UPPSALA
UNIVERSITET

Serienummer

Examensarbete 30 hp

Augusti 2022

Guardrail detection for landmark based localization

Nils Gumaelius

Civilingenjörsprogrammet i teknisk fysik

UPPSALA UNIVERSITET

Guardrail detection for landmark based localization

Nils Gumaelius

Abstract

A requirement for safe autonomous driving is to have an accurate global localization of the ego vehicle. Methods based on Global navigation satellite system (GNSS) are the most common, but are not precise enough in areas without good satellite signals. Instead methods like landmark based localization (LBL) can be used. In LBL, sensors onboard the vehicle detect landmarks near the vehicle. With these detections, the position of the vehicle is deduced by looking up matching landmarks in a high-definition map. Commonly found along roads and stretching for long distances guardrails are a great landmark that can be used for LBL. In this thesis two different methods are proposed to detect and vectorize guardrails from vehicle sensor data, to enable future map matching for LBL.

The first method uses semantically labled lidar data with preclassified guardrail LiDAR points as input data. The method is based on the DBSCAN clustering algorithm to cluster and filter out false postives from the preclassified LiDAR points. The second algorithm uses raw LiDAR data as input. The algorithm finds guardrail candidate points by segmenting high density areas and matching these with thresholds taken from the geometry of guardrails. Similar to the first method these are then clustered into guardrail clusters. Respective methods clusters are then vectorized into the wanted output of a 2D vector, corresponding to points inside the guardrail with a specific interval.

To evaluate the performance of the proposed algorithms, simulations from real life data are analyzed in both a quantitative and qualitative way. The qualitative experiments showcases that both methods performs well even in difficult scenarios. Timings of the simulations shows that both methods are fast enough to be applicable in real-time use cases. The defined performance measures shows that the method using raw LiDAR data is more robust and manages to detect more and longer parts of the guardrails.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala/Visby

Handledare: Förnamn Efternamn Ämnesgranskare: Förnamn Efternamn

Examinator: Förnamn Efternamn

Detta examensarbete är sekretessbelagt t o m Klicka och välj datum eller skriv i fältet. (Tas bort om ej sekretessbelagt)

1 Populärvetenskaplig sammanfattning

En utav förutsättningarna för att ett fordon autonomt ska kunna styra sig själv är att själva fordonet vet var den befinner sig både globalt och mer precist, vart exakt på vägen det är. En vanlig lösning till detta är att använda satellitbaserade system såsom GPS. Dessa system kan vara exakta, men i områden där satellit signalen är dålig duger inte noggrannheten. Ett annat sätt att lösa detta är att använda sig av LBL (Landmark based localization). I LBL används data från sensorer som sitter på fordonet såsom kamera eller LiDAR(Light Detection and Ranging sensor) för att detektera ”landmärken”. Dessa landmärken är inte vad man typiskt tänker på som landmärken utan kan vara vilka objekt som helst runt kring vägen, såsom trottoarer, stolpar och vägmarkeringar. Dessa detekterade objekt kan sen jämföras mot en tidigare genererad karta över objekt kring vägen, från vilket positionen av fordonet slutligen kan beräknas.

Vägräcken är ett ganska outforksat landmärke att användas för lokalisering. På grund av hur vanliga vägräcken är på motorvägar samt avsaknaden av andra tydliga lanmärken på dessa så skulle vägräcken vara intressanta att utforska som ett landmärke för lokalisering. I detta projekt så tas två metoder fram för att detektera vägräcken och sedan skapa ett objekt av dessa som sedan kan användas för lokalisering.

Den första presenterade metoden baseras på LiDAR data. LiDAR sensorn fungerar genom att den skickar ut ljusstrålar som den sedan detekterar intensiteten av när ljuset reflekteras. Från detta kan datapunkter där ljuset studsat fås, innehållande bland annat koordinater på punkten. Metoden går ut på att först detektera vertikala objekt genom att hitta linjer av datapunkter där punkterna ligger väldigt nära varandra. Intressanta linjer kan sedan filtreras ut genom att jämföra linjerna mot geometrin hos vägräcken. Vägräcken kan sedan fås fram genom en klustrings algoritm som grupperar ihop linjer som ligger nära varandra till olika räcken.

Den andra metoden använder SLLD (Semantically labled LiDAR) data. SLLD bygger på LiDAR datan men där vägräckes punkter redan valts ut med hjälp av kamera data och en deep learning algoritm. Metoden bygger på att gruppera områden med tätt placerade punkter istället för tätt placerade linjer som i tidigare algoritm. Genom filtrering baserat på fördelningen av klustren kan områden matchande vägräcken väljas ut.

För att utvärdera de två metoderna gjordes simuleringar på verklig data som analyserades både kvalitativt samt kvantitativt. Reulstaten visade att båda metoderna var bra på att detektera vägräcken på både motorvägar samt mer komplicerade scenarion. Tydligt från både resultatdelarna var hur algoritmen som endast använde LiDAR data både detekterade fler samt längre delar av vägräckena. Det finns många intressanta utvecklingar på algoritmerna. Bland annat vore det intressant att testa metoden som nu användes på ren LiDAR data men med SLLD data istället. Detta skulle resultera i en snabbare algoritm och potentiellt bättre resultat.

Contents

2 Introduction

2.1 Background

2.1.1 Autonomous Driving and Localization

Self-driving vehicles, which a decade ago was no more than a sci-fi fantasy, has in the last couple of years taken huge leaps and have gained huge interest with the public. Even though a lot of progress has been made there are still problems needed to be solved, as there is still no fully autonomous vehicle availabe for the public. A fully working autonomous vehicle would change the way transportation works and revolutionize the transportation industry.

Given the different objectives and methods different solutions use, it is hard to say how far we have come. The most common measure of how autonomous a self-driving vehicle is are the SAE levels [1]. It ranks the autonomy on a scale from 0-5 where 0 is no automation and 5 is fully autonomous, so much that a steering wheel is not needed. Although no one has reached the level 5 autonomy yet, there are self-driving vehicles out there. The examples are endless but maybe the most widely known example are the Tesla autopilot. Currently being in level 2 on the SAE scale with no hands needed on the steering wheel, Tesla have a self-driving system available for roughly 160000 beta testers in their cars[2]. Another famous example are from Waymo which has a driverless taxi service available in Phoenix, Arizona. The taxis have level 4 autonomy meaning they have no human driver at all, but are constrained under very limited geographic conditions. They are also monitored with the possibility to be intervened.

The idea behind autonomous driving is fairly straight forward. Have sensors onboard the car that can track and detect objects around the car. Then make the car be able to steer around those objects. Although the objective might sound simple, the sheer amount of data processed, the uncertainty of these objects and the immense safety needs makes the autonomy task very complex. An important part in this complex system is the localization of the ego vehicle. The car needs to know exactly where it is in a global reference system. The straight forward solution to this is to use methods based on Global navigation satellite system (GNSS). The use of GNSS have some problems to it though. In areas where the GNSS signal is bad in for example urban areas where high buildings both makes it hard to find and reflects signals, leading to inaccurate localization. To get robust localization even in GNSS denied areas, landmark based localization (LBL) can be used instead.

In LBL data from sensors such as LiDAR and cameras are used to detect landmarks near the vehicle. With these detections, the position of the vehicle is deduced by looking up matching landmarks in a high-definition map (HD map). Landmarks suitable for LBL are objects easy to detect that is common along the roads. LBL can be done with either a few landmarks such as poles (trafficsigns, streetlights and trees and so on) or trying to distinguish as many objects around the car as possible. Guardrails are an example of a landmark that can be used for LBL. They can stretch for long distances, making it

impossible to detect both the start and end points. They can still be useful for estimating the lateral position and orientation of the vehicle.

2.1.2 LiDAR and Semantically labled LiDAR data

There are many different sensors available that can be used for detecting objects from vehicles. While the sensors have many use cases the most commonly used sensors for object detection are camera, LiDAR and RADAR sensors. In this paper the algorithms are based on LiDAR data.

2.1.3 LiDAR

The "perception" process an autonomous driving vehicle performs is done by a combination of sensors and cameras. In contest with cameras, the LiDAR sensor is the most used sensor out there. LiDAR is a sensor that can determine distances by "sending" out pulses of lasers and measure the time it takes for the reflected light to be detected by the receiver. The LiDAR data are very accurate and because it contains depth information that cameras do not, it is very useful both as a stand alone data set or fused together with camera data.

