Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology And Methodology

Object extraction of rail track from VLS LiDAR data

Supervisor:
Máté Cserép
Assistant Lecturer

Author:
Adalbert Demján
Computer Science MSc

Budapest, 2020

Contents

Chapter 1

Intro

Railways are an integral part of our transport since the 19th century. Millions of people use it every day and with the latest environmental efforts, this number may start to grow. Naturally, the maintenance of the railroads is a high priority. Today there are certain caretakers whose job includes: walking by the designated railway section, visual inspection of track and equipment, detection of deficiencies, and elimination of minor operational disruptions to ensure undisturbed rail traffic.

The aim of my thesis was to develop and implement an automated way of detecting rail tracks in 3D LiDAR point clouds. The LiDAR datasets are provided by the Hungarian State Railways (in Hungarian: MÁV Magyar Államvasutak Zrt), which is the national railway company of Hungary. The first dataset covers about 100m of railway infrastructure and contains more than 2 million points. The second dataset covers about 200m of railway infrastructure and consists of more than 6 million points. The methodology developed in this study recognizes the rail tracks by using height classification, eigendecomposition, and a growing algorithm. The developed algorithm successfully recognized the rail track without any outliers and is applicable with any slope.

Chapter 2

Background

2.1 LiDAR technology

LiDAR (Light Detection and Ranging)is a surveying method that measures the distance to an object by emitting light and measuring the data of the reflected feedback with a sensor. The determination of the location of large objects is done by performing a lot of this distance measurement in a short period of time which is called laser scanning. LiDAR is, therefore, an active remote sensing system with its own signal source and unlike radar, it operates in the ultraviolet, visible, or infrared range.

The types of measurements based on LiDAR can be categorized:

Meterologic LiDAR : Scanning atmospheric properties from the ground up to the top of the atmosphere. Used for studying aerosols, clouds, temperature, etc.

Laser profile measurement : Detecting, measuring and evaluating profiles on surfaces of different objects. It's most commonly used precision mechanics and electronics.

Airborne laser scanning(ALS) : Scanning the earth's surface by emitting the light from an airborne object. Can be used in, among other, hydrology, geomorphological mapping and for scanning railway environments.

Terrestrial laser scanning(TLS) : The ground-based version of the airborne scanning typically used for terrain and landscape mapping.

Mobile laser scanning(MLS) : The used scanning type for my datasets, detailed in the following subsection.

2.1.1 Mobile laser scanning

Mobile laser scanning collects geospatial data from a mobile vehicle fitted with LiDAR, cameras, and other remote sensors. This mobility can be provided by cars, trains, trucks, manned and unmanned boats. The result is a point cloud of the scanned area with 3D coordinates. Figure 2.1 shows the result of MLS in an urban area.

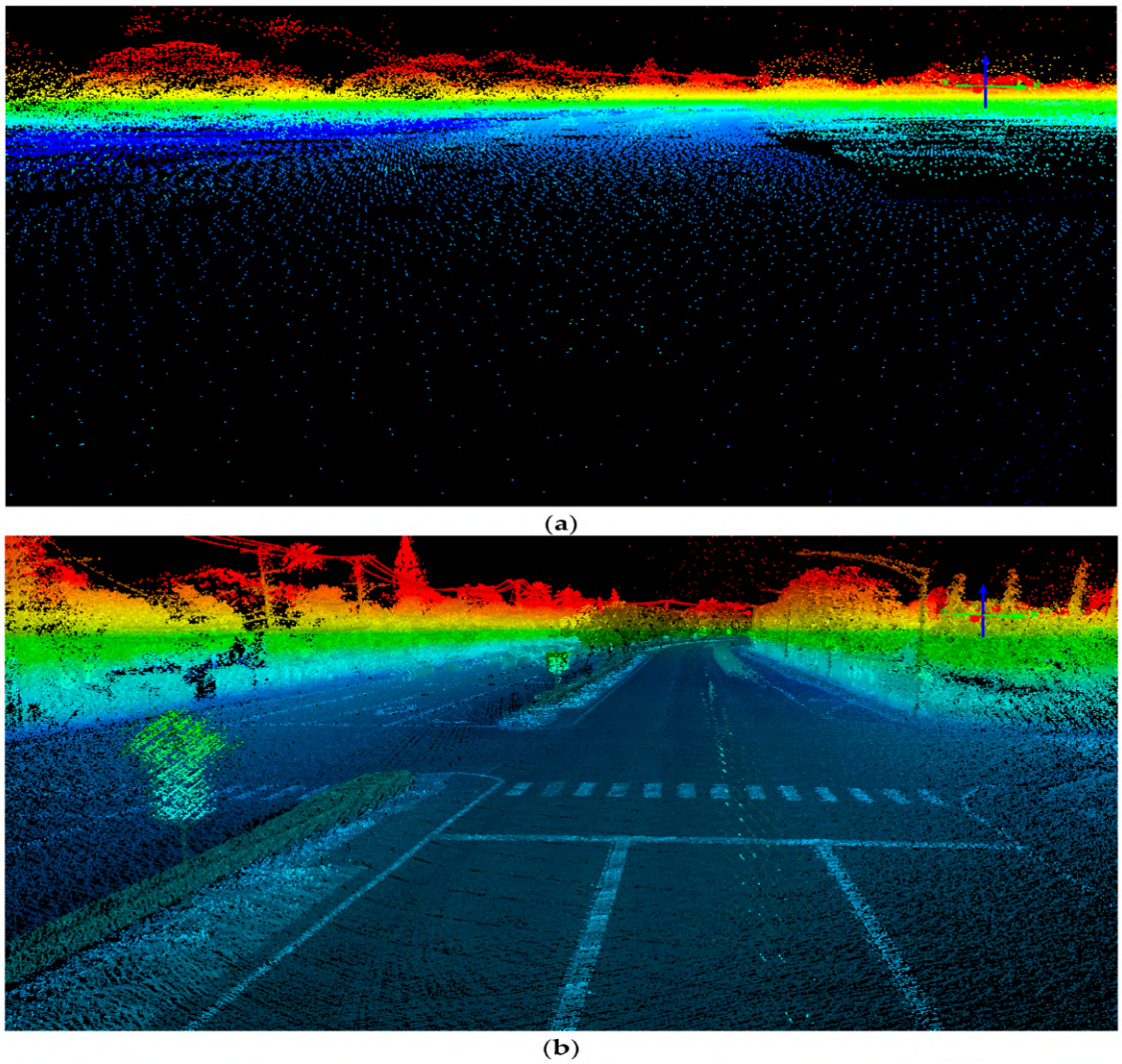

Figure 2.1: Results of mobile laser scanning in an urban area

Railroad detection

In railroad detection, we try to recognize the points from the point cloud which make up our objects of interest. These usually are:

Track bed : The underlying surface of the rail tracks, which holds those in line and on the surface

Rail tracks : Two parallel sheets of steel that are the platform for the trains in motion.

Masts : Posts on the side of the rail tracks which hold the overhead cables

Overhead Cables : Cables that either transmit power to the train(contact) or help to hold the cable structure in place(catenary).

Cantilevers : Thin metal tubes that connect the masts with the catenary cables

In the study, the focus is on the detection of the trackbed and the rail tracks.

2.2 Literature review

My main aim during the literature review was to research the evolution of the methods from the late 2000s to today and find those methodologies, that can be the base for this study.

Rail track extraction using an adapted RANSAC algorithm

The first article I examined was the work of Neubert et al. [1]. It proposed two methodologies:

- The usage of an adapted RANSAC algorithm
- Knowledge-based detection with 2D-cuts

The first step is to filter the data, by classifying the points by their height from the ground, which allows us to focus the algorithm on the vicinity of the height of the rails. After that, a second filter is applied where we examine the differences in height in the points' neighbourhood. We calculate each points' difference to the

ground height in its immediate neighbourhood. Our dataset will contain the points which are in a certain interval.

The RANSAC algorithm is an iterative method for outlier detection, developed by Fischler and Bolles [2]. It is a learning technique to estimate parameters of a model by random sampling of given data. To give an optimal result it uses a voting scheme, which is based upon two assumptions:

1. Noisy features will not vote consistently
2. There are enough features that can give us a good model

The aim of this method is to use the RANSAC algorithm to generate simple geometric objects from the dataset, such as lines and circular arcs. First, we tile the dataset along the trajectory aligning with the rail direction. We now use the RANSAC for each tile. Originally the algorithm only gives one solution, but we adapt it to detect multiple lines or curves. Then the from the detected lines the best-fitting regression lines are calculated. Finally, the inliers are used to calculate a curve-fitting function based on the Least Squares Method.
The results of the overlapping tiles are merged automatically.

Rail object detection with 2D cuts

First, a 2-meter interval is selected in a small stripe perpendicular to the local trajectory from the ALS data. After that we use the following conditions for a hierarchic calculation to detect the track axis:

- The rail bed around the track is nearly flat
- There is no object in the railway loading gauge
- The mean track profile is known and there is a fix distance between the two rails

In the last step of the hierarchical analysis, the local terrain model is compared with the reference profile. If the second condition is fulfilled this comparison is performed for all areas. The result is the points that represent the track axis.

Next is the detection of the catenary cables, which have the following conditions:

- They have an 8.5 cm to 10.0 cm laser footprint size because of the small dimension of the cable
- Its points should be higher than 5.0 m above the track model
- They can only be located in a 0.4m buffer regarding the track axis

All the points located with a height of 5.0m to 6.1m are deducted as contact wire points. The rest are mounting wires or other mounting components. These points can be converted to lines as a final step.
In the next step we detect the cantilevers with the conditions below:

- Cantilever points have at least 0.8 - 1.6m distance from the track axis
- The measurements can only be 'first-pulse' or 'only-pulse' points
- The points have a normalized height of 5.0m above the ground

The corresponding points are converted into binary raster, then the clump-and-sieve function is used to separate the interfering points from the wires. The cantilever points then are used to detect the railway poles by searching for the orthogonal local maxima in the track direction.

Rail track detection with ortho-imagery

The next article I explored was published by Beger et al. and it describes an advanced methodology based similar to the aforementioned RANSAC-driven method [3]. It uses a combined analysis of EHR aerial images and dense ALS data.

The first step is extracting a rail track mask from the ortho-imagery. This consists of the following four steps:

1. Edge layer generation: We use a filter developed by Lee [4], with which we can successfully aggravate edges, as discussed by Rieger et al. [5]
2. Chess-board segmentation: The entire image is segmented into 4x4m squares, then each tile's maximum greyscale value is calculated from the edge layer.

The railroad tracks' pixel values will be among the highest because they are ideal edges. Cells that have high enough pixel value are selected as rail track containers.

3. Quad-tree segmentation: This algorithm splits the image into segments of different sizes. The square sized segments are stored in the quad-tree grid. The squares are analyzed by their brightness and are classified. These classes are quad-tree segmented based on the pixel values calculated before. If a class is fit the criteria calculated of the mean edge layer value, then it is considered as a rail track class.

4. Multi-resolution segmentation: We merge the image objects successively with a given set of parameters.

The second step is the classifications of the laser point data. First, we filter the points above a specific height. Then we buffer the previously calculated rail track mask by 2m. All points that are spatially located in the buffered mask are selected. Then we calculate the height differences from the lowest point in the local neighbourhoods. Points which have a difference between two given thresholds and have a lower slope gradient then 15 degrees are classified as rail points.

The final step is the rail track approximation. This is similar to the method given by Neubert et al.[1], we tile the point cloud then use a modified RANSAC for the rail track detection. Finally, curve-fitting is used to for the exact approximation of the centre lines.

Rail detection with cross-section based template matching

The following methodology was developed by B. Yang and L.Fang in 2014. [6]
The method can be divided into three main parts:

1. Partitioning the point cloud into railway cross-sections

2. Extracting candidate railway beds with a moving-window operator

3. Detecting track points by the geometric and intensity of local points.

Directly extracting the trackbed area is very time-consuming from MLS point cloud, so the method uses the fact the cloud can be divided into consecutive cross-sections called scanning lines. These generally show the spatial patterns of the objects in the railroad corridor.

To extract the bed points we now use the moving-window operator. We use three adjacent windows which slide along the scanning lines. If the operator locates on the edges of the trackbed, the first window will show a height jump, while the two other will have sleight height variation. We determine a ruleset which matches the former attribute and with that, we get the two endpoints of the railway bed. Naturally, the points between those endpoints are the sought ones.

For the recognition of the rail track points, two characteristic features are defined for each candidate point.

1. Shape feature: The rail track has a unique shape, it has a height difference and a special angle between the head and foot of the rails. This enables to use the maximum height difference and the slope to locate the rail points.

2. Intensity Data: Different materials have different reflection intensities. In a railway bed, the tracks have low-reflectance intensities for dark and iron material surface, while the trackbed has a high-reflectance intensity for the stone material surface. This results in a larger standard deviation of intensity when examining the trackbed points.

Based on the features mentioned above we can classify the suitable points as rail track points.

In a lot of section, the sleepers show similar geometric attributes as the rail tracks, which results in false positives. To get rid of these we use linear structure labeling in which we cluster the track points. The points in one cluster are labelled one segment and will be connected to form a track model.

Railway tracking with Kalman filter

The following algorithm is a Kalman filter based method and was developed by Jwa and Sonh in 2015. [7]
The algorithm can be divided into three steps:

1. Preprocessing the LiDAR data
2. Clustering track components and reconstructing initial track models
3. Railway tracking with Kalman filter

The main aim of preprocessing is to determine the orientation of the railway. This is important because the algorithm uses a strip for detecting the initial track regions. We use the line equation in Hesse normal form and its parameters base on weighted least squares minimization. Then we partition the data into a fixed grid space, then we calculate each grid space its centre point. The point density is considered as a weight value in the estimation process of the railway trajectory vector.

