Fast Segmentation of 3D Point Clouds: A Paradigm on LiDAR Data for Autonomous Vehicle Applications

Dimitris Zermas[1], Izzat Izzat[2] and Nikolaos Papanikolopoulos[1]

[1]Department of Computer Science and Engineering,
University of Minnesota
[2]Advanced Engineering Department,
DELPHI Automotive

***Abstract*— The recent activity in the area of autonomous vehicle navigation has initiated a series of reactions that stirred the automobile industry, pushing for the fast commercialization of this technology which, until recently, seemed futuristic. The LiDAR sensor is able to provide a detailed understanding of the environment surrounding the vehicle making it useful in a plethora of autonomous driving scenarios. Segmenting the 3D point cloud that is provided by modern LiDAR sensors, is the first important step towards the situational assessment pipeline that aims for the safety of the passengers. This step needs to provide accurate segmentation of the ground surface and the obstacles in the vehicle's path, and to process each point cloud in real time. The proposed pipeline aims to solve the problem of 3D point cloud segmentation for data received from a LiDAR in a fast and low complexity manner that targets real world applications. The two-step algorithm first extracts the ground surface in an iterative fashion using deterministically assigned seed points, and then clusters the remaining non-ground points taking advantage of the structure of the LiDAR point cloud. Our proposed algorithms outperform similar approaches in running time, while producing similar results and support the validity of this pipeline as a segmentation tool for real world applications.**

I. Introduction

The recent activity in the area of autonomous vehicle navigation has initiated a series of reactions that stirred the automobile industry, pushing for the fast commercialization of this technology which, until recently, seemed futuristic. Despite the excitement of the general audience, it is imperative for the engineering community to realise the responsibility of bringing such product to a mass production level. This realization has pushed for the fusion of multiple sensors in order to enhance the sensing capabilities of the autonomous vehicles.

One such sensor is the *LiDAR*, which utilizes multiple laser beams to locate obstacles in its surroundings and is known for its capability to depict this information in a dense three dimensional (3D) cloud of points. The LiDAR has been popular amongst academic research teams for its long range and satisfactory accuracy while recent hardware advancements that promise better, lower cost, and smaller scale sensors have appeared to attract the interest of the industry.

Mounted on an autonomous vehicle the sensor by itself provides the means to acquire a 3D representation of the surrounding environment, and the challenge is to analyze it and extract meaningful information such as the number of obstacles, their position and velocity with respect to the vehicle, and their class being a car, a pedestrian, a pole, etc.

Similar to image processing, the first important step for this type of analysis is the fine segmentation of the input data into meaningful clusters. The work in this paper is attacking this exact problem and is presenting a methodology that focuses on computational speed and complexity. A fast and low complexity segmentation process allows to redirect precious hardware resources to more computationally demanding processes in the autonomous driving pipeline.

In the case of LiDAR sensors with the ability to capture 360 degrees of information, the data is represented as a set of 3D points called a *point cloud* which is organized in layers. The points in each layer are also organized in an elliptical fashion and the starting points of all elliptical layers are considered to share the same orientation.The presented methodology relies on this type of organization in the point cloud and takes advantage of smart indexing to perform an efficient segmentation.

Similar to preceding work in the same domain, our approach proposes the segmentation process to conclude in two steps; (i) the extraction of the points belonging to the ground, and (ii) the clustering of the remaining points into meaningful sets. Both steps present original approaches to the problem focused on real world applications and extensively tested on the publicly available KITTI dataset [1].

In the following, the related literature review can be found in section II. The methodology is explicitly described in section III and results can be found in section IV, followed by the last section V that summarises the findings of this work.

II. Literature Review

In this work we are focusing on segmentation methods for data points in three dimensions and emphasize to applications related to autonomous vehicle driving. A common practise for this particular application is to split the segmentation process in two steps; first extracting the ground points and then clustering the remaining points into objects the car needs to be aware of.