The LiDAR sensor have a certain amount of lasers it rotates every scan. The amount of lasers decides the vertical resolution of the LiDAR data called pointcloud. Different LiDARs can aswell have different angular resolution deciding the horizontal resolution of the data. The LiDAR pointcloud used as input in the algorithms thus gets the size of the *channels*angular resolution* where each point contains coordinates as well as the intensity of the reflected laser. An example of a LiDAR pointcloud can be seen in figure 1

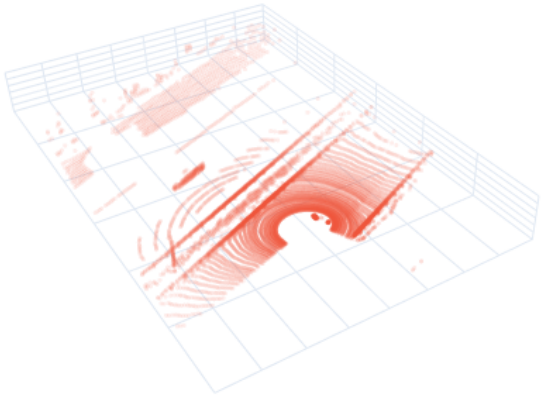

Figure 1: *Showcase of a LiDAR pointcloud. The origin of the circular scans is where the LiDAR sensor are placed.*

2.1.4 Semantically labled LiDAR data

LiDAR data are great for getting depth information about the environment. On the other hand it lacks in distinguishing objects from its depthless information. This is a task cameras excels at. Just like the human eye it is great at detecting information from different pixel values instead of depth. With the advances in machine learning and deep learning modern models have become really great at classifying objects and segmenting the landscape from the camera images.

Paring the data of these sensors together, mapping the segmented image from the cameras onto the LiDAR data, a point cloud with labels can be created. This Semantically labeled LiDAR data (SLLD) can be used for a lot. In the scope of this paper using just guardrail classified points, we get a pointcloud of classified guardrail points. An example of this can be seen in figure 2. Here the LiDAR data is shown in red in the background while the SLLD is shown in blue.

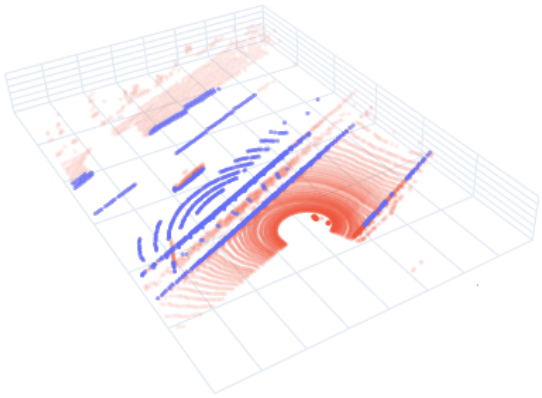

Figure 2: *Showcase of SLLD data. Here raw LiDAR data can be seen in the background in red while SLLD data can be seen in blue.*

2.2 Objectives

In this thesis, detection algorithms for detecting guardrails with pointcloud data will be explored, implemented and tested. The detected guardrails will then be vectorized, to enable future map matching. The results will first be presented in a qualitative part where data from simulations in various driving conditions will display the algorithms performances and compare the methods. Secondly a quantitative part will compare and display the models performances on highway driving with created metrics. To work in a LBL system the algorithm needs to run fast to be able to run in real time. Thus a time analysis will be done to see the efficiency of the algorithms.

3 Related works

This section presents a literature review of various detection algorithms used for detecting landmarks. While there exist some previous methods on real time guardrail detection, the topic is not that well studied. A common use case similar to real time detection of guardrails is to detect guardrails for HD-map making. The algorithms are similar, though one has to have in mind how the different time constraints as well as the fact that when mapping, future data are available, affects the implementation. There are also many choices for what type of landmark to use for LBL. One commonly used landmark are pole-like objects, which are cylindrical objects such as traffic signs, street lights and trees. The similarities in use case and time constraints, makes the algorithms interesting for detecting guardrails as well.

The various data types available from the vehicle sensors makes it possible for a wide range of solutions. The existing methods of guardrails detection includes camera based methods [3, 4], LiDAR based methods [5, 6, 7] and hybrid methods [8, 9]. The lack of a universal labeled test data set, makes comparing the methods very hard. Although from the papers one can find a general idea of it's performance and its pros and cons.

3.1 Camera based methods

Camera based methods are well studied for detecting guardrails, and are especially used in older methods. In [3] back in 2006, a method based on stereo vision, which is a method to extract 3D-information from images, was proposed. The method works well but are based on the hypothesis that the guardrail always moves smoothly along the road. This creates problems when the roads and guardrails have weird shapes or when parts of the guardrails are not visible, due to example vehicles. A more modern approach were introduced in [4]. The method uses inverse perspective mapping to map the image into 2D top-down view. A simple parabolic guardrail model are presented and steerable filters are used to help match the parabolic model to the guardrail. Finally an algorithm using the matching rule are used to filter out false positives. The method is fast and flexible, and performs well in a wide range of scenarios.

3.2 Hybrid methods

One drawback with these camera based methods are that the pixel in the images don't have coordinates. Thus one have to rely on different techniques such as a 2D projection to get an estimate of the detected guardrail. Compared to point cloud based methods this makes it harder to get a reliable distance to the object and also more difficult to calculate the uncertainty of the estimation. To solve this hybrid methods, which uses more than one kinds of input data, are used. They generally use computer vision to detect the guardrail, which can then be projected onto some kind of point cloud, for example LiDAR or RADAR.

As early as 2007 a hybrid method using both RADAR and camera data are proposed in [8]. The RADAR data are used to find search areas of interest. From the images of these search areas different filters are applied to detect guardrails. The method are not on par with today's standard in accuracy and speed but were an inspiration for many future methods.

In the recent years some deep learning methods have been explored for guardrail detection. In [9] a hybrid method based on using CNNs (Convolutional neural networks) on both camera data as well as LiDAR data, are introduced. They used preexisting models (VGG16 for images and PointNet for LiDAR data) and fuse the score of the both models togheter. The results shows promising accuracy. The use of deep learning methods comes with some drawbacks. To get good reliable results lots of data from relevant scenes are needed, where the ground truth had to be manually labeled. To segment the images and LiDAR point cloud the method took use from future data. Thus parts of the method are not applicable for an application running in real time.

3.3 LiDAR based methods

LiDAR data are commonly used for object detection and localization. It comes with the advantage of getting a distance to each point, including a measurement uncertainty. Thus it is advantageous in LBL where uncertainties of each detection are needed.

On of the most studied landmark for LBL are pole-like objects. In [10], the authors introduced a pole detection method based on segmenting individual scan lines. The lines were segmented from the LiDAR data by the distance of adjacent points, resulting in candidate pole sweeps. These candidate sweeps could then be filtered and merged together into pole objects.

This workflow of segmenting high density scan lines have inspired guardrail detection methods as well. In [5], guardrail candidate lines are segmented with a similar method as in [10], by finding lines with many close adjacent points. These lines are then filtered with a number of proposed features from which guardrail candidate lines can be decided. To group the candidate lines together, RANSAC algorithm is applied to filter out lines deviating from the plane along the guardrail. To finally group together guardrail lines as objects, an algorithm based on distance between adjacent lines are used. The accuracy of the method are on par with the best, but the method are proposed for HD-mapping and with running time of 58ms it is not fast enough for a real time application.

In [6] a similar method to [5] are proposed, but here the focus is on creating a method applicable for real time usage. The method uses the same idea of segmenting each scan line by density, as well as using features to filter out guardrail candidates, although fewer. Instead of using RANSAC the method uses a geometry based thresholding problem to group together the lines into objects. The change makes the method nearly five times faster than it's predecessor. Both [5] and [6] have limitations in only being able to detect

one kind of guardrail as well as only detecting the closest guardrail on each side of the vehicle.

Another method also using LiDAR data, that differs from the two earlier are introduced in [7]. It is proposed for HD-mapping and thus take use of future data. The method first uses a multi-level filter to filter out objects from the point cloud that are not guardrails, for example roads, buildings and trees. When only interesting features are left DBSCAN clustering algorithm is used to cluster the data. To find the guard rail clusters, the clusters are filtered over certain features such as intensity, line fitting, and number of points. The method can detect all kinds of guardrails with impressive accuracy but with the drawback of using a fully merged point cloud from many frames.

4 Method

In this paper two methods are proposed, one using SLLD data and one based on raw LiDAR data. The methods share the same foundation, using DBSCAN clustering and filtering methods to find clusters of guardrail data points. These guardrail clusters are then vectorized with the same method to get the final output, a 2D vector corresponding to points with specific interval inside the guardrail. Even though the methods uses similar algorithms, the detection and filtering steps differs between the methods. In the SLLD algorithm, it is expected that most of the points in the dataset already are classified guardrail points. This leads to the detection step being more of detecting false positives and filter out outliers. Here the filtering and thresholding are done for the full point cloud. In contrast, in the Geometry algorithm based on raw LiDAR data guardrail point candidates first have to be detected and classified. This method instead tackles the problem by looking at the LiDARs scan lines one on one, and thus filtering the data before clustering instead of after.