The next step is the railway track localization. The strip contains various objects, so to deal with the uncertainty caused by that, we turn to the Bayesian view. First, we detect track zones including the rail pairs in the initial strip. For this, a supervised classification as a machine learning technique is adopted by using a set of features which represents unique characteristics in describing the track zone.

The final step is using the Kalman filter for the tracking. This consists of two steps, the space and the measurement update. After the space update, we determine the potential track region is by using the measurement deviation and railhead points are segmented from track points detected based on the GMM-EM[1] technique and mean shift clustering. Then we estimate the state vector using the track points detected in the current space based on the measurement update step.

Rail object detection with height deviation

The final chapter of my research was based upon Mostafa Arastounia's work, who published several methods throughout the year.

[1] Gaussian Mixture Model - Expectation-Maximization

The first method examined was published in 2015[8]. The method divides the recognition of each type of object into three primary steps:

1. Inspecting the local neighbourhoods
2. Classification of the points
3. Segmentation

A point belongs to a neighbourhood of a P point if their 3D Euclidean distance is equal to or less than a certain radius. A K dimensional tree data structure is computed for increasing the efficiency of the search. Classification of a point is based on the 3 following attributes: the local neighbourhood structure, shape of objects and topological relationships among the objects. Segmentation clusters the classified points which are i.e belonging to an object type.

The method performs the object-detecting in the following order:

1. Recognition of trackbed
2. Recognition of rail tracks
3. Recognition of overhead power cables
4. Recognition of masts and cantilevers

The points of the trackbed are recognized by examining the histogram of the height standard deviation of each point's neighbourhood. The sought points have the least amount of height variation because they are designed this way for safety reasons. Therefore these points will form a peak in the histogram, and from which the track bed's points are calculated. The recognition of the rail tracks starts off with a similar method, we examine a similar histogram computed from the track bed's points. We expect two peaks, from which the smaller one represents the non-flat portion of the trackbed containing a rail track. Next step is clustering the candidate rail points and afterwards, the recognition of the rail track point is done with a growing algorithm.

Rail object detection with height percentiles

In 2016 an alternative method was published with the contribution of S. Oude Elberbink[9].
The method consists of the following steps:

1. Coarse classification of points
2. Identification of the rail track's points
3. Identification of the overhead cables' points

Coarse classification is based on the height of the points. Since the most points in our cloud belong to the trackbed, the most common height is the track bed's height. We cluster the points within half a meter higher or lower than the above-mentioned height into one class. Points whose height is between 5m and 5.5m above the trackbed are put in another class, this includes the points of the contact cables. Then lastly the points that are higher than 5.5m above of the trackbed are put in the third class, which contains the catenary cables' points.

For the identification of the rail track we only need to examine the points in the first class. First, we identify the candidate rail track points with applying a 2D planimetric grid to the points of the class. Then we mark the grid cells which have height variation as high as the rail track height. The height variation is computed between the 5th and 95th height percentile to filter out the outlier noises. Lastly, we choose the points higher than the 90^{th} height percentile from the marked grid cells as our candidate points.

It is expected that we have candidate points which are not part of the rail track(false positives). To exclude these points we use a model-driven approach: template matching. We make a greyscale image of the candidate points and the contact cables' class' points. Then we make a template image of one meter of the railroad corridor. Then we match the template image with the greyscale image, calculating the correlation between every pixel checking from every angle. If the correlation

exceeds a certain threshold, the pixel is containing true rail track points.

The final step is including the non-classified rail points, we fit a line to one-meter pieces of rail tracks and the pixels lying on the line are considered as rail tracks.

Rail object detection with eigendecomposition

An enhanced and more universal method was developed in 2017 based on the previous methodologies[10].
The algorithm is composed of two main parts:

1. Concurrent recognition of points of the rail track and the contact cables
2. Recognition of the catenary cables

The method uses a similar coarse classification what is described in Arastounia and Oude Elberink's 2016 article[9] but with a slight modification. The original classification assumes that the area's altitude is roughly the same everywhere without bigger changes. This can be proven false in mountainous areas, so we apply the method to small portions of the dataset, where the height difference between the rail tracks and the over-head cables remain constant. Otherwise, the method is the same, we classify the points into 3 classes for the trackbed, contact cables and catenary cables.

The rail seed points are selected by using eigendecomposition on the points of trackbed cluster. The covariance matrix is calculated to each point's local neighbourhood, and after that principal component analysis is used to determine the eigenvectors and the eigenvalues. All local neighbourhoods of the cluster which satisfy a given condition are marked as containing rail points. Finally points higher than 90th height percentile within each neighbourhood are selected as rail track points. False positives are excluded by applying a 2D Hough transformation to the rail points.

With the help of these rail points, we identify the contact cables by projecting the second cluster into a planimetric plane. Points which are within one meter 2D Euclidean distance of the rail pairs are marked as contact cable seed points. Those

seed points that lie within one meter 3D Euclidean distance of one another are aggregated into segments.

The final step is using a growing algorithm for both of the rail tracks and the contact cable. The growing algorithm is considering if the currently examined neighbourhood point height is within the rail's height and if the vector connecting the point and the rail segment is roughly matching the rail segment's orientation.

2.3 Comparison of the methods and conclusion

The performance of the above-mentioned methods when detecting rail tracks are shown in Table 2.1.

	Precision(%)	Accuracy(%)
Neubert et al. - RANSAC	Above 90	-
Neubert et al. - 2D cuts	Above 90	-
Beger et al.	90 - 98	88 - 97
Yang and Fang	99.78	95.43
Jwa and Sonh	99	81
Arastounia 2015	97.1	96.4
Arastounia and Oude Elberink 2016	97.3	97.7
Arastounia 2017	97.4	97.5

Table 2.1: The accuracy and precision of each examined method of railway detection

We can conclude the following:

- The lack of exact numbers for both algorithm proposed by Neubert et al. in 2008 makes them an unappealing choice.
- The high variance for Beger et al. 2011 method is not very convincing either
- Yang and Fang's algorithm has high precision and accuracy, so it is a good choice

- Jwa and Sonh's methodology results in a high precision algorithm, but its completeness is only 81% which dismisses its potential usage.
- Arastounia's 2015 method is a good general algorithm, but given that the author's later work was based on improving this, those methods will be better
- Arastounia and Oude Elberink's method is one of the best performing algorithms but it comes with a cost of assuming there is no big altitude change in the point cloud.
- Arastounia's 2017 algorithm is an ideal option if it's used without the contact cable detection. A potential improvement can be further examined where we use Cserép, Hudoba, and Vincellér's [11] results, which give us the points of the overhead cables from the same dataset. With these points, the detection of the rail track may be optimized.

After the performance analysis I narrowed down the potential methods to two: Yang and Fang's method and Arastounia's 2017 algorithm. Arastounia's other works fit the criteria, but his 2017 method is simply more efficient.
For the final criteria I tried to use runtime data, but unfortunately it was only provided in Arastounia's publication, making Yang and Fang's method risky to use. The former algorithm ran for 30 minutes for a point cloud with more than 3 million points.

The conclusion is Arastounia's 2017 algorithm is the best to base upon my research.

Chapter 3

Datasets

The provided data was collected by Riegl VMX-450 high-density mobile mapping system (MMS) mounted on a railroad vehicle, which operated at 60 km/h. The sensor's average 3-dimensional range precision is 3 mm, has a maximum threshold of 7 mm and is capable of recording 1.1 million points/sec. The data contains approximately $1.54 * 10^9$ points and covers about 18.5 km long and 130 m wide of Hungarian rural railroad. The points have georeferenced spatial information (3 dimensional coordinates of the points) with intensity and RGB data. The original datasets are samples of this data and were created by cutting out 100m along one of the directional axis.

The original datasets can be seen in Figure 3.1 and in Figure 3.2.

As a preprocessing step, the wider surroundings of the rail tracks (containing mostly low and high vegetation, agricultural areas, etc.) were manually cropped around the railway resulting in the point clouds seen in Figure 3.3 and in Figure 3.4. The necessity of this preprocessing step is detailed in Chapter 4, while a possible automatization for this step is discussed in Chapter 7.

The first dataset is roughly 100 m of rural Hungarian railway which consists of more than 2 400 000 points. The second dataset contains around 200m of rural Hungarian railway with more than 6 300 000 points.