For the first step, a grid-based approach was introduced by Thrun et al. [2] which are dividing the grid cells as ground and non-ground based on the maximum absolute difference between the heights of points inside the cell. On the other hand, Himmelsbach et al. [3] are treating the point cloud in cylindrical coordinates and taking advantage of the distribution of the points in order to fit line segments to the point cloud. The segments, based on some threshold of the slope are considered to capture the ground surface. In an attempt to recognize the ground surface, Moosmann et al. [4] are creating an undirected graph and compare local changes of plane normals in order to characterize changes in the slope. Douillard et al. [5] are introducing GP-INSAC, a gaussian-process based iterative scheme that classifies points in ground and non-ground based on the variance of their height from the mean of a gaussian distribution. Similarly, gaussian processes for ground extraction were later used by Chen et al. [6], while probabilistic approaches utilizing Markov Random Fields can be seen in the work of Tse et al. [7] and Byun et al. [8].

Consecutively, the grouping of the remaining well separated non-ground points is usually treated as a clustering problem where appropriate well known clustering algorithms are employed. Such are the cases for clustering algorithms that are popular for their simplicity, easy deployment, and high execution speed. Examples include the euclidean cluster extraction [9] whose implementation can be found in the point cloud library (PCL) [10], DBSCAN [11], and Mean-Shift [12]. The use of voxelization techniques to compress and then cluster the remaining non-ground points is also considered popular ([3], [5]). These algorithms traverse the point cloud in an irregular way and upon finding an unlabeled point, they assign a new label which is then propagated to neighboring unlabeled points based on some rules. Inside a three dimensional space, such irregular accessing of points can lead to exhaustive search for neighbors that slow down the whole process. Although this is necessary for unorganized point clouds, in the targeted application the layer-based organization of the point cloud is not exploited.

III. Methodology

The following paragraphs describe in detail the complete methodology for the segmentation of a point cloud received by a 360^o coverage LiDAR sensor. First, we present a deterministic iterative multiple plane fitting technique we call *Ground Plane Fitting* (GPF) for the fast extraction of the ground points, followed by a point cloud clustering methodology named *Scan Line Run* (SLR) which is inspired by algorithms for connected components labeling in binary images.

Each paragraph is conceptually divided in three sections including a brief reasoning behind the algorithm selection along with the definition of new terms, the overview of the algorithm according to the pseudocode diagrams, and discussion of algorithm implementation details.

Algorithm 1: Pseudocode of the ground plane fitting methodology for one segment of the point cloud.

Result: $\mathbf{P}_g$: points belonging to ground surface
$\mathbf{P}_{ng}$: points not belonging to ground surface

Initialization:
$\mathbf{P}$: input point cloud
N_{iter} : number of iterations
N_{LPR} : number of points used to estimate the *LPR*
Th_{seeds} : threshold for points to be considered initial seeds
Th_{dist} : threshold distance from the plane

Main Loop:
$\mathbf{P}_g =$ **ExtractInitialSeeds**($\mathbf{P}$, N_{LPR}, Th_{seeds});
for $i = 1 : N_{iter}$ **do**
  $model =$ **EstimatePlane**($\mathbf{P}_g$);
  clear($\mathbf{P}_g$, $\mathbf{P}_{ng}$);
  for $k = 1 : |\mathbf{P}|$ **do**
    if $model($ p_k $) < Th_{dist}$ **then**
      $\mathbf{P}_g \leftarrow$ p_k;
    else
      $\mathbf{P}_{ng} \leftarrow$ p_k;
    end
  end
end

ExtractInitialSeeds:
$\mathbf{P}_{sorted} =$ **SortOnHeight**($\mathbf{P}$);
$LPR =$ **Average**($\mathbf{P}_{sorted}($ $1 : N_{LPR}$ $)$);
for $k = 1 : |\mathbf{P}|$ **do**
  if $p_k.height < LPR.height + Th_{seeds}$ **then**
    $seeds \leftarrow p_k$
  end
end
return($seeds$);