4.1 SLLD based algorithm

When using SLLD data we have LiDAR points classified as points corresponding to a guardrail by a deep learning model. However the quality of the classifications of these points varies a lot. This algorithm tries to filter out all the false positives and outliers to then cluster the true positives into separate guardrails. The algorithm can be divided into 4 steps, clustering, filtering clusters, vectorization and detection of inaccurately vectorized points. The workflow and it's expected results can be seen in figure 3.

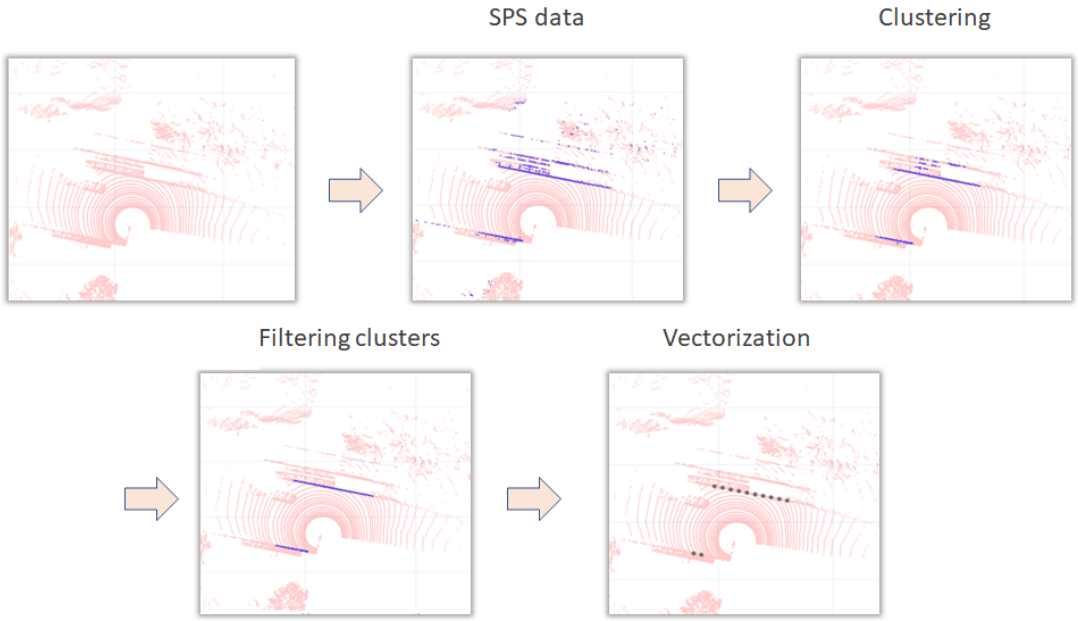

Figure 3: *Workflow for the first detection algorithm using SLLD data. In the frames which represent each stage of the algorithm background LiDAR data is shown in red while the extracted features are shown in blue. The last image shows the final vectorized output as black dots.*

4.1.1 DBSCAN clustering

In the input data the expected guardrails can be seen as dense areas of SLLD data points. To find these dense clusters of data points a DBSCAN (Density-Based Spatial Clustering of Applications with Noise) clustering algorithm is implemented [11]. In Algorithm 1 psuedo code for the implemented simplified version of the algorithm can be seen. The algorithm is great at filtering out sparse noise, can handle all various forms of clusters, and also has no need to specify the number of clusters. The algorithm is also rather fast with an average time complexity of $\mathcal{O}(n \log n)$. All this makes DBSCAN clustering a good fit for this clustering problem.

The idea with the method is to find dense areas of points and group them together into clusters. By looking at the amount of points surrounding each point in a small circular area called neighbourhood, the density can be defined. Then points with high enough neighbours which have connected neighbourhoods can be grouped together into clusters.

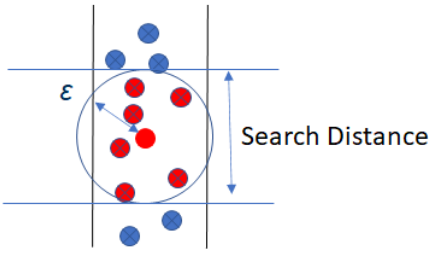

Figure 4: *DBSCAN neighbourhood, showing points inside the search distance radius as red, and points outside as blue.*

More in depth the algorithm works as follows: Let ϵ be a chosen parameter specifying the radius of the points neighbourhood. Starting with a random point in the input data, find the number of neighbouring points inside of the ϵ radius as can be seen in figure 4. There are different techniques to find these neighbourhood points, where the simplest way is checking the euclidean distance to all data points in the dataset. If the point have more neighbours than the chosen parameter $minPts$ in its neighbourhood it creates a cluster. The points in the neighbourhood that have not yet been visited are added to a stack. Then this process continues but instead of picking a random point, iterating over the stack with neighbours.

If the new points from the stack have more neighbours than $minPts$ they are added to the alredy created cluster. When the stack gets empty the cluster is finalized and a new random starting point is chosen. This loop then iterates until all the points have been visited.

Algorithm 1 Simplified DBSCAN

```
for i in length of datapoints do
    if datapoints(i) is visited then
        continue
    end
    Set datapoints(i) to visited
    numNeigh, neigh = FindNeigbours(i, datapoints) ; /* Function seen in Algo 2 */
1   if length of neigh < minPts then
2       Clear tempNeigh, numNeigh
        continue
3   end
4   Append neigh to cluster
    while neigh not empty do
5       index = neigh.back
        Remove neigh.back
        if point is visited then
6           continue
7       end
8       Set point to visited numNeigh, tempNeigh = FindNeigbours(index, datapoints)
9       if length of tempNeigh < minPts then
10          Clear tempNeigh, numNeigh
            continue
11      end
12      Append tempNeigh to cluster
13  end
14  Append cluster to clusters
end
```

The time complexity depends on how the search area for the neighbourhood is chosen. In the case of checking the distance of every point we reach a time complexity of $\mathcal{O}(n^2)$(source). (källor) show some examples of how this can be done with a lower time complexity. In this case, one can make use of the fact that the clusters are guardrails which are straight lines. Thus the data are sorted by its x values and the search interval can easily be narrowed down to $x - e < k < x + e$. The psuedo code for this search algorithm can be seen in algorithm 2.

Algorithm 2 Function for finding neighbours inside searchRadius

```
Function FindNeigbours(i, dataset):
Set j to i - 1
while datapoints(j).x_coord >= (datapoints(i).x_coord − searchRadius) do
    dist = Euclidean distance between datapoints(i) and datapoints(j)
    if dist < searchRadius then
        if datapoints(j) not visited then
            neigh.pushback(j)
        end
        numNeigh++
        j−−
    end
end
Set j to i + 1
while datapoints(j).x_coord <= (datapoints(i).x_coord + searchRadius) do
    dist = Euclidean distance between datapoints(i) and datapoints(j)
    if dist < searchRadius then
        if datapoints(j) not visited then
            neigh.pushback(j)
        end
        numNeigh++
        j++
    end
end
```

4.1.2 Thresholding

The generated clusters are all areas where the LiDAR data is dense. These mainly represent guardrails, but because the SLLD data is not perfect in its classifications some other objects are falsely classified. This can for example be hillsides, fences, buildings and so on.

All guardrails have some features in common. They all extend horizontally in a main direction, the direction of which its variance is highest, usually along the road. They also extends vertically with a height of 20-60cm and perpendicular to the main direction with a width of 5-15cm. For clusters representing guardrails this three properties should correspond to the three vectors which the clusters have highest variance in. For each cluster C_i the covariance matrix is calculated. The value of the eigenvalues for each eigenvector of the matrix corresponds to how large the variance is in that direction. Thus the eigenvector with the largest eigenvalue corresponds to the direction of the largest variance and so on. From this we can set up a thresholding problem for filtering out assumed guardrail

clusters.

Firstly it is checked if the clusters main direction are horizontal, thus it does not have a large component in the z direction, $\mathbf{v}[3] < max\_zdir$. It is also checked if the corresponding eigenvalue is larger than a minimum threshold: $\lambda_{max} > min_{\lambda}$. Thereafter, the cluster are filtered by their height (eigenvalue corresponding to eigenvector with large z component) and width (eigenvalue corresponding to smallest horizontal eigenvector). If they have either larger than or smaller than the specific threshold they are removed. Thus satisfying: $min_{\lambda_z} < \lambda_z < max_{\lambda_z}$ and $min_{\lambda_{min}} < \lambda_{min} < max_{\lambda_{min}}$.

4.2 Geometry based algorithm

When working with the full LiDAR dataset the approach of clustering the input data as a first step becomes impractical. The large size of the input data makes it very time consuming and it would be very hard to distinguish two connected or very closely placed objects with dense LiDAR points. Instead this algorithm first detects guardrail candidate line segments by searching the scan lines from the LiDAR one by one. In the feature extraction step these segments are filtered based on thresholds of guardrails geometry to finally be clustered and vectorized. The workflow can be seen in images in figure 5.

