s	1.7133 s

Table 7.2: *Detailed running times of the mast detection pipeline on the sample inputs*

Further breakdown of this second stage of the mast detection pipeline shown by Table 7.3 reveals some valuable information. In this table, the two most time-consuming steps are shown in relation to the overall running time of the centroid localisation stage.

	Detailed run times of mast centroid finding stage			
	Density filtering	Centroid refinement	Complete stage	Time % of centroid refinement
MÁV-simple	0.0862 s	0.4117 s	0.5451 s	**75.5 %**
MÁV-long	0.6738 s	11.5280 s	13.2667 s	**86.9 %**
GYSEV-standard	3.4337 s	0.6318 s	4.0845 s	15.4 %
GYSEV-mixed	2.7496 s	0.6231 s	3.3840 s	18.4 %

Table 7.3: *Detailed running times of the mast centroid localisation stage on the sample inputs*

As it is visible from the data, the centroid refinement step can take up to 87 % of the processing time of this stage (see **MÁV-simple/-long** samples). This is caused by the refinement process being executed on the complete input cloud, as only that and the original centroids are available inside the filter due to the current architecture of the railroad framework. In Section 8.1 on Page 50, I will address some possible solutions to this, but as the measured running times were found to be sufficiently quick, these modifications were not implemented.

	Accuracy of mast detection			
	False negatives	False positives	Precision	Accuracy
MÁV-simple	0.62 %	1.55 %	98.45 %	99.99 %
MÁV-long	0.72 %	1.07 %	98.93 %	99.99 %
GYSEV-standard	4.01 %	0.59 %	99.41 %	99.99 %
GYSEV-mixed	5.85 %	6.80 %	93.20 %	99.95 %

Table 7.4: *Precision and accuracy of the mast detection pipeline*

Finally, Table 7.4 shows the precision and accuracy evaluation of the mast detection pipeline. The mast points were successfully detected for all samples, with some minor errors. Figure 7.1 shows the detected masts in red on the **MÁV-long** sample.

As it is also visible from this data and Figure 7.2, the **GYSEV-mixed** sample showed the worst results. This is due to the non-cylindrical masts found in this sample, one of them also having tensioning weights installed. Even though the pipeline was not optimized for such cases, it still yields usable results.