A. Ground Plane Fitting

Cloud points that belong to the ground surface constitute the majority of the point cloud and their removal significantly reduces the number of points involved in the proceeding computations. The identification and extraction of ground points is rather suitable for this application for two main reasons; (i) they are easily identifiable since they belong to planes, which are primitive geometrical objects with a simple mathematical model, and (ii) it is acceptable to assume that points of the point cloud with the lowest height values are most likely to belong to the ground surface. This prior knowledge is used to dictate a set of points for the initiation of the algorithm and is eliminating the random selection seen in typical plane-fit techniques such as the RANdom SAmple Consensus (RANSAC), resulting in much faster convergence.

Generally, a single plane model is insufficient for the representation of the real ground surface as the ground points do not form a perfect plane and the LiDAR measurements introduce significant noise for long distance measurements. We have observed that in most instances the ground surface

exhibits changes in slope which need to be detected. The proposed ground plane fitting technique extends its applicability to such instances of the ground surface by dividing evenly the point cloud into a number of segments N_{segs} along the x-axis (direction of travel of the vehicle), and applying the ground plane fitting algorithm in each one of those segments.

As depicted in the main loop of Alg. 1, for each of the point cloud segments the ground plane fitting starts by deterministically extracting a set of seed points with low height values which are then used to estimate the initial plane model of the ground surface. Each point in the point cloud segment $\mathbf{P}$ is evaluated against the estimated plane model and produces the distance from the point to its orthogonal projection on the candidate plane. This distance is compared to a user defined threshold Th_{dist}, which decides whether the point belongs to the ground surface or not. The points belonging to the ground surface are used as seeds for the refined estimation of a new plane model and the process repeats for N_{iter} number of times. Finally, the ground points resulting from this algorithm for each of the point cloud segments can be concatenated and provide the entire ground plane.

Our approach for the selection of initial seed points introduces the lowest point representative (LPR), a point defined as the average of the N_{LPR} lowest height value points of the point cloud. The LPR guarantees that noisy measurements will not affect the plane estimation step. Once the LPR has been computed, it is treated as the lowest height value point of the point cloud $\mathbf{P}$ and the points inside the height threshold Th_{seeds} are used as the initial seeds for the plane model estimation.

For the estimation of the plane, we utilize the simple linear model:

$$\begin{aligned} ax + by + cz + d &= 0 \\ \mathbf{n}^T\mathbf{x} &= -d, \end{aligned} \tag{1}$$

with $\mathbf{n} = [a\ b\ c]^T$ and $\mathbf{x} = [x\ y\ z]^T$, and solve for the normal $\mathbf{n}$ through the covariance matrix $C \in R^{3x3}$ as computed by the set of seed points $S \in R^3$:

$$C = \sum_{i=1:|S|} (s_i - \hat{s})(s_i - \hat{s})^T, \tag{2}$$

where $\hat{s} \in R^3$ is the mean of all $s_i \in S$.

The covariance matrix C captures the dispersion of the seed points and its three singular vectors that can be computed by its singular value decomposition (SVD), describe the three main directions of this dispersion. Since the plane is a flat surface, the normal $\mathbf{n}$, which is perpendicular to the plane, indicates the direction with the least variance and is captured by the singular vector corresponding to the smallest singular value.

After the acquisition of $\mathbf{n}$, d is directly computed from Eq. 1 by substituting $\mathbf{x}$ with $\hat{s}$ which is a good representative for the points belonging to the plane.