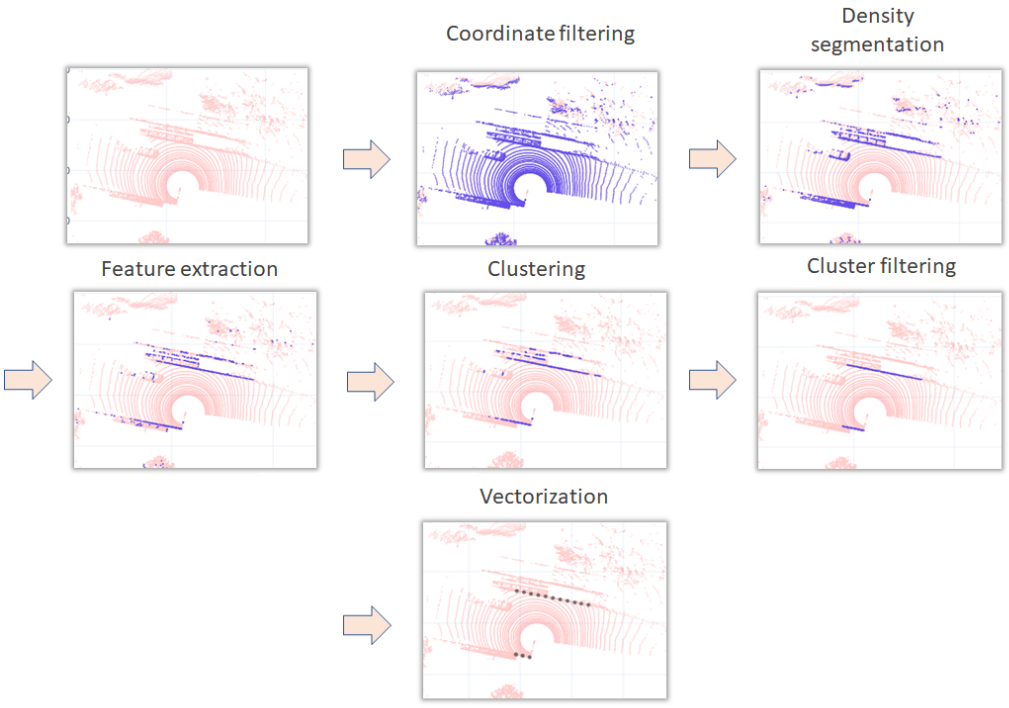

Figure 5: *Workflow for the Geometry based algorithm. In the frames which represent each stage of the algorithm background LiDAR data is shown in red while the extracted features are shown in blue. The last image shows the final vectorized output as black dots.*

4.2.1 Density based angular scan line segmenting

The Input LiDAR data is first filtered based on distance and height to reduce the size of the input data. Points further away than threshold max_dist and closer than min_dist are removed. Points with a height of more than max_height are also removed. This filtering, drastically reduces the time needed for segmenting the lines as many objects far away from the road such as buildings and trees can be dense with LiDAR data even far from the road.

The input data from the LiDAR is represented as a matrix. Here each row represent the number of channels the LiDAR sensor have, thus the vertical resolution. The columns represent the horizontal angular resolution and we call them angular scan lines. Thus the scan lines represents a series of points in a specific angle from the LiDAR. An example of this can be seen in figure 6 where each 32:nd line is shown. Because of the tilt of the LiDAR the scan lines data will be more dense in areas with vertical objects while more sparse in flat horizontal areas. This can be taken advantage of by segmenting the scan lines by density to extract the vertical objects.

An algorithm based on the distance between adjacent points, similar to the discontinuity detection method proposed in [10] is implemented. Let $P_k(p_1, p_2, p_3, \ldots, p_i, \ldots p_n)$ denote each point p_i in every scan line P_k. The euclidean distance between each adjacent point are then calculated, starting with p_2 and p_1 for respectively scan line P_k . If the distance between the points are less than min_dist, p_i are added to the same segment S_j as p_{i-1}. If not, a new segment S_{j+1} are created for p_i. After all scan lines P_1 to P_m are segmented, segments with less points than threshold $minPts$ are removed. The final output from the segmentation can be seen in figure 7.

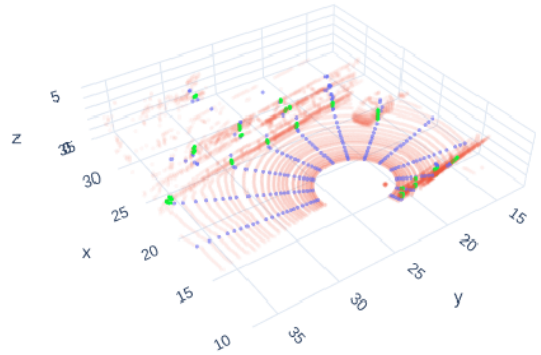

Figure 6: *Density based scan line segmenting. Here every 32:nd scan line is shown. The input data for the scan lines are shown in blue, while the segmented data is shown in green.*

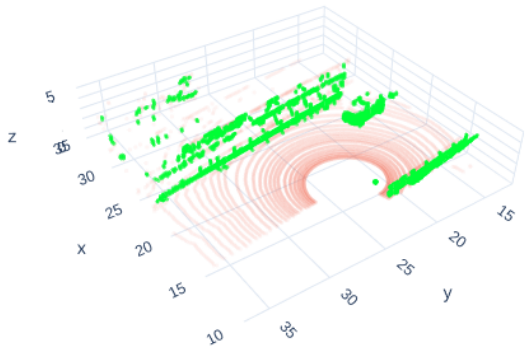

Figure 7: *Output from the Density scan line segmenting from all scan lines.*

4.2.2 Feature extraction

From the line segmentation, vertical objects are extracted from the input data. To filter out the possible guardrail line segments, three features are extracted and analyzed. Firstly two width features of the line segments are extracted. Common unwanted false positives found after the density segmentation are ground points that have been included. Compared to guardrails these segments often extend horizontally and these segments and others can be filtered out with the two proposed width features. The first feature width w_j is defined as the $L2$ distance in the xy-plane between the lowest and highest point in each line segment. The second width feature max_w_j is the maximum $L2$ distance between two points in the xy-plane. From this, points are selected on the criterion: $min\_w < w_i < max\_w$ and $max\_w_j < max_w$.

One other well defined feature of guardrails is the expected height for the line segment. There exists many different kinds of guardrails, but the guardrail of interests in this paper all have a height between $20cm$ and $60cm$. Here the height d_{zj} of the guardrail is found as the z difference between the first and last point in each segment. Thus line segments not satisfying: $min\_height < d_{zj} < max\_height$ are removed.

4.2.3 Clustering

Maybe the most distinct feature of guardrails are how when looking from a bird eye view they extends horizontally in lines. While the line by line feature extraction finds possible guardrail candidate points based on vertical information from each single line it misses the information from the relationship between the points in different scan lines. To find out if the guardrail candidate points from the feature extraction actually extends horizontally as guardrails do, the same density based clustering algorithm DBSCAN as the SLLD algorithm seen in Section 1.1.1, are used. The clustering both checks if the candidate points extends horizontally and groups them together into separate guardrails.

To easier distinguish the different guardrail clusters all the points except the top points are removed. An example of this can be seen in figure 9 and 10. This has two positives, firstly it makes it easier for the clustering algorithm to separate clusters close to each other based on the z-distance. Secondly, it reduces the amount of data, which for a rather timely clustering algorithm speeds up the algorithm.

To further make use of the the z-distance separation the distance function for finding neighbours in the DBSCAN algorithm is modified to further penalize difference in z-distance. This is done by adding a factor of 5 to the z factor in the calculation of the euclidean distance. Especially in cases where guardrails are placed really closed to or upon each other as in example 8 this method were especially effective.

Figure 8: *Image displaying the scene of figure 7 and 8.*

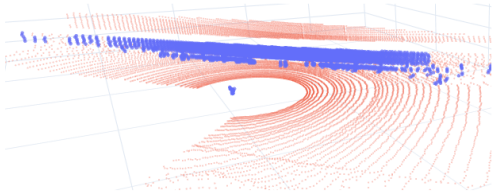

Figure 9:
All the datapoints from the filtered segmented lines.

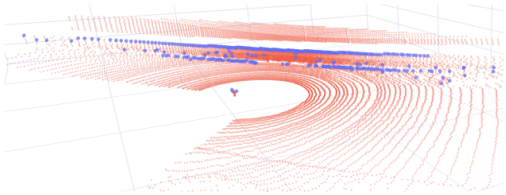

Figure 10: *Only the top point of each line segment from the filtered segmented lines. The reduction in data points both separates clusters better and helps performance.*

4.3 Vectorization

The final wanted output of the algorithm is a vector representing the guardrail as coordinates with a specific interval *seg*. This part of the algorithm takes guardrail clusters as inputs, thus both previously mentioned methods can be used previous to this step.