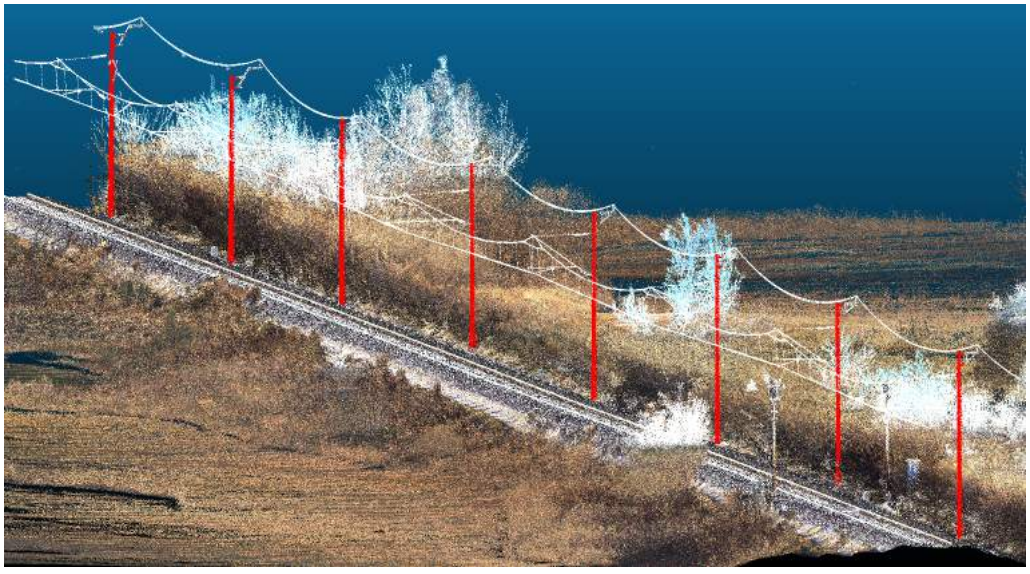


Figure 7.1: *The detected masts shown on the **MÁV-long** sample*

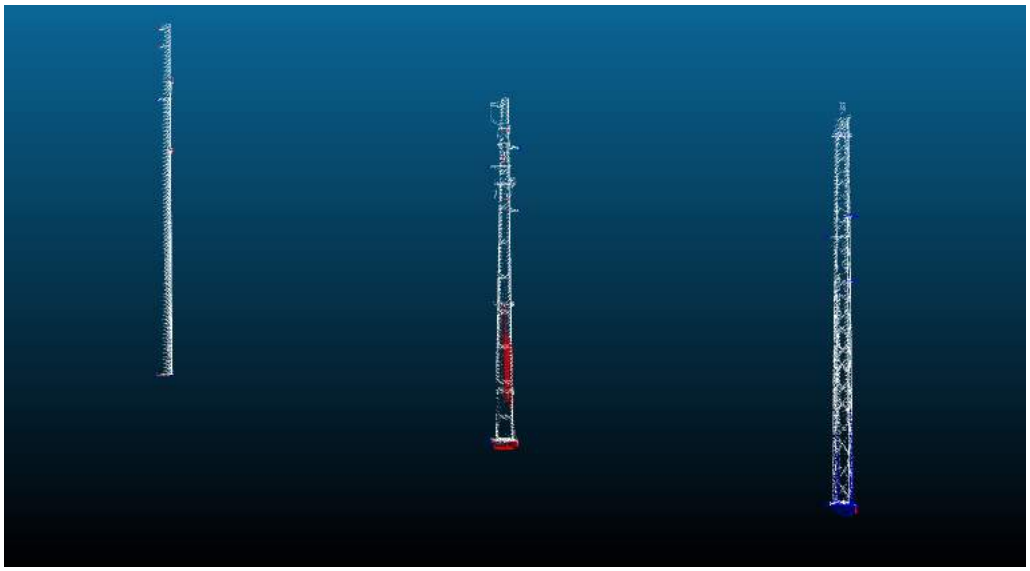


Figure 7.2: *False negatives (blue) and false positives (red) of the mast detection results on the* ***GYSEV-mixed*** *sample*

7.0.2 Cantilever detection

The cantilever detection pipeline was evaluated on the **MÁV-simple** and **MÁV-long** samples. The table below (7.5) shows the running times of the different steps of the pipeline, and the accumulated running time.

	Cantilever detection run times			
	Width filter	Height filter	Cantilever filter	Total
MÁV-simple	0.7585 s	0.0366 s	7.3423 s	8.1374 s
MÁV-long	4.4318 s	0.2149 s	55.0550 s	59.7017 s

Table 7.5: *Cantilever detection pipeline running times on the sample inputs*

The accuracy and precision of the cantilever detection are shown by Table 7.6. There are some false negatives, but overall the pipeline successfully segments the points belonging to cantilevers. Figure 7.3 demonstrates the accuracy of the pipeline visually on the **MÁV-simple** sample.

	Accuracy of cantilever detection			
	False negatives	False positives	Precision	Accuracy
MÁV-simple	2.76 %	1.64 %	98.36 %	100 %
MÁV-long	4.74 %	1.50 %	98.50 %	100 %