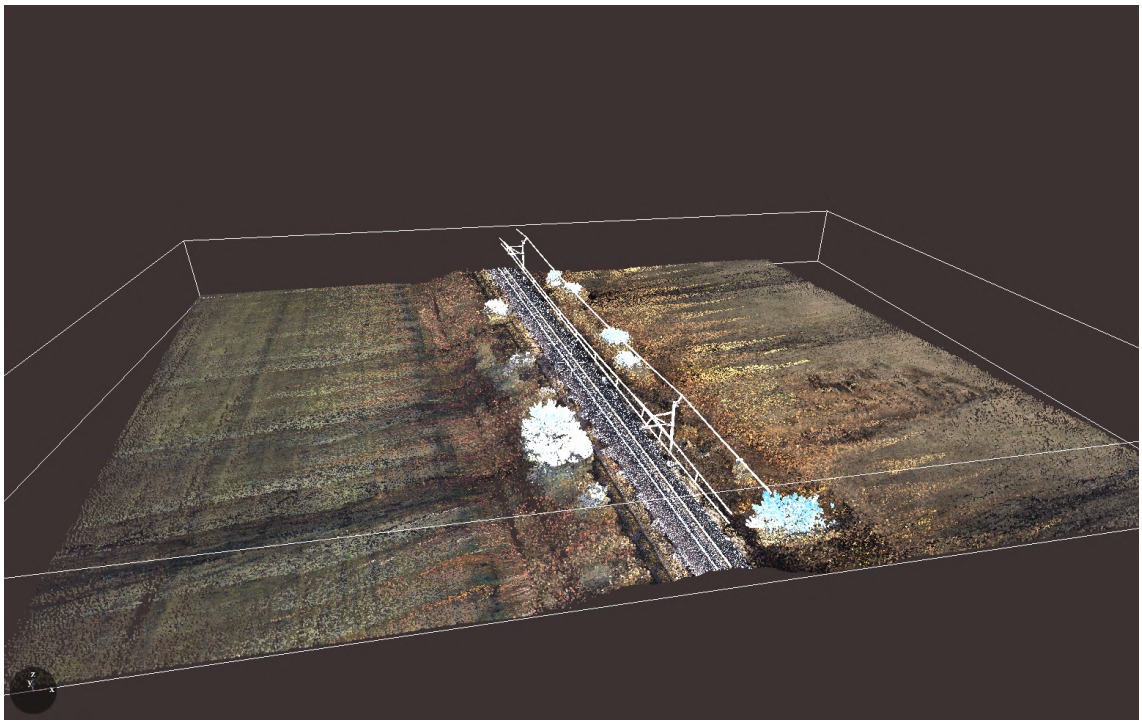

Figure 3.1: Original version of first dataset

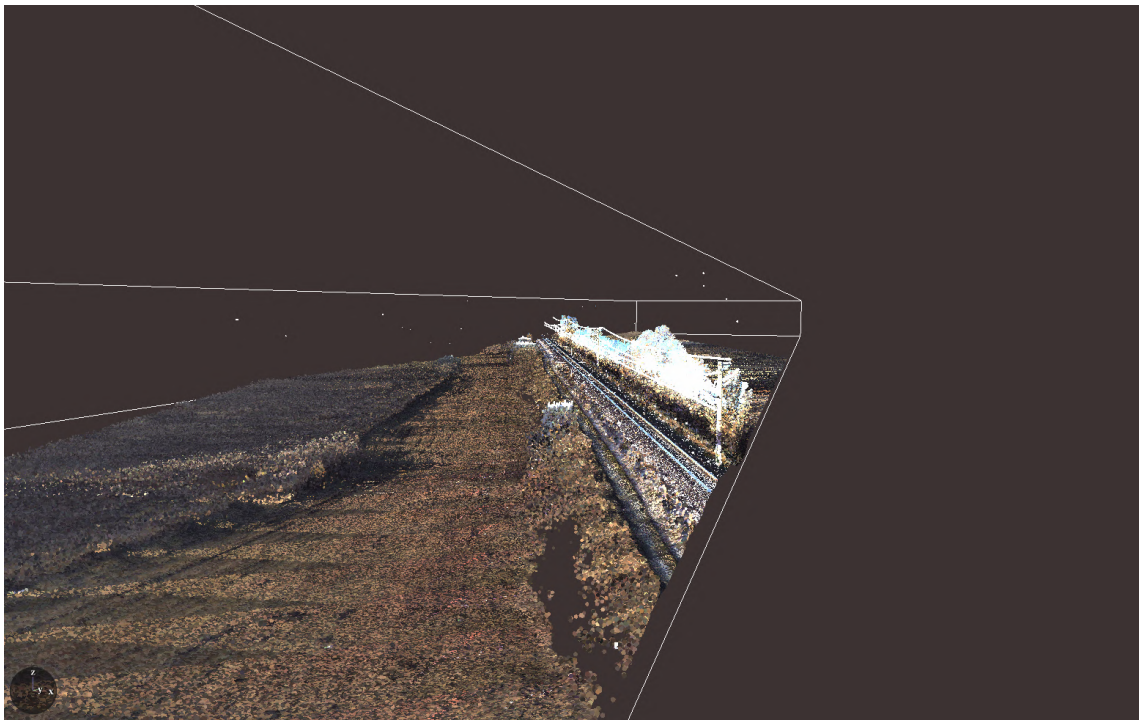

Figure 3.2: Original version of second dataset

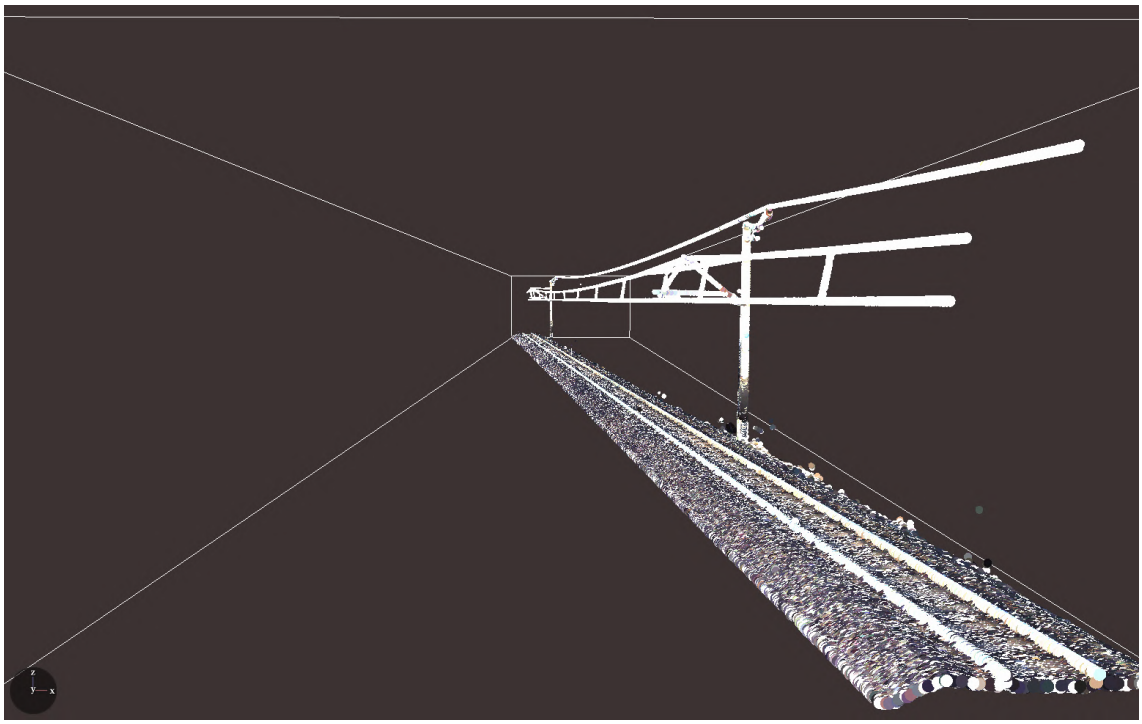

Figure 3.3: Final version of first dataset

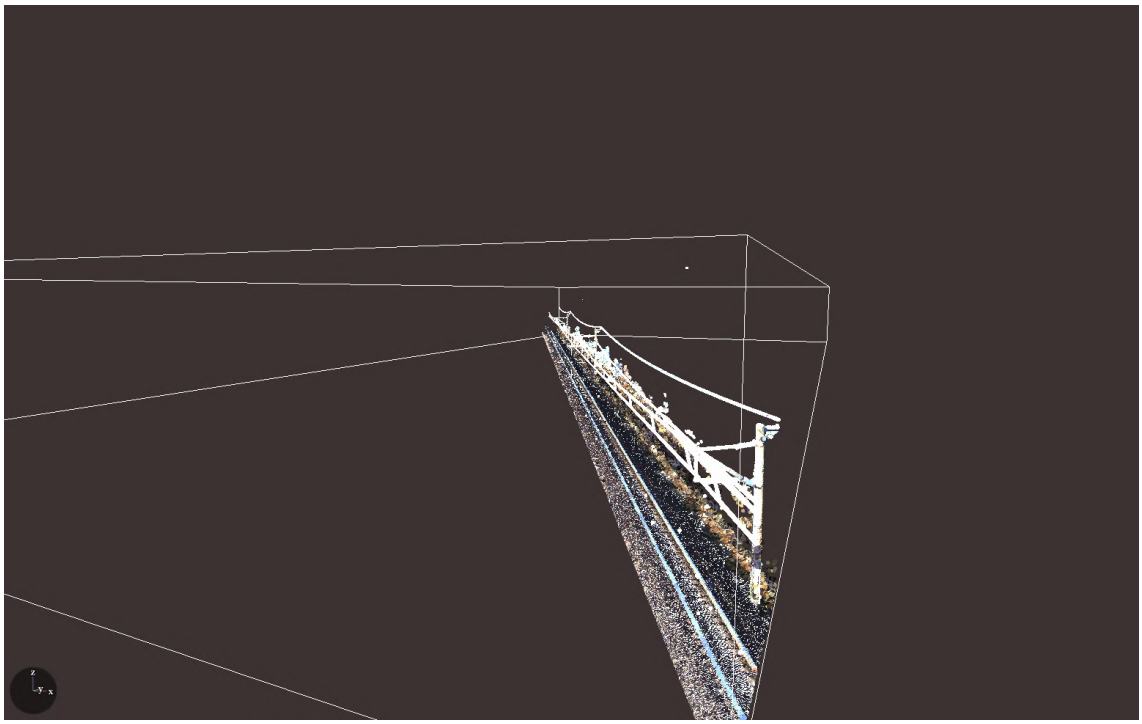

Figure 3.4: Final version of second dataset

Chapter 4

Methodology

My algorithm is an adapted version of Arastounia's proposed algorithm from 2017[10]. In this chapter, I will describe the developed algorithm, the differences from the original method and the causes of these modifications. The results of important steps will be shown using the first dataset throughout this chapter.

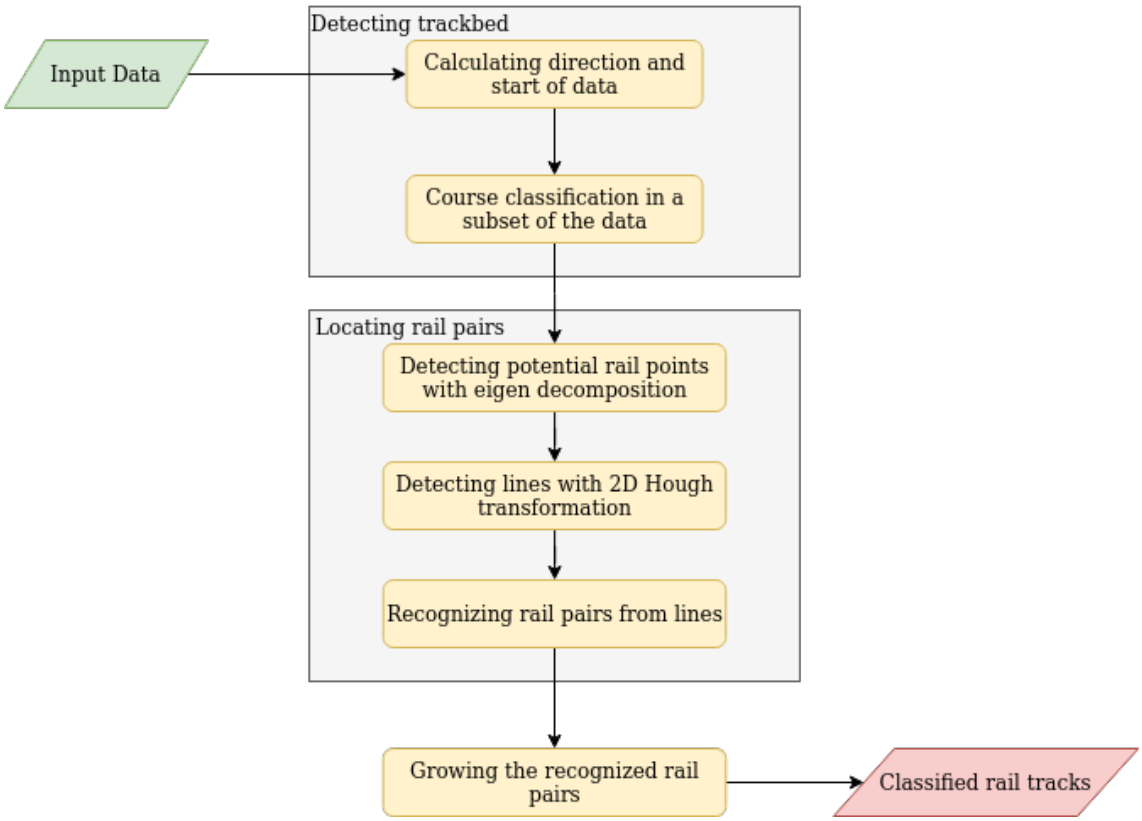

Figure 4.1: Flowchart of the developed algorithm

The developed algorithm consists of three main parts:

1. Locating the trackbed with classification on a small subset of the data
2. Detecting the rail pairs in that subset
3. Growing the rail pairs throughout the rest of the data

Figure 4.1 shows the flowchart that depicts the steps of the proposed algorithm.

4.1 Locating the trackbed on a subset

The location process of the trackbed can be split into two main parts: calculation of the railway direction axis and start coordinate which is described in subsection 4.1.1 and course classification on a subset of the cloud, this is further elaborated in subsection 4.1.2.

4.1.1 Calculating railway direction axis and start of dataset

The first step of the algorithm requires that we cut out a small portion of the dataset, in which we detect the rail pairs. From this, the first problem emerges, that without directional data - which is indeed not at our disposal -, we don't know where to cut the dataset.

The fact that our dataset is cropped (as explained in chapter 3) around the railway environment comes to help here. First, we search for the points that have the minimal x, minimal y, maximal x and maximal y coordinates. These points together create a 2D rectangle, where the sides represent an approximate length of the dataset in the direction of x and y coordinates. Because our dataset is cropped the coordinate axis by which our railway goes will have a larger length. Then the corresponding minimum value will be the start of the dataset.

With these two information at our disposal now we can successfully cut out a small portion of the dataset by the direction axis. In the algorithm, we cut out an approximately 5m section in the general direction for both datasets. It should be noted

that because we only have a general direction of the railway the cut will often result in rail pairs with unequal length, which will be important later on.

4.1.2 Course classification

We carry out the course classification based on the heights of the points in our cloud potion. In a railway environment, the object with the most points in it should always be the trackbed. So the height of the trackbed is found by searching for the most common height in our subset. The points that are within 0.75m of this height are classified as trackbed points. This is a modification of the original algorithm where the threshold is much stricter, only 0.5m. The reason for that is the fact that in our datasets the height variance of the trackbed points is much higher than Arastounia assumes in his study. Arastounia assumes that the trackbed is mainly flat, with very little variance, which is not the case in our datasets. Figure 4.2 clearly shows that in our second dataset, parts of the rail tracks are not even considered as part of the trackbed with the 0.5m threshold as opposed in the result with the 0.75m value, vid. Figure 4.3.