To simplify the vectorization we want to sort the cluster in the guardrails main direction. For each guardrail cluster the covariance matrix is calculated. From the covariance matrix, the vector corresponding to the direction the cluster have the highest variance in can be found as the eigenvector with the largest eigenvalue. A copy of the original cluster is created and projected onto this vector. With this projection, the original cluster can then be sorted by its matching projected copy in the direction the guardrail extends.

A guardrail is not always straight and in some rare cases such as really sharp turns overlaps may occur when sorting the cluster along it's projection. To fix this, after the sorting the L2 distance is checked between each adjacent point. If the distance is larger than max_dist the cluster is split at the previous point.

The points in the vector output that should represent the clusters needs to be in the middle of the guardrail to give a fair representation. Thus finding a point each seg is not enough as its placement in the width of the guardrail is unknown. Instead an average is taken in a neighborhood containing points closer than $neighDist$ from the point at $lastPoint + seg\_len$. An example of this can be seen in figure 11. With the sorted guardrail cluster this can easily be done with a single loop and checking the $L2$ distance between the previously vectorized point and point p_i. If the distance is larger than $dist > seg\_len - \frac{neighDist}{2}$ a new neighbourhood N_j is created. All points p_i are added untill $dist > seg\_len + \frac{neighDist}{2}$. A new point $pvec_j$ is created at the average of N_i and added to the 2D output vector. This point also becomes the new $lastPoint$. This is done until no more points and clusters are found and all the clusters are vectorized.

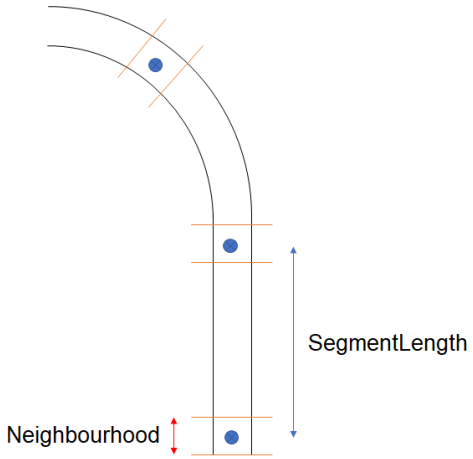

Figure 11: *Figure showing how the vectorization is done. The vectorization points are chosen as the mean of each neighbourhood seen in the image. A vectorizaiton point is chosen for each SegmentLength.*

4.4 Detection of inaccurately vectorized points

When working with the SLLD method, the final output is dependent on the quality of the SLLD data. Although it can handle most cases of false classifications there still exists cases where false positives makes it through the clustering and filtering steps. These are often cases where most of the output guardrail cluster are correct, while only parts of it includes false positives from example the ground under the guardrail or a vertical object connecting to the guardrail. An example of this can be seen in figure 25. Although this may not cause any problems in the vectorization it will if the false negatives are inside the

neighbourhood where the vector point $pvec_j$ is created. To hinder vector points outside guardrails to be included in the output, a function to flag these inaccurately vectorized points is created.

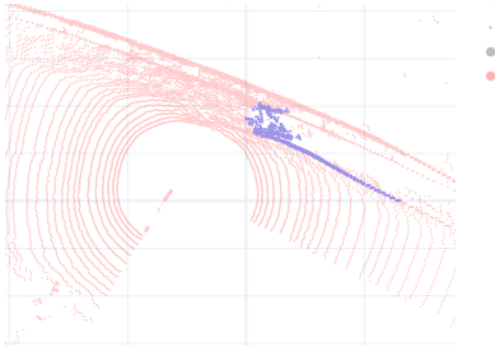

Figure 12: *Image showcasing a scene where the filtered SLLD cluster includes false positive points from the ground. Here red points are LiDAR data and blue points are filtered clusters from the SLLD data.*

To detect if the local neighbourhoods N around the output points *pvec* includes false positives and looks different from the rest of the cluster, the distributions of all neighbourhoods N_0 to N_M are compared. Firstly we want to change the basis of each neighbourhood to always have the basis in each "main direction" and width of the neighbourhood. Thus each neighbourhood N_0 to N_M are mapped onto the two eigenvectors of the the covariance matrix of points in the next segment length $SegLen_{j+1}$, which includes all points between points $pvec_j$ and $pvec_{j+1}$. To find out how the neighbourhoods differs in width the SD (standard deviation) are calculated on the component corresponding to the width(eigenvector corresponding to second largest eigenvalue). Here we can now flag each point $pvec_j$ corresponding to N_j which have larger SD than the threshold $maxSD$ of the median SD of all neighbourhoods: $SD(N_j) > maxSD * Median(SD(\mathbf{N}))$.

5 Experiments

The experiments in this paper are split into two parts. The first part is a qualitative part where the functionality of the algorithms is analyzed by visually observing the output of different scenarios and scenes. Part two of the results contains quantitative results on the performance of the two algorithms. For this three metrics are proposed to compare the two models. L_t the total detected guard rail length, L_a the average length of the longest detected guard rail and T_a, the average amount of guardrails detected.

5.1 Experiments data

The parameters for the algorithms are tuned on data collected by Scania in various surroundings around Södertälje, including both highway driving and low speed urban driving. For all the training and testing a single LiDAR sensor mounted on the top left of the vehicle was used.

The quantitative test data are from a long log taken from the highway between Södertälje and Järna and are completely separate from the train data used. The log contains driving in both directions on a typical highway route. Both the training data and test data are taken in nice driving conditions with a varied amount of clouds.

5.2 Exmperiments part 1

After observing and analyzing large amount of data, a great understanding in how well the algorithms perform can be made. In this section the output of a variety of different scenes are shown. Both generic common environments as well as more troublesome unique environments are shown for the reader to get an understanding of the performance. The displayed results will be further analyzed in the discussion section.

5.2.1 Results

In the first scene two extra reinforced guardrails are present, see figure 15. The guardrails have an extra bar on top of them and fences on the side. As can be seen in figure 13 and 14 both algorithms managed to classify the wanted part of the guardrail as well as getting no false positives. Both methods also manages to vectorize a large portion of the rails, with the Geometry algorithm slightly outperforming the SLLD algorithm.

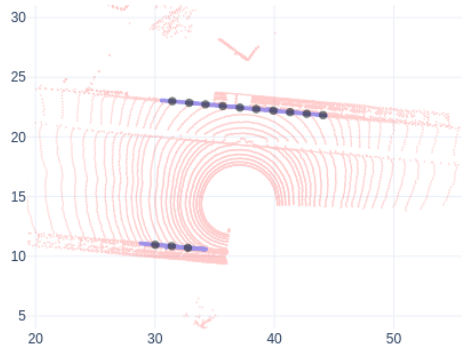

Figure 13: *SLLD algorithm output from the scene seen in figure 18.*

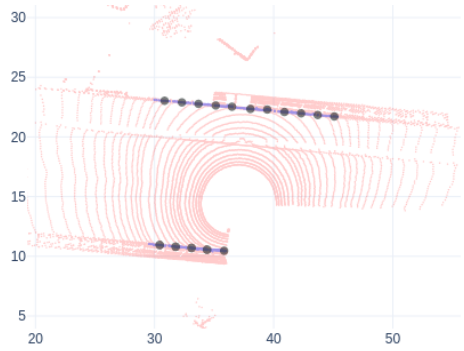

Figure 14: *Geometry algorithm output from the scene seen in figure 18.*

Figure 15: *Scene of the outputs in figure 13 and 14. The scene depicts guardrail with fences on both sides.*

The second scene seen in 18 depicts a highway access. Here a one lane file are merging with a two lane highway. One guardrail exists on the right while two guardrails are about to merge on the left side. From the results seen in figure 16 and 17 the Geometry based algorithm clearly outperforms the SLLD algorithm. The Geometry algorithm both detects all three guradrails while also outputs more vector points for each guardrail.