Table 7.6: *Precision and accuracy of the cantilever detection pipeline*

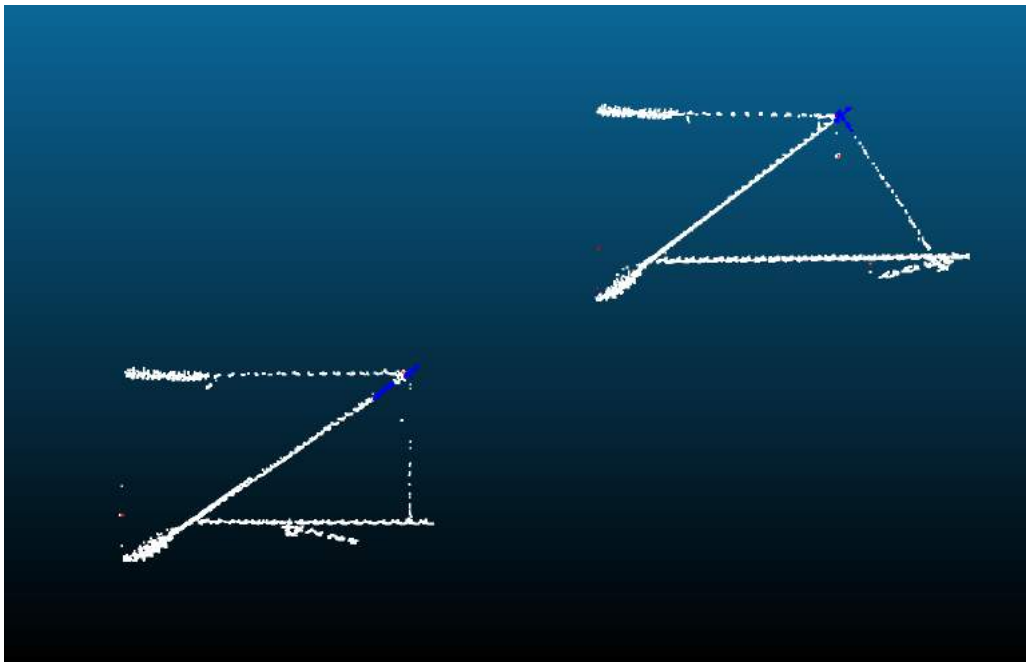


Figure 7.3: *False negatives (blue) and false positives (red) of the cantilever detection results on the **MÁV-simple** sample*

Figure 7.4 shows the detected cantilevers in red on the **MÁV-long** sample.

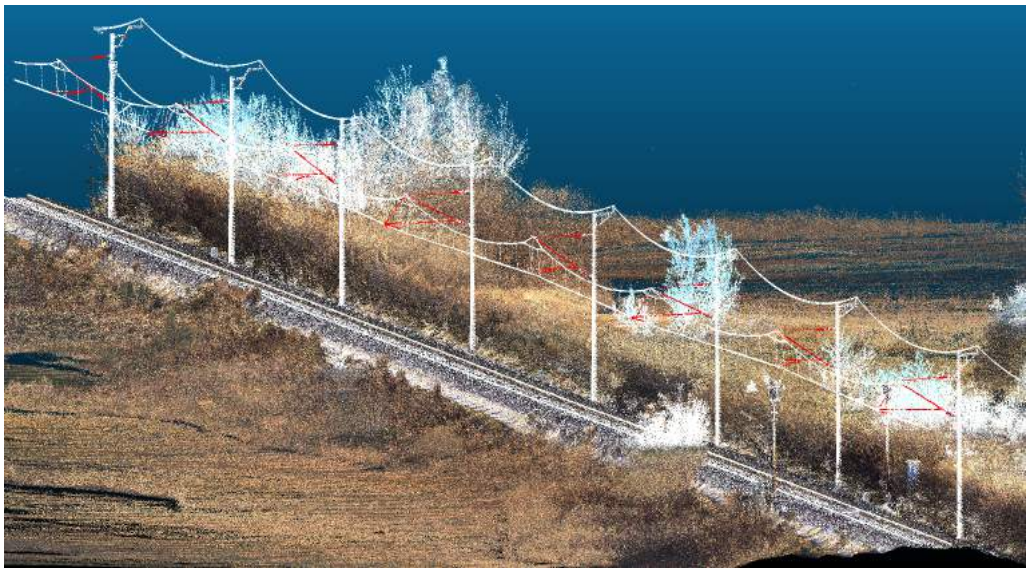


Figure 7.4: *The detected cantilevers shown on the **MÁV-long** sample*

7.0.3 Contact cable stagger checking

The contact cable stagger checking pipeline was evaluated on the **MÁV-simple** and **MÁV-long** samples. The table below (7.7) shows the running times of the different steps of the pipeline, and the accumulated running time.