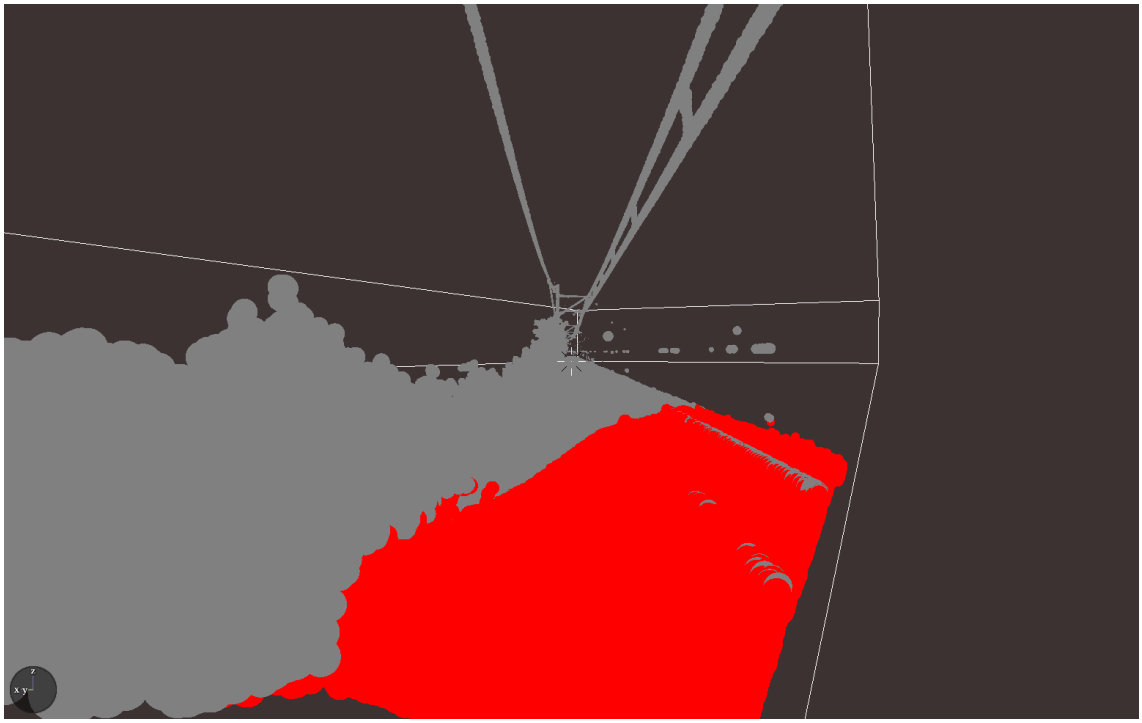

Figure 4.2: Trackbed of second dataset with 0.5m threshold

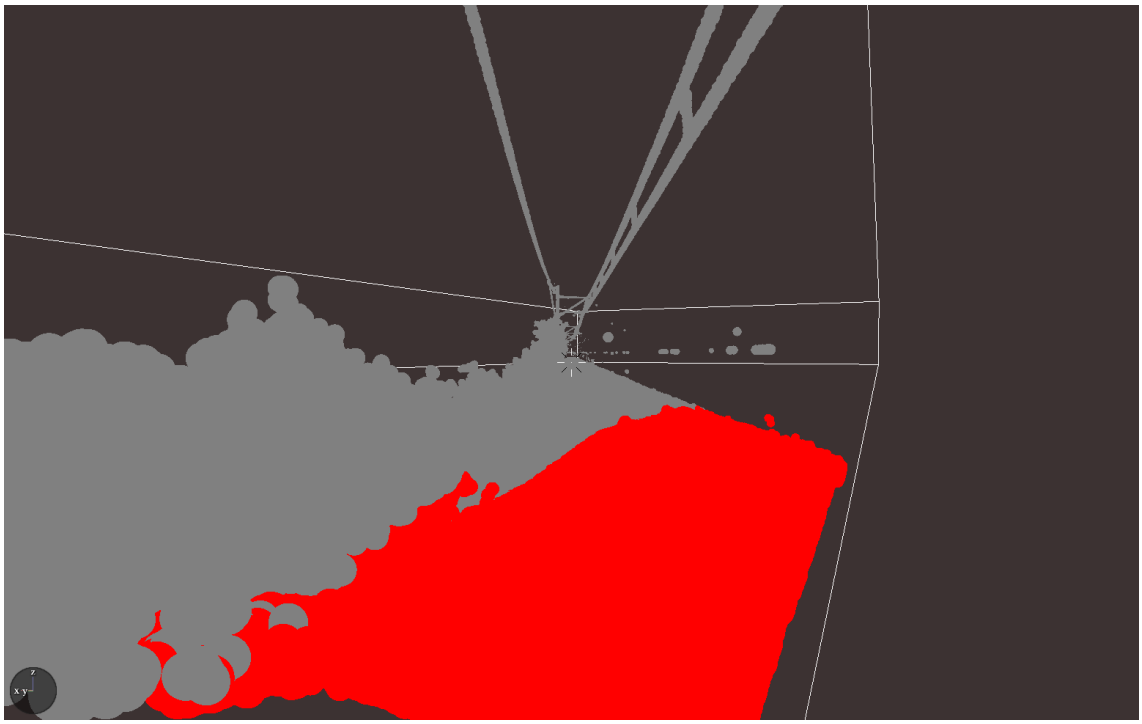

Figure 4.3: Trackbed of second dataset with 0.75m threshold

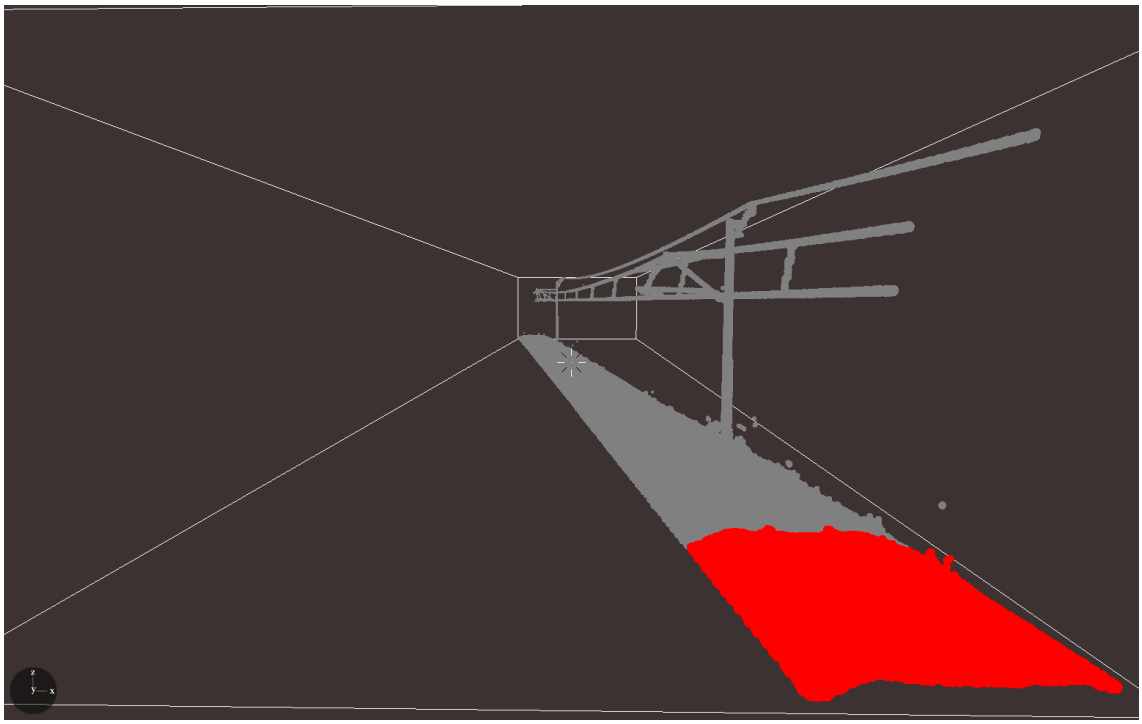

Figure 4.4: Trackbed of first dataset

The cut and course classification significantly improves the data processing in both accuracy and time, with the simple fact that it reduces the input data massively. In the 2 million point dataset, the resulting trackbed size was around 130,000 points and in the 6 million point dataset, the detected trackbed had 260,000 points. Figure 4.4 shows the detected trackbed of the first dataset.

4.2 Detecting rail pairs

Rail pair detection consist of 3 main steps:

1. Calculating candidate seed points as seen in subsection 4.2.1
2. Detecting lines as described in subsection 4.2.2
3. Recognizing rail pairs as detailed in subsection 4.2.3

4.2.1 Calculating candidate rail seed points with eigendecomposition

What we are looking for in our trackbed are points which are outliers in their respective local neighbourhoods. This comes from the fact that rail tracks by definition are narrow and relatively high objects. For this task the algorithm has the following steps given p point of the trackbed:

1. Calculate p's local neighbourhood, let that be N_p
2. Calculate N_p's covariance matrix C
3. Apply eigendecomposition to C
4. Classifying candidate rail seed points

Calculating a point's local neighbourhood

For effective neighbourhood calculation, we use a data structure called octree, which is a tree data structure in which each internal node has exactly eight children. Its use was pioneered by Meagher [12] and it can be used for spatial representation of

3D point clouds. Each node in the octree subdivides its represented space into eight octants, basically, octrees split around a point. One of its advantages of using an octree that computing a neighbourhood radius of a point has $log(n)$ computational complexity, where n is the size of the point cloud. The only question remaining was the resolution of the octree we use. I tried out several resolutions, the results can be seen in table 4.1. It depicts the time all neighbourhood search took in the respective datasets and how many points is detected as rail track in the end result of the algorithm.

Value	First dataset	Second dataset	First result size	Second result size
0.25 m	3.2 s	7.2 s	0	0
0.5 m	5.3 s	13 s	66822	137909
0.75 m	8.5 s	20.8 s	67305	138162
1 m	12.7 s	28.5 s	67305	138143
2 m	35.3 s	63.4 s	67305	138068
4 m	75.8 s	163.9 s	67305	137525
8 m	97.9 s	312.9 s	67305	0

Table 4.1: Runtime of the neighbourhood search with different resolution values

From the table, we can see that the optimal resolution is 0.75, any lower value worsens the accuracy and higher values only add to the runtime.

Calculating covariance matrix of a neighbourhood

We construct a covariance where each element will represent the variance or covariance along its respective cardinal directions. The structure of the matrix is the following:

$$C = \begin{bmatrix} Var(x) & Cov(x,y) & Cov(x,z) \\ Cov(x,y) & Var(y) & Cov(y,z) \\ Cov(x,z) & Cov(y,z) & Var(z) \end{bmatrix} \tag{4.1}$$

where

$$Var(a) = \frac{\sum_{i=1}^{n}(a_i - \overline{a})^2}{n-1} \tag{4.2}$$

and

$$Cov(a,b) = \frac{\sum_{i=1}^{n}(a_i - \overline{a})(b_i - \overline{b})}{n-1} \tag{4.3}$$

with n being the size of the neighbourhood.

Eigendecomposition of the covariance matrix

The eigenvalues and eigenvectors are calculated by the equation shown below:

$$CV = \lambda V \tag{4.4}$$

which gives us

$$(C - \lambda I)V = 0 \tag{4.5}$$

therefore

$$det(C - \lambda I) = 0 \tag{4.6}$$

where C is the covariance matrix, V is the matrix of the eigenvectors, λ representing the matrix of eigenvalues and I being the identity matrix.

The covariance matrix has 3 eigenvalues given its size, with each representing the dispersion magnitude of the neighbourhood along its associated eigenvector. The highest eigenvalue represents the highest dispersion while the lowest denotes the lowest dispersion.

Classifying candidate rail seed points

The trackbed is usually constructed to have the smallest height variation possible in the longitudinal direction because of the safety regulations. Therefore the smallest eigenvalue of a local neighbourhood without a rail piece should be

zero. While in theory, it would be safe to say that every neighbourhood where the smallest eigenvalue is non-zero contains a piece of rail, in reality, there are false positives due to noise and the trackbed's covering material. Despite that, all neighbourhoods that have its smallest eigenvalue being higher than 0.01 is considered as containing a piece of rail.

Then we iterate through these neighbourhood's points and all points that are higher than the trackbed's 95th percentile are considered as rail points.
There are two major deviations here from the original algorithm:

1. Calculating the height percentile of the trackbed, not the local neighbourhood
2. Calculating the 95th percentile instead of the 90th

Calculating the overall height percentile instead of local

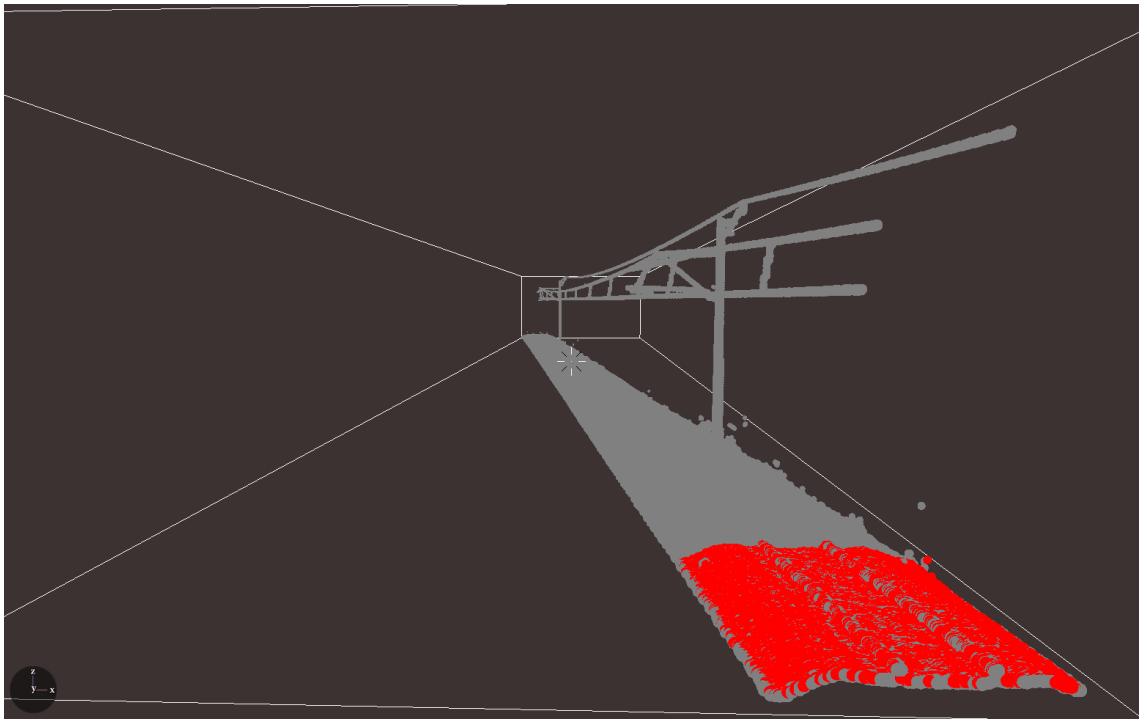

Figure 4.5: Results with local neighbourhood's height percentile

Arastounia's proposition of calculating each local neighbourhood's height percentile for the second criteria only works properly if the trackbed has small height deviation. As mentioned before in subsection 4.1.2 in our dataset that is not the

case. The result given by the original approach can be seen at Figure 4.5. From that result the next step - line detection - is impossible. So I modified the algorithm to calculate the whole trackbed's height percentile. The downside is if the rail has a bigger slope the result won't contain the whole rail track in the cut, but as long as we find a pair the growing algorithm will take care of detecting the missing parts. The result of this approach can be seen in Figure 4.6.