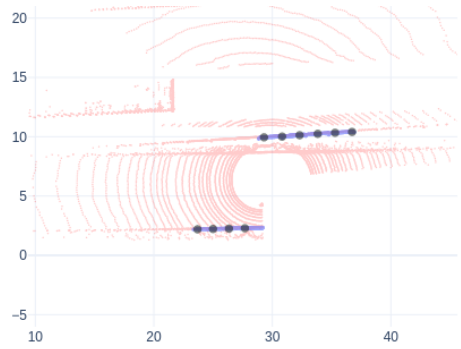

Figure 16: *SLLD algorithm output from the scene seen in figure 15.*

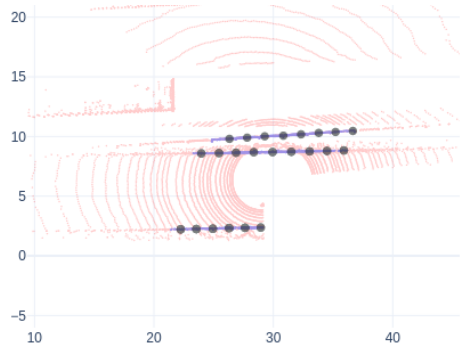

Figure 17: *Geometry algorithm output from the scene seen in figure 15.*

Figure 18: *Scene of the outputs in figure 16 and 17. The scene depicts a highway access.*

The third scene seen, in 21 shows a scene containing one guardrail on each side of the road. Here the right side rail is close to the truck while the left side one is two lanes away. Worth noting in this scene is the slightly higher elevation of the guardrail compared to the road and the sunlight shining right into the camera. The algorithms results can be seen in figure 19 and 20. For this scene none of the algorithm managed to detect the left guard rail and it can be seen that the Geometry based algorithm performs better and outputs a much longer vectorized guardrail for the right one.

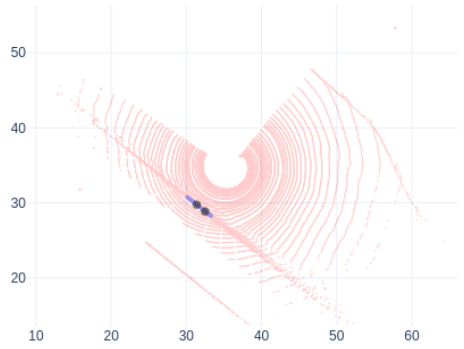

Figure 19: *SLLD algorithm output from the scene seen in figure 12.*

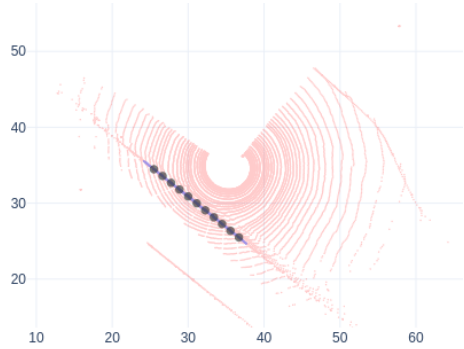

Figure 20: *Geometry based algorithm output from the scene seen in figure 12.*

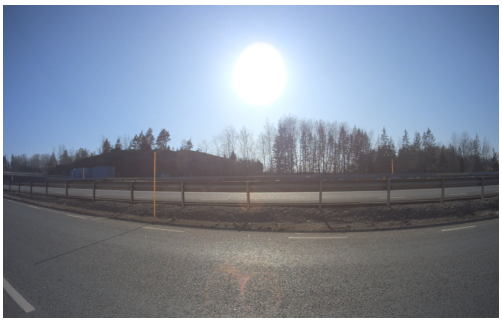

Figure 21: *Scene of the outputs in figure 19 and 20. Special for the scene is the direct sunlight and the slight slight hill on which the guardrail are placed.*

Regular highway driving is an important application for guardrail detection. Scene four see figure 24, depicts an example from a highway with four guardrails. The highway contains three lanes in each direction which makes the distance to the outer guardrail long. The results in 22 and 23 clearly shows how the Geometry algorithm manages to detect and vectorize a much larger part of the guardrails in the scene.

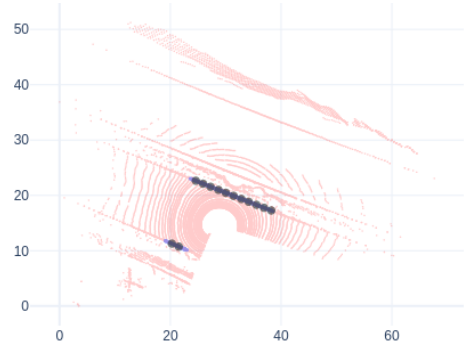

Figure 22: *SLLD algorithm output from the scene seen in figure 21.*

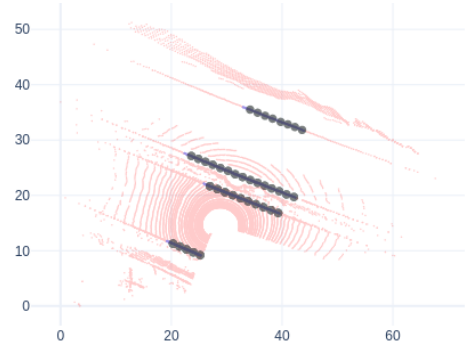

Figure 23: *Geometry algorithm output from the scene seen in figure 21.*

Figure 24: *Scene of the outputs in figure 22 and 23. The scene depicts regular highway driving with four possible rails to detect.*

5.2.2 Detection of inaccurately vectorized points

After experiment 1 the cases were the function detecting inaccurately vectorized points was needed were rare. A couple of cases where the guardrail ended the SLLD data included points from the ground that got included in the guardrail cluster. An example of this can be seen in figure 25. Here the use of the function detecting inaccurately vectorized points can be seen as the vectorized point that are not inside the guardrail is flagged and marked in red.

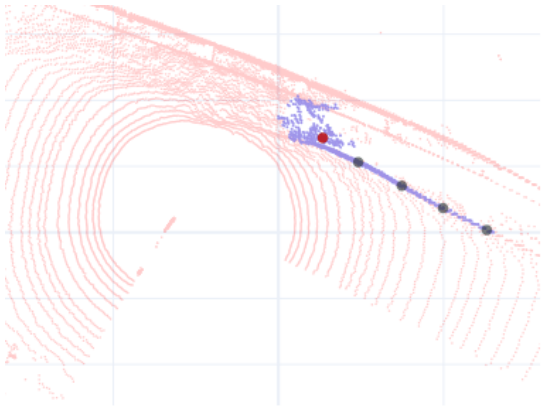

Figure 25: *Image showcasing the function for detecting inaccurately vectorized points. Red background points are LiDAR data, blue points are SLLD filtered clusters, black points are the vectorized points and the red point are a flagged vectorized point.*

5.3 Exmperiments part 2

The long log from Södertälje and Järna as is described in 4.1 was analyzed. In this test a more generic use case in highway driving is analyzed. Firstly, worth noting from this test is that after observing the output no false positives were seen for either of the algorithms. Thus metrics analyzing the completeness of the predictions were proposed. Three metrics were used to validate the results. Let $F[f_0, f_1, ..f_i, ..f_n]$ denote frames 0 to n and $G[g_{i0}, g_{i1}, ..g_{ij}, ..g_{im}]$ denote the detected guardrails 0 to m for each frame f_i. The metrics L_t is the sum of the total guardrail length for each frame, L_a is the average length of the longest detected guard rail and T_a is the average amount of guardrails detected. They are defined as

$$\begin{cases} L_t = \sum_{i=0}^{n} \sum_{j=0}^{m} d(g_{ij}) \\ L_a = \frac{\sum_{i=0}^{n} max(g_{ij})}{n} \\ T_a = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m}}{n}, \end{cases} \quad (1)$$

where the function d is the 2D euclidean distance function between the first and last factorized point in guardrail g_{ij}. Furthermore a time-series plot with the data collected from the detection's over all the frames of the logs were generated to visually see the results from the experiment.

5.3.1 Results

In table 1 the results for experiment 2 are shown. The metrics shows how the algorithm based on LiDAR both detects longer parts of guardrails as well as more guardrails. This

aligns with figure 27 and 26 which show time-series plots of the output from both algorithms. Here the truck drives from left to right on the right lane, with the results using the front left LiDAR. The output from the LiDAR algorithm in 27 seems to detect much more of the guardrails not on the same lane as the driving truck. Noticeably is that the algortihm outputs more of the right side rail on the trucks lane.