	Stagger checking run times			
	Min-height filter	Ransac filter	Stagger filter	Total
MÁV-simple	0.0176 s	0.0038 s	0.1454 s	0.1667 s
MÁV-long	0.0953 s	0.1194 s	1.8388 s	1.6241 s

Table 7.7: *Stagger checking pipeline running times on the sample inputs*

Running the stagger checking pipeline on the two samples with the parameters matching the construction time and class of the section in question (0.4 metres maximum stagger with a threshold of 3 cm) verified that the stagger on these sections is within operational limits, with no points being classified as excessively staggering, and no warning shown for lack of stagger either. Figure 7.5 shows the output of the algorithm with overly strict parameters (0.2 metres maximum stagger with a threshold of 0 cm) with the excessively staggering points shown in red. After setting the parameter for the minimum stagger limit at an overly strict value of 0.35 metres the warning message about potential lack of stagger was also correctly displayed.

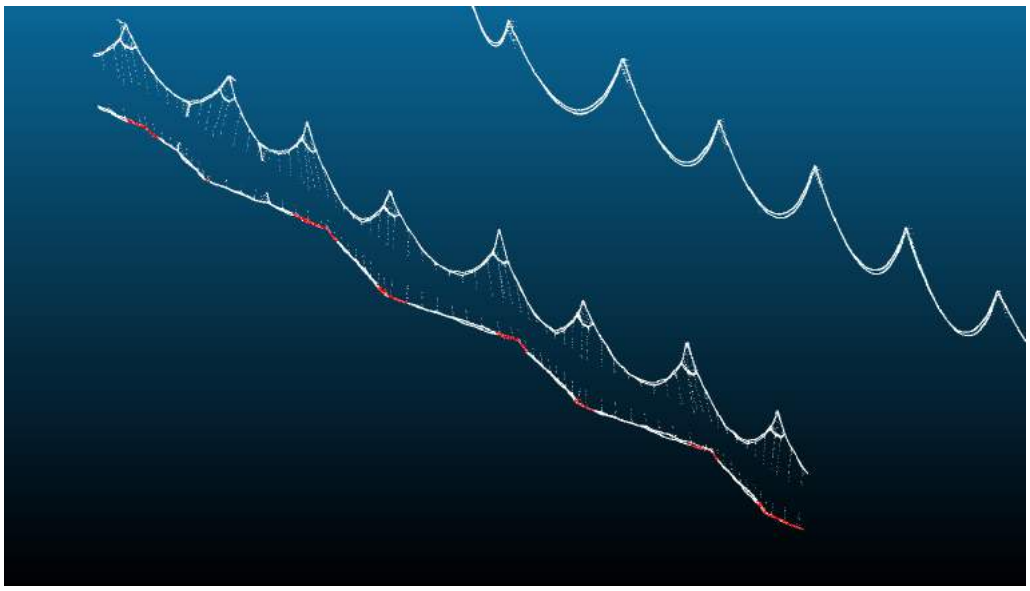


Figure 7.5: *Running the stagger checking with overly strict parameters on the* ***MÁV-long*** *sample*

Chapter 8

Conclusion

During my work, I made myself familiar with both LiDAR and railway infrastructure-related technological terms (mast, cantilever segmentation and contact cable stagger detection) to get a better understanding of the goals I aimed to achieve. I researched existing literature on both traditional- and machine-learning-based solutions to these problems, and based on this I decided to tackle them with the traditional methods.