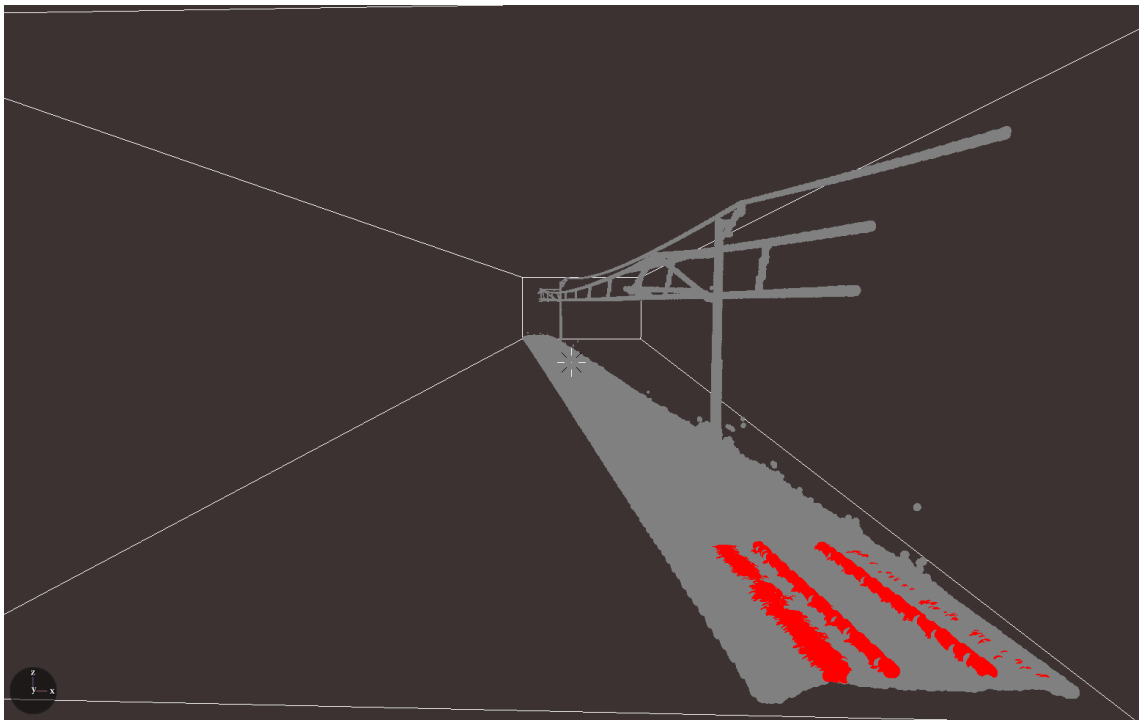

Figure 4.6: Results with overall height percentile

Calculating the 95th percentile

This change was made because in the second dataset it became apparent that the height variance in the trackbed makes the 90th percentile too low of a criterion and a lot of neighbouring grass is getting picked up as rail piece. These then successfully broke the growing algorithm because the noise they gave constantly grew with each iteration. Figure 4.7 and Figure 4.8 show the detected rail pair and the failing growing algorithm after 5 iterations.

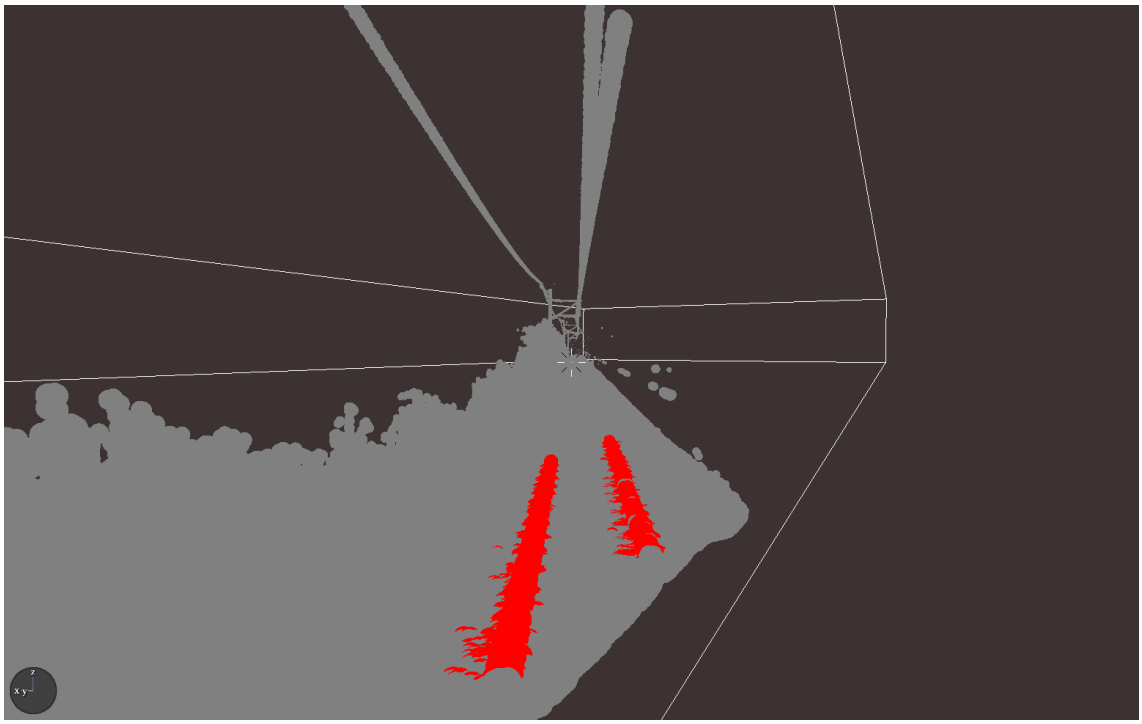

Figure 4.7: Railpair with 90th height percentile

Figure 4.8: The failed growing algorithm after 5 iterations

The original datasets mentioned in Chapter 3 contain a large part of the railway environment which was proven to provide too much noise for the proposed algorithm to work. In Figure 4.9 we can see the calculated candidate rail seed points, which doesn't even include the rail tracks.

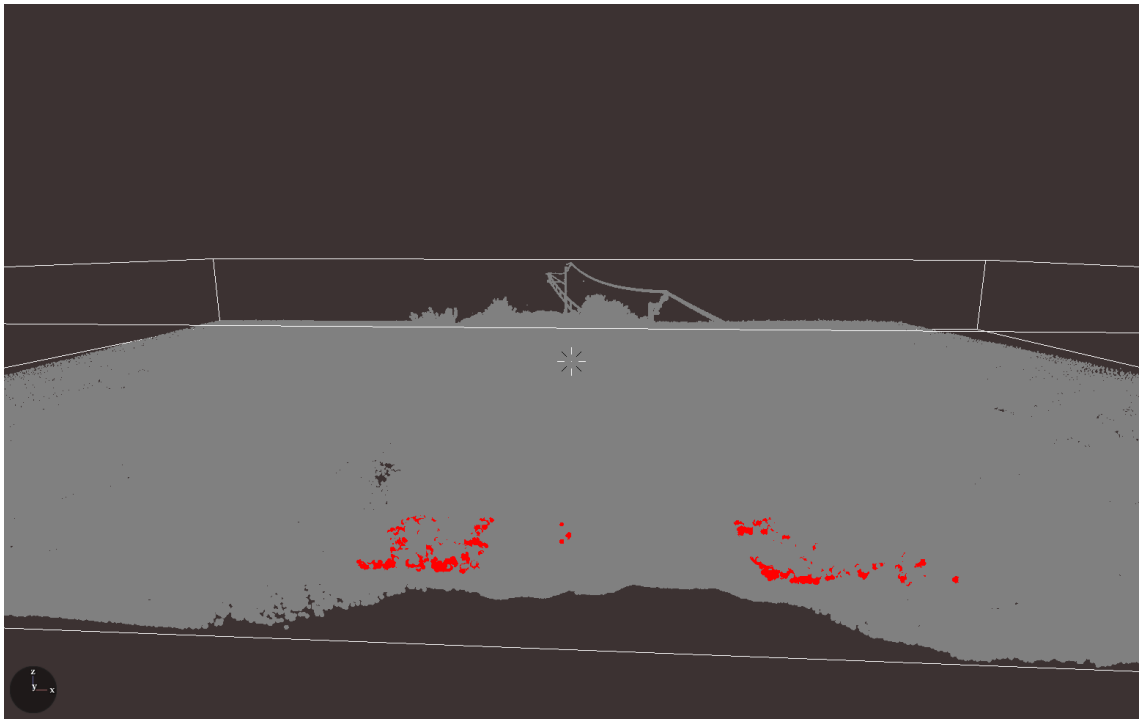

Figure 4.9: Candidate rail seed points on original first dataset

It is worth mentioning here that the necessity of the cut in the dataset can be proven easily by applying this calculation to the whole dataset. After that, we can conclude it is a lot faster to detect small rail pairs and grow them, then trying to detect the whole rail at once. As seen in table 6.1 in chapter 6 the latter took more than 2 minutes in the first dataset, which is at least 3 times slower than this method as a whole. Also as seen in Figure 4.10, the height difference throughout the rail causes a lot of accuracy issues, which implies the need of even more calculations in the approach that already has a longer runtime.

The final candidate rail seed points in the first dataset are shown in Figure 4.11.

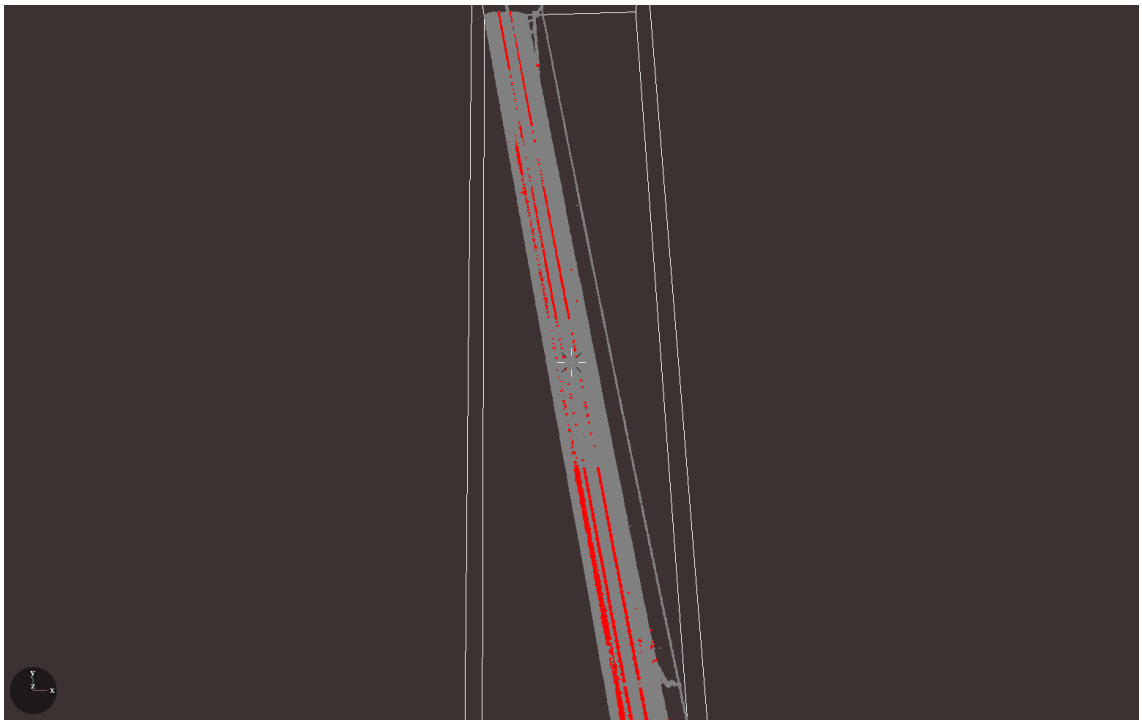

Figure 4.10: Candidate rail seed points calculated for the whole first dataset

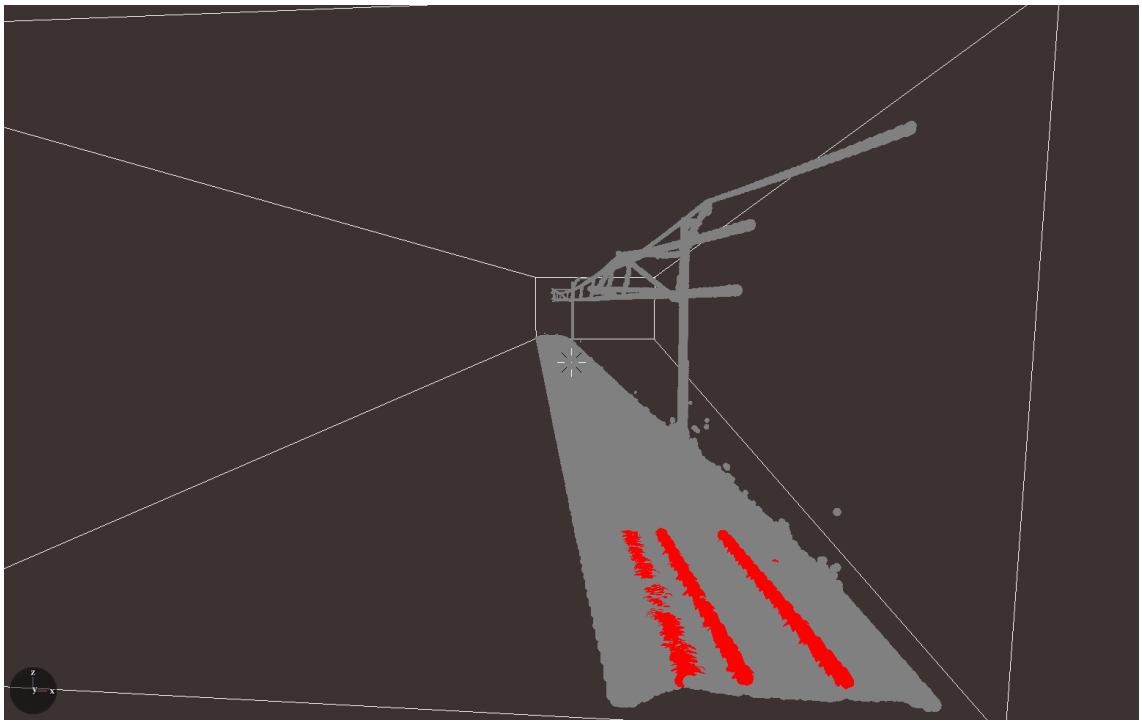

Figure 4.11: The final candidate rail seed points in the first dataset

4.2.2 Detecting lines with 2D Hough transformation

Hough transformation[13] is an iterative algorithm which is used to detect complex patterns - including lines - in 2D images. In the Polar coordinate system a line can be represented by the following equation:

$$y = -\frac{cos\theta}{sin\theta} \cdot x + \frac{r}{sin\theta} \tag{4.7}$$

This can be arranged as

$$r = x \cdot cos\theta + y \cdot sin\theta \tag{4.8}$$

For each point (x_0, y_0) exist a family of lines that goes through this point, defined as:

$$r_\theta = x_0 \cdot cos\theta + y_0 \cdot sin\theta \tag{4.9}$$

where (r_θ, θ) represents the different lines.