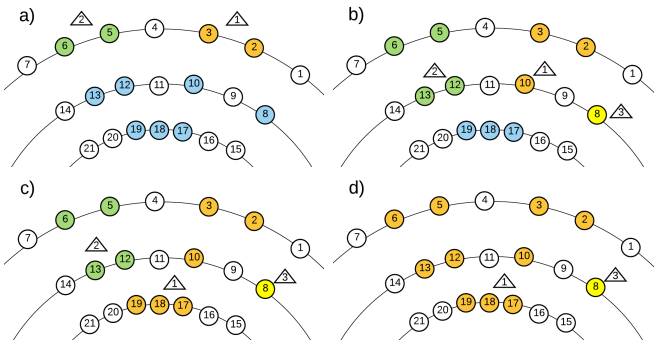


Fig. 1: The four stages exemplify the processes of the SLR clustering algorithm. Circles represent points and triangles report the cluster labels.

B. Scan Line Run

The remaining points P_{ng} that do not belong to the ground surface need to form clusters to be used in higher level post processing schemes. Our goal is for each point $p_k \in P_{ng}$ to acquire a label l that represents its cluster identity while using simple mechanisms that will ensure the fast running time and low complexity of the process.

In the case of 360^o LiDAR sensor data, the multi-layer structure of the 3D point cloud resembles strongly the row-wise structure of 2D images with the main differences being the uneven number of elements in each layer and its circular form. The proposed solution treats the 3D points as pixels of an image and adapts a two-run connected component labeling technique from binary images [13] to produce a real time 3D clustering algorithm.

We call a layer of points that are produced from the same LiDAR ring a *scan-line*. Within each scan-line, its elements are organized in vectors of contiguous points called *runs*. The elements within a run share the same label and are the main building blocks of the clusters.

According to Alg. 2 and without loss of generality, we assume the point cloud P_{ng} is traversed in a raster counter-clockwise fashion starting from the top scan-line. The runs of the first scan-line are formed and each receives its own $newLabel$ which is inherited by all of its point-elements. The runs of the first scan-line then become the $runsAbove$ and are used to propagate their labels to the runs in the subsequent scan-line. The label is propagated to a new run, when the distance between a point of the new run and its nearest neighbor in the above scan-line is less than Th_{merge}. When many points in the same run have nearest neighbors with different inheritable labels, the winning label is the smallest one. On the other hand, when no appropriate nearest neighbors can be found for any of the points in the run, it receives a $newLabel$. The above are performed in a single pass through the point cloud and when this is done, a second pass is performed for the final update of the point's labels and the extraction of the clusters.

The following example accompanying Fig. 1 covers the main instances of the proposed algorithm with the white and colored circles representing ground and non-ground points respectively. The blue circles are non-ground points not yet

Algorithm 2: Pseudocode of the scan line run clustering.

Result: labels : labels of the non ground points

Initialization:

$\mathbf{P}$: input point cloud

$N_{scanlines}$: number of scan lines

Th_{run} : threshold for points to belong in the same run

Th_{merge} : threshold to merge neighboring runs

$newLabel = 1$: label identity

Main Loop:

$runsAbove =$ **FindRuns**($scanline_1$);

for $i = 1 : |runsAbove|$ **do**

  $runsAbove_i.label = newLabel$;

  $newLabel++$;

end

for $i = 2 : N_{scanlines}$ **do**

  $runsCurrent =$ **FindRuns**($scanline_i$);

  UpdateLabels($runsCurrent, runsAbove$);

  $runsAbove = runsCurrent$;

end

ExtractClusters();

UpdateLabels:

for $i = 1 : |runsCurrent|$ **do**

  for $j = 1 : |P_{runsCurrent_i}|$ **do**

    $p_{NN} =$ **FindNearestNeighbor**($p_j, runsAbove$);

    $labelsToMerge \leftarrow p_{NN}.label$;

  end

  if ***isEmpty***($labelsToMerge$) **then**

    $runsCurrent_i.label = newLabel$;

    $newLabel++$;

  else

    $l_R =$ **min**($labelsToMerge$);

    $runsCurrent_i.label = l_R$;