While learning from this information, I also tried diverging from existing solutions, to explore hidden potential in the geometrical relationship between the different elements of the railway infrastructure. The implemented methods yield comparable, or even better results to what was shown by other researchers, with regards to both running time and accuracy. The implemented mast segmentation pipeline showed an average accuracy of 99.87 %, with an average running time of 4.48 seconds per 100 metres. The cantilever segmentation method yielded 100 % accurate results, taking 9.34 seconds per 100 metres to run on average. The contact cable stagger detection solution was able to detect both the lack- and the excess of contact cable stagger successfully, with an average running time of 0.25 seconds per 100 metres. The running time of these implemented algorithms scales in a linear fashion with regard to the size of the input clouds.

8.1 Future work

All of the implemented algorithms yield some false positives and false negatives. This is most prominent with the mast detection pipeline, especially in the case of non-cylindrical masts. A better and more precise algorithm could be developed for detecting these non-cylindrical masts, instead of the currently used, overly simple method of filtering points within a cylindrical bounding box of their presumed location. The speed of this pipeline could be also significantly improved by caching the already cleaned points outside of the region of interest for the refinement of candidate centroids or using an Octree data structure to speed up searching in the original input cloud. Some more, less significant running time improvement could also be achieved by running the actual mast segmentation stages in parallel for each candidate centroid.

The cantilever detection pipeline's speed could also be improved. The main bottleneck is the step where the points in the vicinity of masts are removed. To overcome this bottleneck, a tight bounding box could be constructed for the individual masts. Removing the points falling inside these bounding boxes would be much faster than individually checking the neighbourhood of all mast points.

The stagger checking pipeline could be refined to make detection of lack of stagger more localised, reporting the exact EOV coordinate of the problematic areas instead of checking globally on the complete section.

Finally, the railroad tool could be modified to allow for running pipelines in parallel for multiple consecutive sections, further speeding up the detection of infrastructure and error checking for complete datasets.

Acknowledgements

I'm extremely grateful to Máté Cserép for his help as my supervisor. I would like to express my gratitude to Csaba Bajnóci and Zoltán Németh, the experts at MÁV, for generously sharing their knowledge and providing valuable information regarding railway infrastructure-specific questions. I'm also grateful to MÁV for providing the datasets that were crucial for this work. Additionally, I extend my appreciation to Judit Knoll and Levente Greczula for taking the time to proofread this thesis and providing valuable feedback to further improve its quality. Lastly, I would like to thank my family and my colleagues for their continuous encouragement.

Bibliography

[1] Adnan Shaout, Dominic Colella, and S. Awad. "Advanced Driver Assistance Systems - Past, present and future". In: *2011 Seventh International Computer Engineering Conference (ICENCO'2011)*. 2011. DOI: 10.1109/ICENCO.2011.6153935.

[2] *Transport and environment report 2021*. [Online; accessed 18. Apr. 2023]. Feb. 2023. URL: https://www.eea.europa.eu/publications/transport-and-environment-report-2021.

[3] UIC. *RAILISA statistics, 2021 data*. [Online; accessed 18. Apr. 2023]. URL: https://uic-stats.uic.org.

[4] *"Eyes" for Autonomous Mobile Robots*. [Online; accessed 15. May 2023]. May 2017. URL: https://news.panasonic.com/global/stories/805.

[5] J. H. Middleton et al. "Resolution and Accuracy of an Airborne Scanning Laser System for Beach Surveys". In: *J. Atmos. Oceanic Technol.* 30.10 (Oct. 2013). ISSN: 0739-0572. DOI: 10.1175/JTECH-D-12-00174.1.

[6] *Siemens presents the world's first electric railway*. [Online; accessed 6. Nov. 2022]. Oct. 2022. URL: https://www.siemens.com/global/en/company/about/history/stories/on-track.html.