If we plot these lines in a plane θ - r given (x_0, y_0) point, we will get a sinusoid. We do this for all points and every intersection between sinusoids will mean that the points whom those sinusoids belong are on the same line. With this fact, we can now find lines by finding the number of intersections between these curves. All we need to do is to set up a threshold, which represents the number of intersections over which we acknowledge a line.

In our algorithm, the first task is to convert our 3D point cloud of candidate rail seed points to a 2D image. For this purpose, we will use the projection filter introduced and implemented by Cserép, Hudoba, and Vincellér.[11].

The only downside of the Hough line transformation is that it is very dependent on the threshold given. That means there is a high chance that the same threshold won't give appropriate results for two different datasets. For that instance the algorithm does the following steps:

1. Sets the threshold to a high number.

2. Runs the Hough transformation on the image.

3. If the Hough transformation didn't give at least two lines, it lowers the threshold.

4. Repeats steps 2. and 3. until at least two lines are found or the threshold reaches zero.

Figure 4.12: Result of Hough transform in first dataset in 2D (zoomed in)

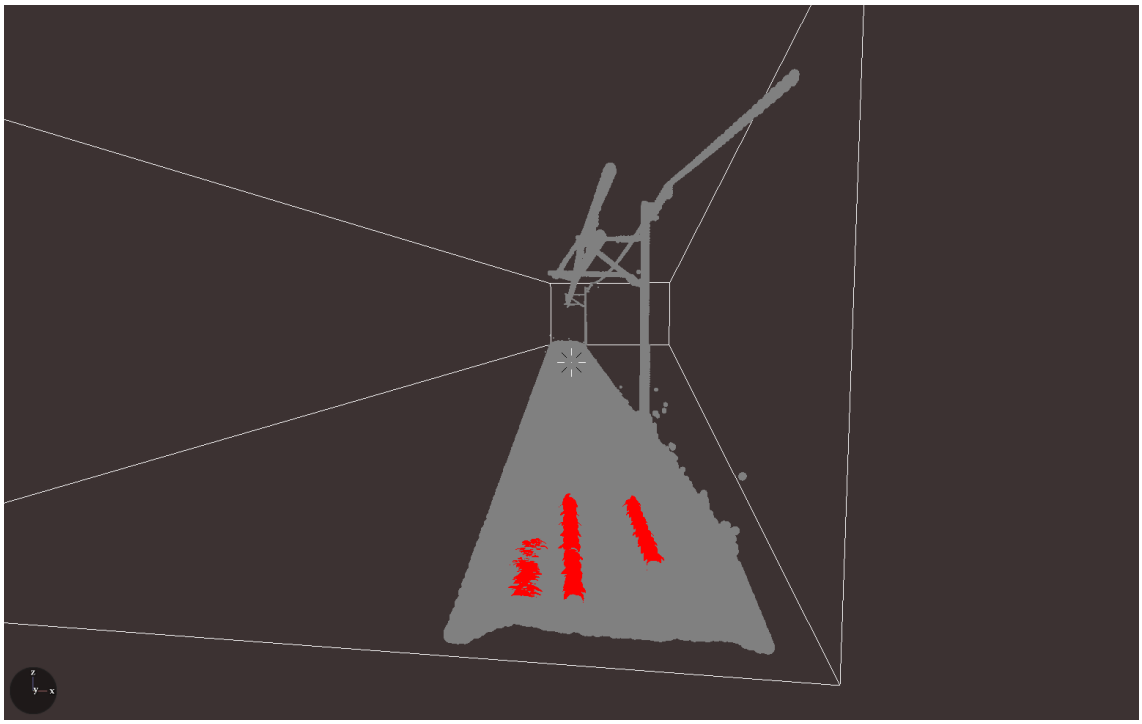

Figure 4.13: Result of Hough transform in first dataset in 3D

If the Hough transform was executed successfully we now convert the 2D image back to 3D. The results of the Hough transform in the first dataset are shown

below. Figure 4.12 shows the original projection and the detected lines next to it (zoomed up for visibility) and Figure 4.13 shows the result in 3D.

4.2.3 Recognizing rail pairs

For detecting the rail pairs from the lines we use on our knowledge of railway standards:

- The rail tracks of a rail pair has a set distance from each other, called the track gauge
- The rail tracks of a rail pair follows the same direction

From these we can construct the following criterias:

$$\angle \vec{d_1}\vec{d_2} \leq 5^\circ \tag{4.10}$$

$$|DistanceBetweenLines - Gauge| \leq 0.05m \tag{4.11}$$

The vectors d_1 and d_2 are the direction vectors of the lines calculated from the start and end points given by the Hough transform. Arastounia denotes the distance between the lines as the distance between the centroids of the lines. Unfortunately in our case that would lead us to a incorrect result, because our cut of the dataset doesn't ensure that these lines are at equal length. So we use the following equation:

$$distance(x_1, y_1, x_2, y_2, x_0, y_0) = \frac{|(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}} \tag{4.12}$$

where (x_1, y_1) and (x_2, y_2) are points of the first line and (x_0, y_0) is a point of the second line.

We can test all line combinations against the criteria and those fit them will be recognized as rail pairs. All other lines will be dismissed. The 2D-3D conversion results in a lot of duplicate points so before the growing step, we go through all recognized rail pairs and get rid of these duplicates.

The recognized rail pair in the first dataset is shown in Figure 4.14.

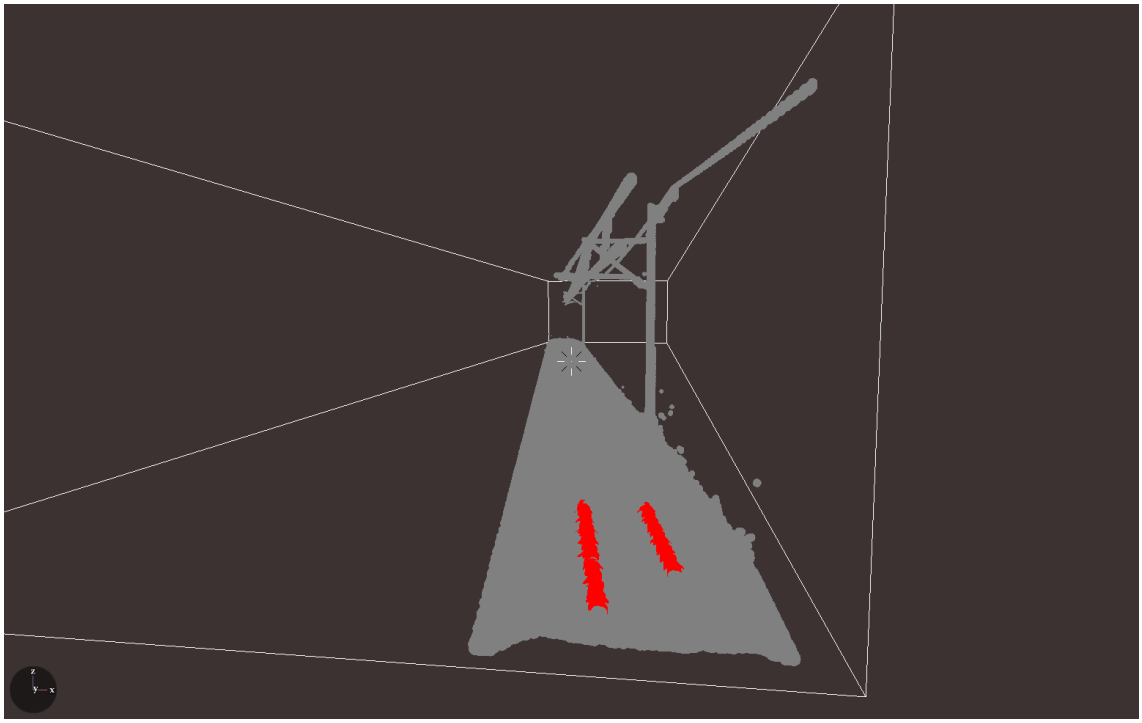

Figure 4.14: The detected rail pair in the first dataset

4.3 Growing the rail pairs

The final step of the method is growing the recognized rail pairs. For this, an iterative algorithm was implemented which fully grows its input rail pair. We construct two criteria what each point has to meet in order to become a candidate rail point. These are the following:

$$|H_{rail} - H_p| \leq 0.05m \tag{4.13}$$

$$\angle \vec{v}_{raildirection} \vec{v}_p \leq 5^\circ \tag{4.14}$$

H_{rail} depicts the average height of the current segment we grow, H_p is the height of the point, $v_{raildirection}$ is the direction vector of the rail and v_p is the vector connecting the point to the current rail segment.

In Section 4.2.3 we got the directional information from the Hough transform's results, however, in this case, we don't have that type of data at our disposal. In

Arastounia's method, the direction vector is calculated by another eigendecomposition, but in our datasets, the result vectors were often null vectors which made it very unreliable for our calculations. Instead of using eigendecomposition, the method is trying to fit a 2D rectangle on the rail segment, given that rail tracks in the 2D image should shape us a rectangle. For this, we calculate the points in the point cloud, that have the minimal X, minimal Y, maximal X and maximal Y values. As you can see in Figure 4.15 these will serve as our rectangle's vertices.

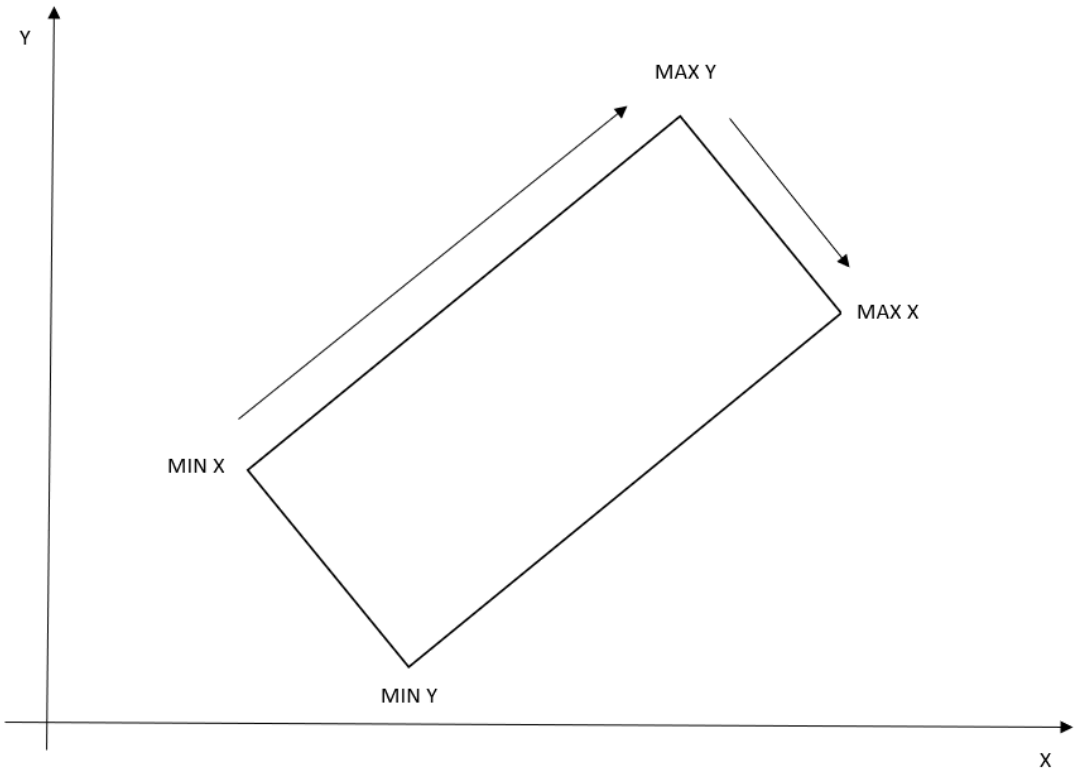

Figure 4.15: Example of the 2D rectangle which we fit on a rail segment

It is obvious that the rail segment's direction lies along with one of the sides, our only task now is to decide which one of the two is the correct answer. Railway standards come to our help again: a rail track's standard width is between 10 to 20 millimetres, so as long we choose a grow size bigger than 20 millimetres, the direction will be always along the longer side. So we compare the length of the two sides and calculate the direction vector from the endpoints of the longer one.

To grow a rail segment we do following for each p point in it:

1. Calculate the local neighbourhood N_p with the radius being our grow size.
2. Recognizing candidate rail seed points from N_p

4.3.1 Calculating local neighbourhood

For our neighbourhood calculation we once again use the octree data structure, which will cover the whole dataset. The chosen grow size is 1 meter for both datasets. Table 4.2 shows the runtime of the different resolutions tested (without threading) and if it could succesfully grow through the whole dataset:

Value	First dataset	Second dataset	Fully grown(First)	Fully grown(Second)
0.0625 m	20.5 s	41.1 s	No	No
0.125 m	20.1 s	28.2 s	No	No
0.25 m	24.2 s	38.2 s	Yes	Yes
0.5 m	38.2 s	73 s	Yes	Yes
0.75 m	43.2 s	84.2 s	Yes	Yes
1 m	45.1 s	98.8 s	Yes	Yes
2 m	84 s	137.4 s	Yes	Yes
4 m	140 s	283.6 s	Yes	Yes

Table 4.2: Runtime of neighbourhood search during the growing algorithm with different resolution values

From the results it was concluded that the optimal resolution for the growing algorithm is 0.125. In both lower values, a larger part of the end of the rail was not detected.