Table 1: *Results for the full test run described in 4.1.2.*

	Geometric Algo	**SLLD Algo**
Total distance (Lt)	102165.76m	50237.52m
Average Longest rail (La)	16.77m	9.90m
Average detected rails (Ta)	2.62	2.22

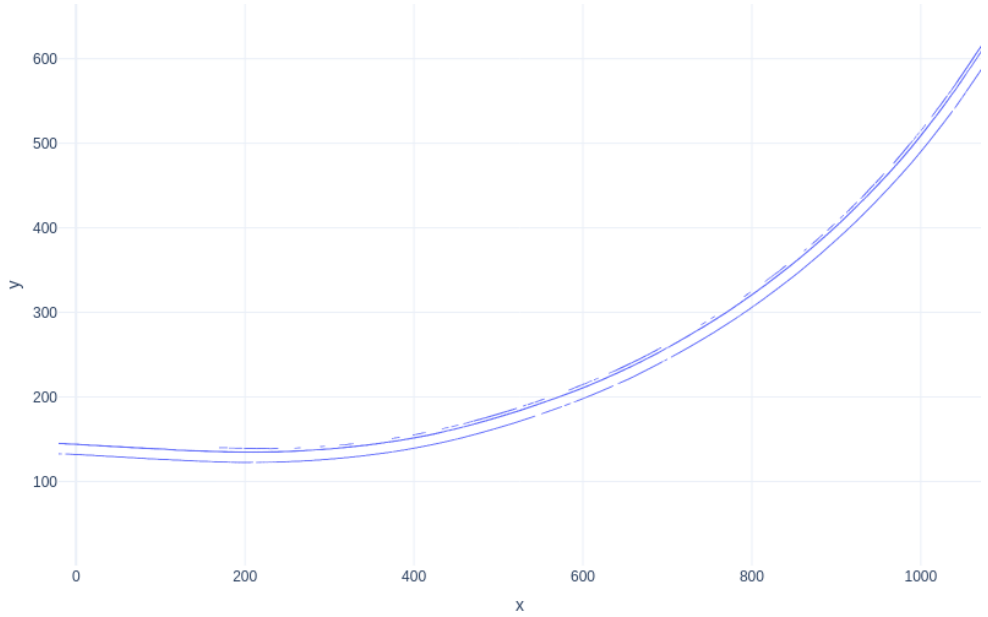

Figure 26: *Timeseries plot of the output from the SLLD algorithm.*

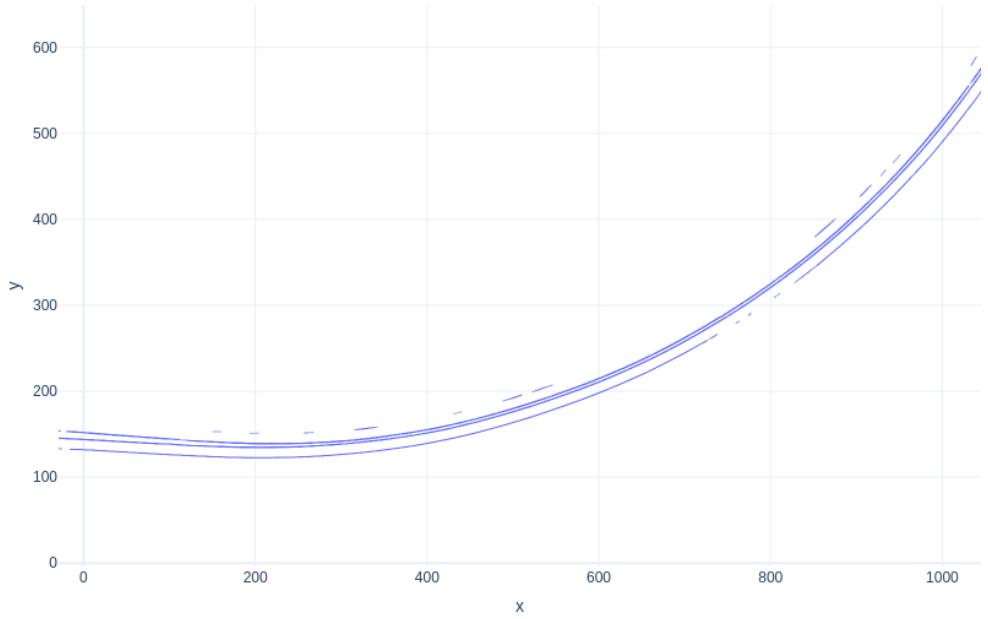

Figure 27: *Timeseries plot of the output from the LiDAR algorithm.*

5.3.2 Time analysis

Landmark based localization are done in real time, thus the goal with the detection algorithms is to with a optimized version be able to be ran real time. This experiment, the whole algorithm has been implemented in C++ and run only on CPU on a Linux laptop with a *Intel® Core™ i7-10850H* CPU running at $2.70GHz \times 12$, and $32GB$ of RAM. Timings taken from the experiments can be seen in table 2 and 3. Here an average case is compared to a worst case scenario. The average case represent a random frame from the log used in experiment 2. The worst case scenario are for both algorithms chosen as the frame where the SLLD algorithm has the highest number of input points. The frame comes from a scene on a highway access containging many false positives in the SLLD data.

The results shows that the SLLD algorithm in general is faster than the LiDAR algorithm. However the clustering algorithm does not scale that well with more input data, thus worst case scenario the SLLD algorithm performs much worse than the LiDAR algorithm.

Table 2: *Time consumption for SLLD algorithm. Here the Average case are an randomly chosen frame from highway driving, while the worst case are the frame where the SLLD input data are the largest.*

SLLD Algorithm	**Average case**	**Worst case**
Input points	2576	4249
Number of guardrails	3	2
Clustering	0.00236s	0.01446s
Filter clusters + Vectorization	0.00124s	0.00401s
Total	**0.00511s**	**0.02074s**

Table 3: *Time consumption for Lidar algorithm. Here the Average case are an randomly chosen frame from highway driving, while the worst case are the frame where the SLLD input data are the largest.*

Geometric Algorithm	**Average case**	**Worst case**
Input points	14446	18204
Number of guardrails	4	2
Distance filtering	0.00435s	0.00423s
Density segmentation	0.00183s	0.00390s
Feature extraction	0.00015s	0.00054s
Clustering + filtration	0.00045s	0.00043s
Vectorization	0.00063s	0.00045s
Total	**0.01107s**	**0.01274s**

6 Discussion

6.1 Experiment 1

Experiment 1 shows qualitatively the performance of both methods. Both methods manages to detect the relevant guardrails, although depending on the environment the range of the vectorized outputs differes between the methods.

The first scene seen in figures 13, 14 and 15, shows a complicated type of guardrail. On both sides of the road they have an extra bar on top of them and fences on the sides. Interesting to notice from this, which are also confirmed by the other scenes, are that both methods performs well at detecting and vectorizing guardrails close to the vehicle in good conditions. Here even though the guardrail are of a special kind both the outputs almost detects the full guardrail on the right side, and a small part of the left side, which is acceptable due to using the right side LiDAR.

In the second scene seen in figures 16, 17 and 18 a highway access are showcased. Here both guardrails manages to vectorize a big chunk of the detected guardrails. However, while the

Geometry based method manages to detect all three guardrails the SLLD method struggles to distinguish the two guardrails close to each other on the right. This is something the Geometry based method excels at. This is mainly because of how it clusters only the top points of the guardrails and extra penalizes the z-distance, which can be read about in chapter 3.2.3.

The third scene, seen in figures 19, 20 and 21 have a regular guardrail one driving lane away from the LiDAR sensor. Not surprisingly, the Geometry based method has no problems detecting and vectorizing a large part of the guardrail. More surprisingly, the SLLD model only manages to detect a small part of the guardrail. Looking at the SLLD input data used for this scene in figure 28, it's clear that the error doesn't depend on the algirithm but the input data used. Analyzing more parts of the logs the quality of the SLLD data seems to get worse when the sun is shining straight into the camera. In this case the guardrail is slightly raised from the driving lane, which could make it even worse. Interesting is that the data points missing were classified as non-drivable ground, thus were mixed up with the background.

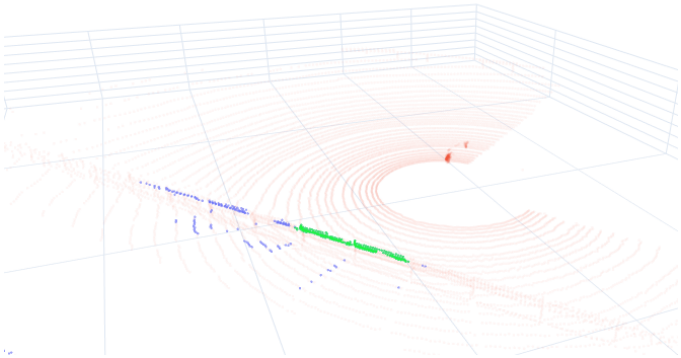

Figure 28: *It can be seen that the SLLD data from scene 3 in experiments 1 is missing a lot of datapoints on the guardrails, making the algorithm just vectorizing parts of it. Here the SLLD data are seen as blue, LiDAR data in red and the filtered clusterd data in green.*

The fourth scene, seen in 22, 23 and 24 showcases how the algorithms performs on regular highway driving. This scene emphasizes what we've seen before, that in good conditions the methods performs similarly well in short range, as both methods outputs similar length on the guardrail in the driving direction. The geomtric models clearly outperforms the SLLD model on the guardrails far away as it manages to detect a good chunk of both guardrails on the road going in the opposite direction.

While running the SLLD algorithm during the tests very few cases where seen where the function for detecting inaccurately points actually were used. Usually the SLLD data either do not have dense areas with false positives close to the guardrails or the filtering step filtererd out the whole cluster including these points. Although not common, it is certainly possible for the clusters to have parts of it including areas of false positives that

messes up the vectorization, as seen in figure 25. The detection function may not be used that much which in itself is positive, but having it as safety in the background to make sure that the vectorized points are valid may be a good safety measure.