[7] Hadas Ádám. "The Effect of Overhead Line and Pantograph Failure for Railway Operations". In: *MKK* 29.2 (Aug. 2019). In Hungarian. ISSN: 2063-4986. DOI: 10.32562/mkk.2019.2.12.

[8] *Catenary Measurements*. [Online; accessed 28. Apr. 2023]. Apr. 2023. URL: https://rail-vision.com/infrastructure/overhead-lines/catenary-measurements.

[9] Mostafa Arastounia. "Automated Recognition of Railroad Infrastructure in Rural Areas from LIDAR Data". In: *Remote Sens.* 7.11 (Nov. 2015). ISSN: 2072-4292. DOI: 10.3390/rs71114916.

[10] Elżbieta Pastucha. "Catenary System Detection, Localization and Classification Using Mobile Scanning Data". In: *Remote Sens.* 8.10 (Sept. 2016). ISSN: 2072-4292. DOI: 10.3390/rs8100801.

[11] Yixuan Geng et al. "UAV-LiDAR-Based Measuring Framework for Height and Stagger of High-Speed Railway Contact Wire". In: *IEEE Trans. Intell. Transp. Syst.* 23.7 (May 2021). ISSN: 1558-0016. DOI: 10.1109/TITS.2021.3071445.

[12] Friderika Mayer. "Powerline tracking and extraction from dense LiDAR point clouds". http://hdl.handle.net/10831/56221. MSc thesis. ELTE Eötvös Loránd University, Faculty of Informatics, 2020.

[13] Máté Cserép, Péter Hudoba, and Zoltán Vincellér. "Robust Railroad Cable Detection in Rural Areas from MLS Point Clouds". In: *ScholarWorks@UMass Amherst* 18.1 (2018). DOI: 10.7275/z46z-xh51.

[14] Alexis Gutiérrez-Fernández et al. "Automatic Extraction of Power Cables Location in Railways Using Surface LiDAR Systems". In: *Sensors* 20.21 (Oct. 2020). ISSN: 1424-8220. DOI: 10.3390/s20216222.

[15] Javier Grandio et al. "Point cloud semantic segmentation of complex railway environments using deep learning". In: *Automation in Construction* 141 (2022). URL: https://doi.org/10.1016/j.autcon.2022.104425.

[16] Md Zahangir Alom et al. "A State-of-the-Art Survey on Deep Learning Theory and Architectures". In: *Electronics* 8.3 (Mar. 2019). ISSN: 2079-9292. DOI: 10.3390/electronics8030292.

[17] A. Manier et al. "Railway lidar semantic segmentation with axially symmetrical convolutional learning". In: *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci* (2022). URL: https://doi.org/10.5194/isprs-annals-V-2-2022-135-2022.

[18] Xinyi Yu et al. "Real-time Rail Recognition Based on 3D Point Clouds". In: *Computer Vision and Pattern Recognition* (2022). URL: https://doi.org/10.48550/arXiv.2201.02726.

[19] S. A. Guinard et al. "Fast weakly supervised detection of railway-related infrastructures in lidar acquisitions". In: *ISPRS Ann. Photogramm. Remote Sens. Spatial Inf. Sci* (2021). URL: http://dx.doi.org/10.5194/isprs-annals-V-2-2022-135-2022.

[20] Balázs Tábori. "Fragmenting LiDAR point cloud of railway tracks". In Hungarian, http://hdl.handle.net/10831/77213. MSc thesis. ELTE Eötvös Loránd University, Faculty of Informatics, 2021.

[21] Adalbert Demján. "Object extraction of rail track from VLS LiDAR data". http://hdl.handle.net/10831/56224. MSc thesis. ELTE Eötvös Loránd University, Faculty of Informatics, 2020.

[22] R. Rusu. "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments". In: *KI - Künstliche Intelligenz* (2010). URL: https://mediatum.ub.tum.de/doc/800632/941254.pdf.

List of Figures

List of Tables