4.3.2 Recognizing candidate rail segment points

If a point fits all the criteria given in equation 4.13 and equation 4.14 then we check if it's already part of the rail. This is important because if we include already recognized points the following problems will occur:

- The algorithm will become very slow because the input point size will gradually grow with the number of duplicate points.

- The algorithm will never stop because it can always detect a newly grown segment

However instead of checking the point against the whole detected rail, we only test against the result of the last 3 iterations of our algorithm. This will enhance the performance of the algorithm as the size of the point cloud we check against will be kept low.

A small setback I ran into in this stage is calculating the vector angle. Initially, I calculated the angles in 3D dismissing the noise that the height information will give. The error this resulted can be seen in Figure 4.16.

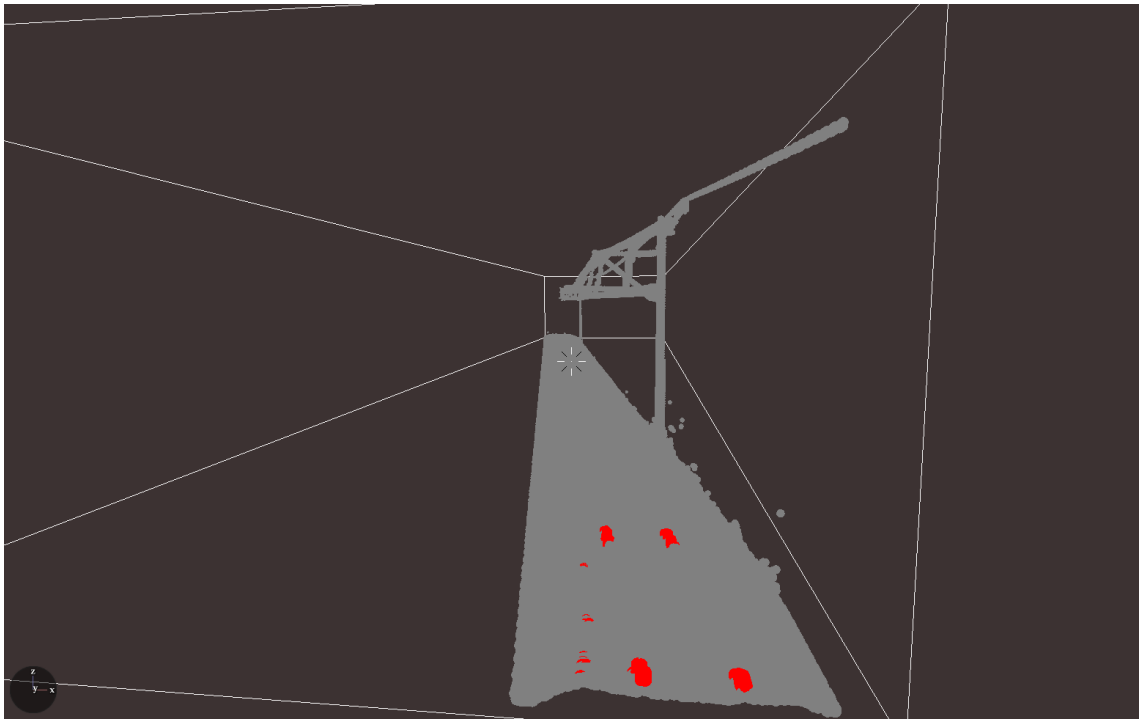

Figure 4.16: A grow with 3D angle calculation in the first dataset

Taking a closer look at the equation given for 3D angle calculation, my error became evident, but it was still fascinating to see that calculating the angle of two points that has 0 height - essentially 2D - and calculating the angle of those points at 100m height would give different results, cause without looking at the equations I would have never assumed that. After switching to the 2D angle calculation these errors disappeared as seen in Figure 4.17.

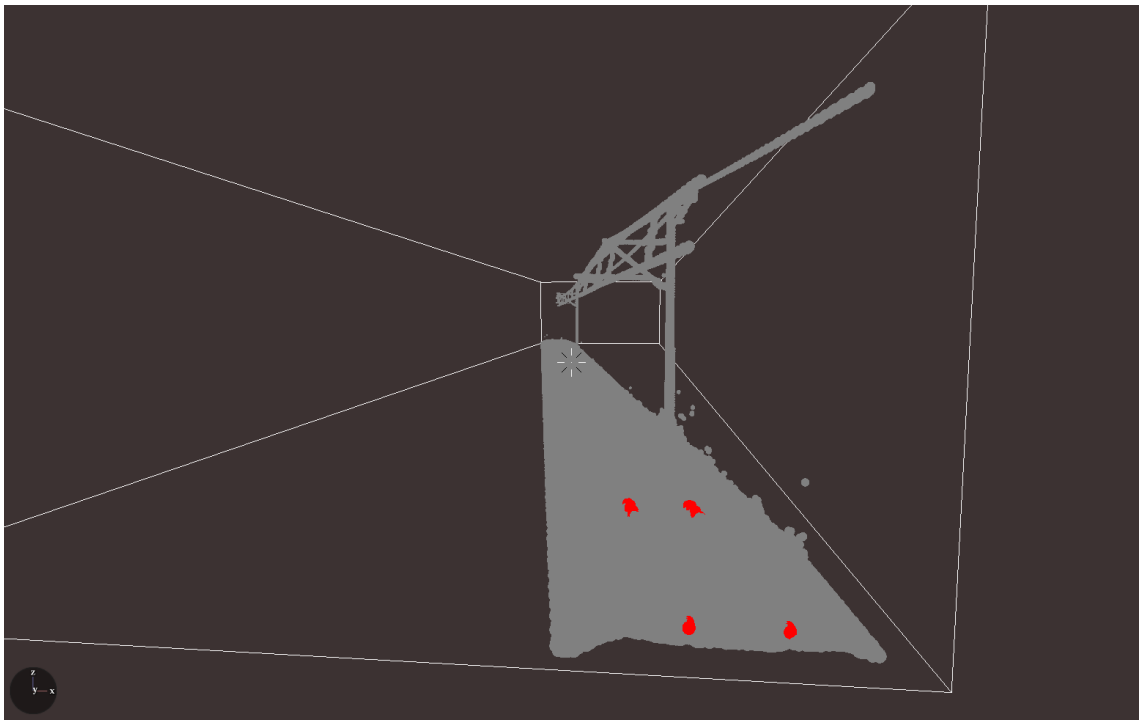

Figure 4.17: Same grow with 2D angle calculation in the first dataset

A more interesting problem presented itself when I noticed that the segments supposed length started to grow. What that means that the newly grown segments started the have points that were out of the supposed neighbourhoods' range, points were detected along the already detected rail track. This can be explained by the slope of the rail. As shown in Figure 4.18 points can be detected along already detected segments. While growing Segment 1, point p was not detected cause its height was too low. However Segment 2 has a lower average height, so p now meets our criteria. When growing Segment 2 we will check p's neighbourhood as well which will lead us to more and more falsely detected points. To prevent this happening, after a segment is detected an outlier detection is executed which will remove these points from the segment cloud.

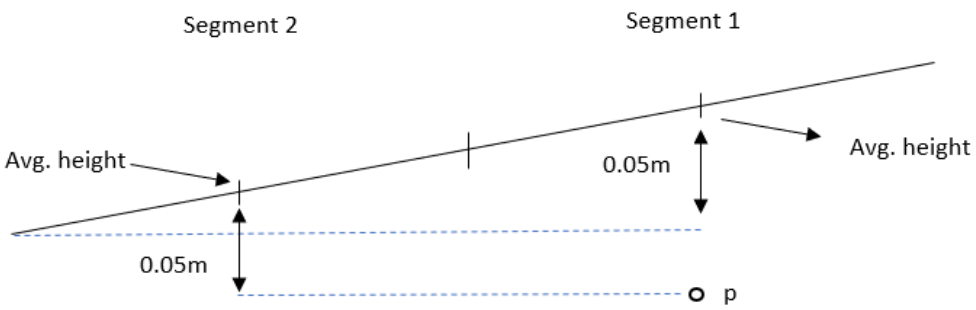

Figure 4.18: Example of backward point detection caused by slope

After growing both tracks of pair we check if the candidate rail segments fit for the following criteria:

$$\angle \vec{v}_{segment1} \vec{v}_{segment2} \leq 10^{\circ} \tag{4.15}$$

In the original method, the degree threshold is at 5 and also checks the distance of the detected rail segments. In my method, due to the occasional inaccuracy of the direction vector calculation, the distance calculation was omitted and the degree threshold was raised to 10. This, unfortunately, means the method will not work if the rail segment contains a railway switch.

The algorithm then runs again on the newly found segments until no segment is found or the criteria set in equation 4.15 fails. The inequality of the original rail pairs may result in that one of rail track's end will not be detected because the algorithm will reach the end of the other rail track faster and stops. The results of the final version can be seen in Figure 4.19 and in Figure 4.20.

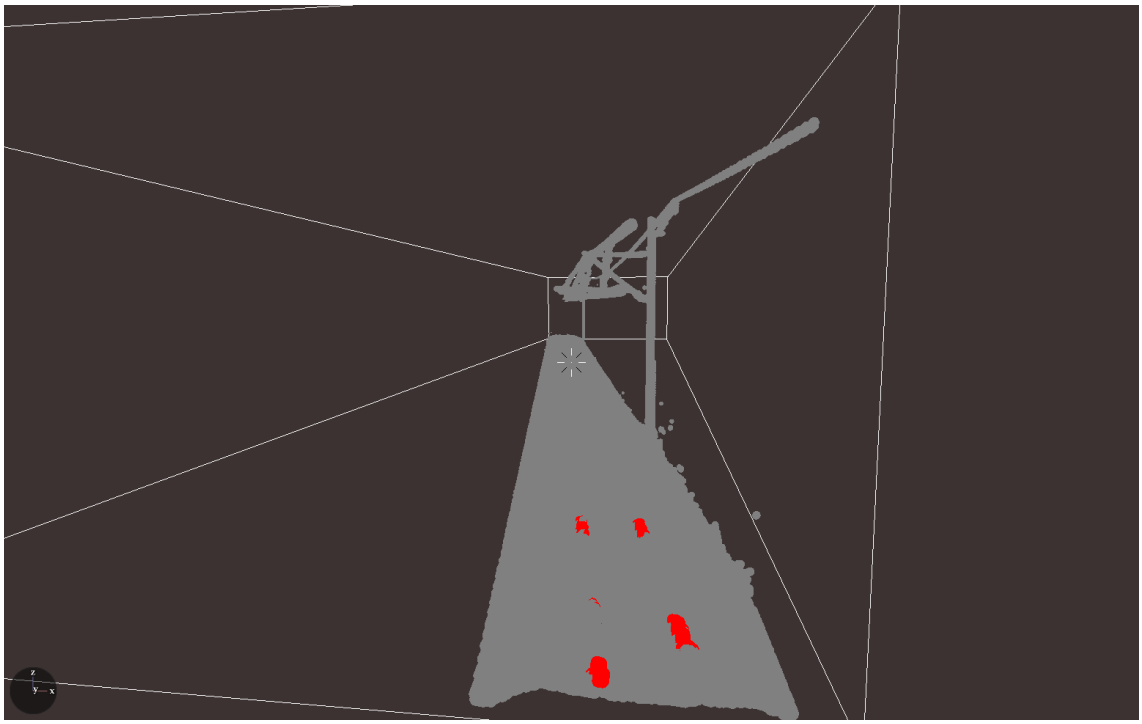

Figure 4.19: The first iteration of the growing algorithm in the first dataset

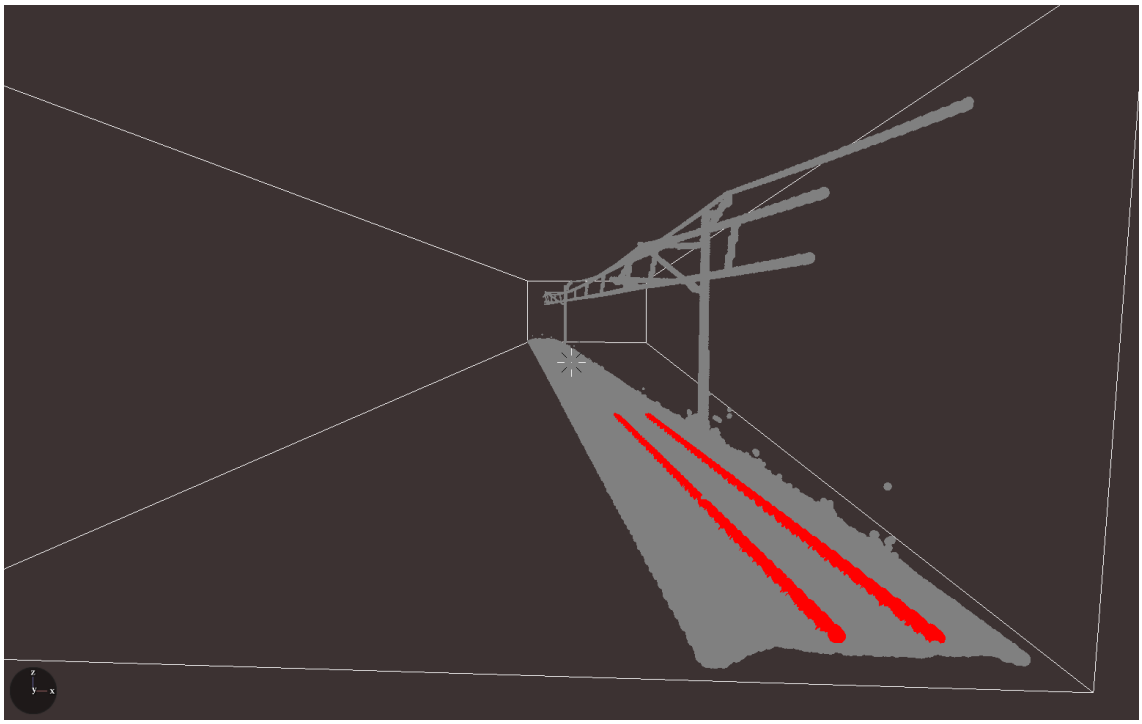

Figure 4.20: Growing until 20m in the first dataset

Chapter 5

Implementation

The implementation was made as part of an already implemented framework called *railroad*, described in the publication of Cserép, Hudoba, and Vincellér [11]. This framework allows to load and save point clouds, save point clouds as 2D images, between steps and at the end as well. For performance reasons, the implementation was written in C++11, with the usage of open source libraries. OpenCV [14] was used for the projections and the 2D Hough transformation. Point Cloud Library(PCL) [15] was used for storing the point clouds and it provided the octree data structure with the radius neighbourhood search as well. LasLib and LasZip were used for handling the point cloud files. For 3D visualization Displaz[2], for the cropping of the datasets LasTools [16] was used. The algorithm was implemented as a single filter within the framework.