6.2 Experiment 2

The results from experiment 2 furthermore concludes what were seen in some parts of experiment 1. The time series plots in figure 27 and 26 clearly shows how the Geometry based method outperforms the SLLD method in detecting guardrails far away from the sensor, in this case the guardrails from the lanes going in the opposite direction. It can also be seen how the two methods performs comparably in the detection of the two guardrails close to the sensor.

Looking at the three performance metrics calculated from the highway test drive, the total distance (L_t) and average number of detected rails (T_a) confirm the previously seen results that the Geometry based method detects more guardrails, which for the highway drive resulted in twice as much vectorized guardrail distance. Interesting what haven't been clear from previous tests are how in the averages longest rail (La) the Geometry based method clearly outperforms the SLLD method even on the close by guardrails, which mostly are the longest vectorized guardrail used in this metric.

Looking at the time consumption for both algorithms they both run sufficiently fast, that a optimized version should be able to run in real time using LBL. Worth noting from the timings in table 2 and 3 are that the code for neither of the methods have been optimized, and the time consumption could be reduced a lot. The timings at least gives a hint on the performance and a comparison between the two methods. Compared to the similar methods proposed in the related works it's comparable to the fastest available method, which clocked in at 15.8ms [6]. However it is hard to do an exact comparison given the use of different datasets.

Noticeably the Geometry based method performs slightly worse in the average case. On the other given almost no time difference in the "worst case" scene with large input data and the average case scene the time consumption is stable and how the scene looks like and how many guardrails needs to be vectorized does not affect the runtime. As the time consumption is almost solely based on the input size which for the LiDAR data is rather consistent, this is expected. Worth noting from the timings are how the distance filtering and distance segmentation are done in two steps. More effectively this could be done in one step and reduce the time needed drastically.

For the SLLD method the time consumption varies more. While the average case takes $0.00511s$ the worst case takes more than four times as long $0.02074s$. This has two reasons. Firstly the input data varies a lot more for the SLLD data than for the raw LiDAR data. As the input data depends on how the scene looks like there is a wide range of the size of the input data. Driving in the country side theres alot of frames having zero input data while the normal case when there is two easily detected guardrails, the input size regularly

becomes roughly 2500 datapoints. In difficult cases where many false positives are included the input data can become much bigger, up to 5000 datapoints. Another big reason for the big time differences is how the clustering is done directly on the input data. Compared to the Geometry based method the most time consuming tasks are the distance filtering and density segmentation which both have time complexity $\mathcal{O}(n)$. Compared the clustering function in the SLLD model have in the worst case scenario $\mathcal{O}(n^2)$. Thus the SLLD model becomes much more sensitive to changes in the input size.

6.3 Proposed methods and future work

In the paper, two proposed methods using two different input data were proposed. Although the methods have some differences and are tuned for the different kind of point cloud data, the methods are very general and each method would with some small changes have no trouble running on each others data sets. That begs the question how much did the different choices in the methods used affect the results, compared to the use of different input data.

Using both datasets the final results are quite similar but the use of data source have some noticeable differences. When using the SLLD data we get a smaller dataset and thus getting a faster algorithm and the possibility for using a more time consuming algorithm. The task of detecting the guardrails becomes easier, but with the drawback on being reliable at how well the SLLD data are generated.

While in theory the SLLD method could run on the full LiDAR dataset, it would have its struggles. Probably most noticeably would be the time consumption. Given the use of clustering early in the algorithm, using input data four times the size it does now, the algorithm would even with massive improvements from optimization have a hard time to be run in real time. The Geometry based method on the other hand would be much easier to use with the SLLD data. A change in input data would result in a much faster runtime and it would be interesting to explore if the method would work better and if so would it be worth a slight performance boost for being reliable on how well the SLLD data are.

The SLLD data not only classifies guardrail points but also other objects such as drivable road, trees, buildings and many more. An interesting possibility would be to use the SLLD data, not as the input data but as a supplement to the raw LiDAR data. It could for example help minimizing the input data as we can remove ground points and buildings. Also interesting would be to see if one could use it as a extra penalty parameter in the clustering. For example it could be used such as when checking if a neighbour is inside the search radius, if it is not classified as a guardrail point by the SLLD data the distance is multiplied with a constant > 1 to penalize it.

Another improvement that could be explored is how one could change the distance function in the clustering method to be able to detect longer chunks of the guardrail. Right now a circular neighbourhood with radius ϵ is used. If one thinks about a guardrail it has a small width, a little larger height, but extends far more horizontally, in it's main direction along

the road. With the goal of detecting as long parts as possible in the main direction, the search area can be changed from a circle to something else. Right now in the Geometry based method a penalty is put on the z-distance modifying this circle into an ellipse. But what if we knew the main direction of the guardrail could we penalize the width part of the distance function which is horizontal orthogonal to the main direction. By doing this we would get a function that makes it easier to detect neighbours in the main direction and thus possibly can detect larger chunks of the guardrail. The problem becomes how to find the main direction before doing the clustering? For the Geometry based method, given that the clustering is done on just a small part of the dataset one could for example do the clustering step twice. First cluster the whole dataset, calculating the main directions of the cluster, then do clustering again starting with points inside the clusters using a modified distance function. Another easier but maybe more naive way would be to assume that we only detect guardrails along the road and use the roads heading as the main direction of the guardrail.

7 Conclusion

While detecting guardrails may seem like an easy task, the diversity in their shape and surroundings makes it hard to do perfectly. The problem often becomes a sacrifice between accuracy and completeness of the detections, and different use cases may require different decisions and tuning. In this papers two methods were proposed that both manages to reliably detect and vectorize guardrails with high performance. One Geometry based based method which uses raw LiDAR data as input and one method based on SLLD data. The results showes that while both methods are reliable on highway driving, the Geometry based based method both manages to output a more complete result in regular driving cases, while also being more reliable in difficult scenarios. Previous work based on raw LiDAR data and density scan line segmentation either were focused on HD-map creation and thus uses slower methods to cluster the guardrails, or choose to only detect a specific kind of guardrail, as well as only the closest one. In this paper new ways to cluster the data using the dbSCAN algorithm and clustering on only the top points in each scan line were proposed and the results shows it to be a fast and reliable way to solve the problem. The SLLD method proposed a novel way to use Sematically labeled LiDAR data for guardrail detection with dbSCAN clustering and filtering based on the variance of the clusters distributions. This paper also extends previous works by proposing a way to vectorize the clusters in real time, to be possible for use in future map matching. The SLLD data although the model using it does not perform as well, may have it's use cases. Interesting future work could include how the SLLD data could be used with the better perfoming Geometry based method. Such as testing the Geometry based method with SLLD data as input or using the SLLD data for filtering and minimize the size of the LiDAR input data.

References

[1] "SAE Levels of Driving Automation™ Refined for Clarity and International Audience."

[2] "Tesla 'Full Self-Driving' Beta opens to another 60,000 testers | CarExpert."

[3] R. Danescu, S. Sobol, S. Nedevschi, and T. Graf, *Stereovision-based side lane and guardrail detection*. Oct. 2006. Pages: 1161.

[4] H. Zhu, X. Wang, and B. Guo, "Camera-Based Guardrail Detection for Intelligent Vehicle," in *2018 Chinese Automation Congress (CAC)*, pp. 777–782, Nov. 2018.

[5] Y. Jiang, B. He, L. Liu, R. Ai, and X. Lang, "Effective and robust corrugated beam guardrail detection based on mobile laser scanning data," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1540–1545, Nov. 2016. ISSN: 2153-0017.

[6] H. Zhu and B. Guo, "A Beam Guardrail Detection Algorithm Using Lidar for Intelligent Vehicle," in *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, pp. 1398–1402, July 2018. ISSN: 2379-7711.

[7] J. Gao, Y. Chen, J. M. Junior, C. Wang, and J. Li, "Rapid Extraction of Urban Road Guardrails From Mobile LiDAR Point Clouds," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 1572–1577, Feb. 2022. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[8] G. Alessandretti, A. Broggi, and P. Cerri, "Vehicle and Guard Rail Detection Using Radar and Vision Data Fusion," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, pp. 95–105, Mar. 2007. Conference Name: IEEE Transactions on Intelligent Transportation Systems.

[9] H. Matsumoto, Y. Mori, and H. Masuda, "Extraction of Guardrails from MMS Data Using Convolutional Neural Network," *International Journal of Automation Technology*, vol. 15, pp. 258–267, May 2021.

[10] M. Lehtomäki, A. Jaakkola, J. Hyyppä, A. Kukko, and H. Kaartinen, "Detection of Vertical Pole-Like Objects in a Road Environment Using Vehicle-Based Laser Scanning Data," *Remote Sensing*, vol. 2, Mar. 2010.

[11] M. Ester, H.-P. Kriegel, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," p. 6.