5.1 Code Availability

The source code of the program is an attachment of the thesis. It can also by viewed online at the `https://github.com/mcserep/railroad` GitHub repository.

[2] `http://c42f.github.io/displaz/`

Chapter 6

Results and conclusion

6.1 Results

The results of the algorithm can be shown in Figure 6.1 and in Figure 6.2, with the former being the first dataset and the latter being the second dataset.

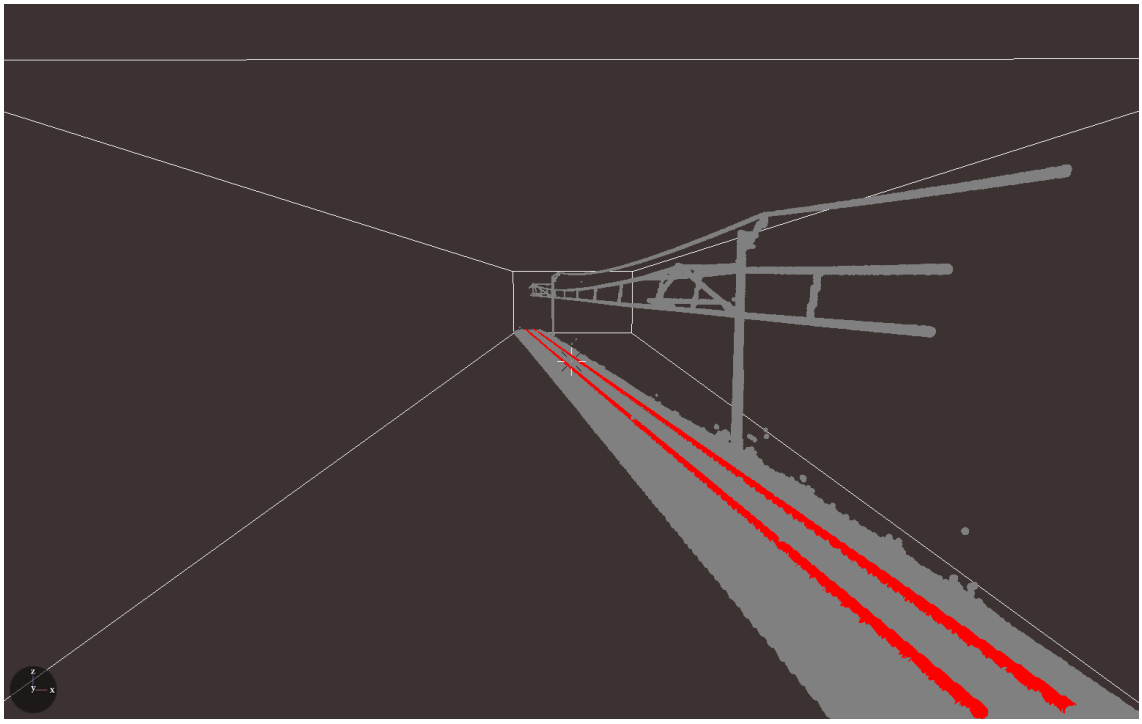

Figure 6.1: Detected rail track of first dataset

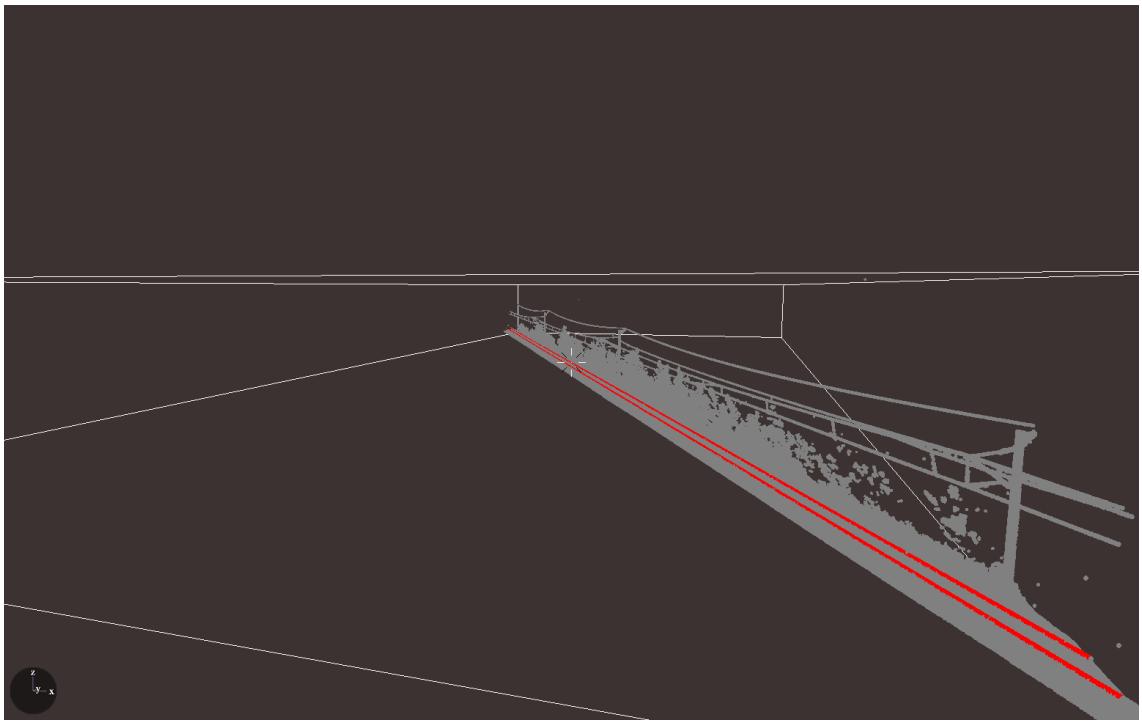

Figure 6.2: Detected rail track of second dataset

The algorithm was ran on Lenovo Y700-15ISK laptop with the following specifications

- CPU: Intel Core i7-6700HQ 2.60GHz
- RAM: 24 GB
- OS: Ubuntu 19.10

Threading was implemented in the growing algorithm, both tracks are grown concurrently. The algorithm's runtime can be seen in Table 6.1 where we can also see the runtime without threading and the runtime of applying the eigendecomposition on the whole dataset.

Dataset	Proposed Method	Method without threading	Eigendecomposition on whole
First	36 s	61 s	140 s
Second	77 s	124 s	396 s

Table 6.1: Runtime of the various types of the algorithm

For comparison, the Arastounia's proposed method ran for approximately 30 minutes on a dataset containing more than 3 million points.

6.2 Conclusion

Automated detection of railway tracks based on 3D LiDAR can be both done accurately and fast. The study uses a small subset of the given dataset to detect the trackbed with height classification, then uses eigendecomposition and Hough transformation for detecting the rail pairs in the subset. In the final step, the detected rail pair is grown throughout the whole dataset with help of a growing algorithm which results in the detected rail track for the dataset. The method works for datasets with high trackbed height variance and with slopes that are within safety regulations. The proposed algorithm has its restrictions, it cannot handle railway switches and the data has to be cropped around the railway environment, but these can be eliminated in future studies.

Two datasets of Hungarian rural railway were used in this study. The first dataset covers 100m of rural railway which was collected by a VLS system and contains more than 2 million points. The second dataset covers 200m of rural railway which also collected by a VLS system and consists of more than 6 million points. The algorithm detected the rail track in the first dataset in 36 seconds and needed 77 seconds to detect the rail track in the second dataset. The proposed method in my thesis is significantly faster than any of the examined state-of-the-art solutions. It can be utilized as a pre-filter for other algorithms but also can be improved into a potentially 100% accurate algorithm with a fast runtime.
The potential improvements are detailed in Chapter 7.

Chapter 7

Future work

The algorithm can be further enhanced with the following improvements.

More threading

The algorithm can be made more concurrent by implementing threading in the following parts:

- Calculating the rail seed points
- Growing a rail segment

In both parts, the calculations for the points can be done concurrently with the usage of a proper mutex. Efforts were made to achieve this, but the results were too inconsistent and it had to be reverted.

Automatizing the cropping of the dataset

The cropping of the original datasets were made manually in LasTools. An automatization can be proposed for that if we obtain the rail's trajectory. A potential way to obtain this would be using Cserép et al.'s[11] results on the same dataset. That would give us the cable points which follow the rail's trajectory. With the knowledge of the trajectory cropping the dataset to a certain width would be easily done, given in VLS data the railway is always in the centre. This can be implemented as a filter in the railroad framework and run before my implemented filter.

Other improvements with trajectory knowledge

With directional information, we wouldn't have to lower our criteria when checking two grown rail segment and we could reintroduce the distance criteria which would help in detecting railway switches.

The cut also could be done more precisely which would make the algorithm more accurate.

Acknowledgement EFOP-3.6.3-VEKOP-16-2017-00001: The research behind this master thesis is supported by the Hungarian Government and co-financed by the European Social Fund.

Bibliography

[1] Marco Neubert et al. "EXTRACTION OF RAILROAD OBJECTS FROM VERY HIGH RESOLUTION HELICOPTER-BORNE LIDAR AND ORTHOIMAGE DATA". In: (Aug. 2008).

[2] Martin A. Fischler and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24 (June 1981), pp. 381–395. DOI: 10.1145/358669.358692.

[3] Reinhard Beger et al. "Data fusion of extremely high resolution aerial imagery and LiDAR data for automated railroad centre line reconstruction". In: *Isprs Journal of Photogrammetry and Remote Sensing - ISPRS J PHOTOGRAMM* 66 (Dec. 2011). DOI: 10.1016/j.isprsjprs.2011.09.012.

[4] Jong-Sen Lee. "Digital image smoothing and the sigma filter". In: *Computer Vision, Graphics, and Image Processing* 24 (1983), pp. 255–269. DOI: 10.1016/0734-189X(83)90047-6.

[5] W Rieger et al. "Roads and buildings from laser scanner data within a forest enterprise". In: *International Archives of Photogrammetry and Remote Sensing* 32 (Jan. 1999).

[6] Bisheng Yang and Lina Fang. "Automated Extraction of 3-D Railway Tracks from Mobile Laser Scanning Point Clouds". In: *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of* 7 (Dec. 2014), pp. 4750–4761. DOI: 10.1109/JSTARS.2014.2312378.

[7] Y. Jwa and G. Sonh. "KALMAN FILTER BASED RAILWAY TRACKING FROM MOBILE LIDAR DATA". In: *ISPRS Annals of Photogrammetry,*

Remote Sensing and Spatial Information Sciences II-3/W5 (Aug. 2015), pp. 159–164. DOI: 10.5194/isprsannals-II-3-W5-159-2015.

[8] Mostafa Arastounia. "Automated Recognition of Railroad Infrastructure in Rural Areas from LIDAR Data". In: *Remote Sensing* 7 (Nov. 2015), pp. 14916–14938. DOI: 10.3390/rs71114916.

[9] Mostafa Arastounia and Sander Oude Elberink. "Application of Template Matching for Improving Classification of Urban Railroad Point Clouds". In: *Sensors* 16 (Dec. 2016), p. 2112. DOI: 10.3390/s16122112.

[10] Mostafa Arastounia. "An Enhanced Algorithm for Concurrent Recognition of Rail Tracks and Power Cables from Terrestrial and Airborne LiDAR Point Clouds". In: *Infrastructures* 2 (June 2017), p. 8. DOI: 10.3390/infrastructures2020008.

[11] Máté Cserép, Péter Hudoba, and Zoltán Vincellér. "Robust Railroad Cable Detection in Rural Areas from MLS Point Clouds". In: July 2018. DOI: 10.7275/z46z-xh51.

[12] Donald Meagher. *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Oct. 1980.

[13] P.C.V Hough. "Method and means for recognizing complex patterns". Pat. 3 069 654. 1962.

[14] OpenCV. *Open Source Computer Vision Library*. 2015.

[15] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

[16] Martin Isenburg. *LAStools - efficient tools for LiDAR processing*. URL: http://lastools.org.

List of Figures

List of Tables