    MergeLabels($labelsToMerge$);

  end

end

visited. In step **a)**, the first scan-line is initialized with two runs (*orange and green*) each receiving a $newLabel$ (1 and 2 inside the triangles). Step **b)** demonstrates the assignment of a $newLabel$ and the propagation of two labels. In particular, the nearest non-ground neighbor of 8 is 2 and their distance is greater than Th_{merge}. In this case, $labelsToMerge$ is empty and point 8 represents a new cluster. On the other hand, the nearest non-ground neighbor of 10 is 3 with their distance smaller than Th_{merge}, which makes label 1 to propagate over to point 10. Similarly, points 12 and 13 are both close to their respective neighbors 5 and 6, and based on the non-empty $labelsToMerge$, label 2 is assigning to them. Next, the final scan-line is considered in step **c)** where one run is present. Points 17 and 19 have neighbors 10 and 12 which belong to different clusters and are both appropriate to propagate their label. According to our algorithmic logic the smallest of the two labels (namely label 1) is inherited.

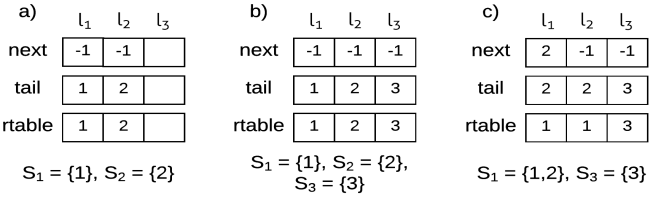


Fig. 2: An example of the label conflict resolving technique based on Fig. 1.

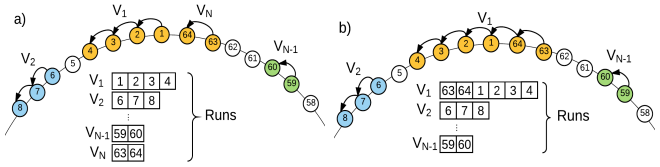


Fig. 3: Example on bridging the two ends of a circular scan-line.

In step **d)** the merging of the two labels 1 and 2 is noted and handled accordingly by the label equivalence resolving technique which is discussed below.

Implementation Details: The outline of the algorithm is straight forward, but for an efficient implementation we propose solutions on (i) how to create runs, (ii) how to look for the nearest neighbor, and (iii) how to resolve label conflicts when merging two or more connected components.

i) A run is created upon the first visit of the scan-line as a vector of indeces and holds information on which consecutive points are close enough to be considered a single block inside a scan-line. Considering the circular form of the scan-lines, a run may bridge over the first and last indeces. When detected, this case is resolved by attaching the indeces of the ending of the scan-line at the beginning of the indeces of the first run as seen in the example of Fig. 3.

ii) When the input point cloud is expressed in cylindrical coordinates with points $\mathbf{x} = [r\ \theta\ z]$, then indexing the nearest neighbor in the scan-line above can be viewed as simply comparing θ values. In autonomous vehicle applications though, clustering is one small component of a much larger system of sensors and algorithms, and the cartesian coordinate system is preferred for compatibility reasons. Implementation-wise, the *naive* solution is to build a *kdtree* structure with all the non-ground points in the scan-line above and use this to find each nearest neighbor, resulting in a suboptimal but viable solution that can be further refined.

Under the assumption that the points in a scan-line are evenly distributed along the whole scan-line, we are utilizing a smart indexing methodology that overcomes the problem of the uneven number of elements in the different scan-lines and significantly reduces the number of queries for the nearest neighbor. Assume that each scan-line has N_i number of points and that each point owns two indeces; one global ind_g which represents its position in the whole point cloud, and one local ind_l that identifies the point inside the scan-line. One can easily alternate between the indeces of the scan-line K by:

$$ind_{l_K} = ind_g - \sum_{i=0}^{K-1} N_i, N_0 = 0. \quad (3)$$

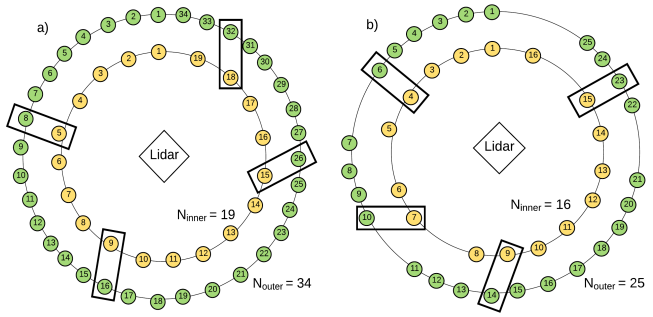


Fig. 4: Examples of smart indexing; (**a**) when the two scan-lines have a significant difference in points (N_{outer} is almost double N_{inner}), and (**b**) when points in both lines are missing because of noise and physical limitations of the sensor.

Given a point index in scan-line i with local index ind_{l_i} it is possible to directly find the local index of a neighbor ind_{l_j} in the close vicinity of the actual nearest neighbor in the above scan-line j by the following equation:

$$ind_{l_j} = floor(\frac{N_j}{N_i} ind_{l_i}), \tag{4}$$

as well as computing its global index from Eq. 3.

Depending on the distribution of the points inside the scan-line, the index might not indicate the nearest neighbor but a close enough point. In this case, it may be necessary to search through a number of its surrounding points for the nearest neighbor, but this number is far smaller than considering the whole scan-line.

In a run, identifying potential neighbors and searching through their surroundings for the best match results in a large overhead that undermines the performance of the algorithm. Bearing this in mind, the proposed solution is to find the nearest neighbors of the first and last points of a run via the smart indexing, form a kdtree structure with all the non-ground points within that range, and use this to search for nearest neighbors.

Two visual examples of the smart indexing can be seen in Fig. 4. In **a)**, although the number of points in the two scan-lines is quite different, the randomly selected points with local indeces 8, 16, 26, and 32 in the outer scan-line are indicated as the nearest neighbors of the points with local indeces 5, 9, 15, and 18 respectively in the inner scan-line. In addition, in **b)** the distribution of points is highly uneven but smart indexing still succeeds to indicate appropriate neighbors. These cases are common to the first few scan-lines when some of their laser beams never return, because of absorption or very high distance. In rare cases where the number of points between consecutive scan-lines is vastly different or a significant portion of the scan-line is missing, the smart indexing will most likely fail. In these cases, the naive solution where the whole scan-line is considered as potential nearest neighbors still produces good results.

iii) The methodology to resolve label merging conflicts is being introduced in [13] where all the details for the implementation and deep understanding are provided. Following, a brief presentation of the essentials along with a simple example is given.

The label merging conflicts arise when two or more different labeled components need to merge. According to He et al. [13], the solution is given by accumulating their labels l in the same set S and utilizing a sophisticated methodology with three 1-dimensional arrays to capture their hierarchies and connections. All three vectors have the size of the number of total labels that have been created during the first pass through the point cloud. Each entry of the first vector *"next"* stores the next l in its S and the entry for the last l in the S is -1. Next, the vector *"tail"* stores the index to the last l of the S. The last vector *"rtable"* has the assistive role of reporting what the final label of each l would be at any given moment. At the end of the first pass, *rtable* is used as the look-up table for the final labelling.

Let us examine the example of Fig. 2 from the point-view of the three vectors. In the first step **a**), two labels are created (1 and 2) and the l_1, l_2 entries are filled. Each of the two sets has only one element thus *next* entries are both -1, *tail* entries show the index of the last element in the S which is 1 and 2 respectively for the two sets S, and *rtable* shows the final representative label. Next, in **b**) the l_3 is created and the vectors are filled the same as before. Finally, the S_1 and S_2 merge which means that the first entry of *next* will point to the index of the next element in S_1, the *tail* for both elements in S_1 is the same and points at the index of the last element of the set, and *rtable* is updated to properly depict the final labels.

IV. Experimental Results

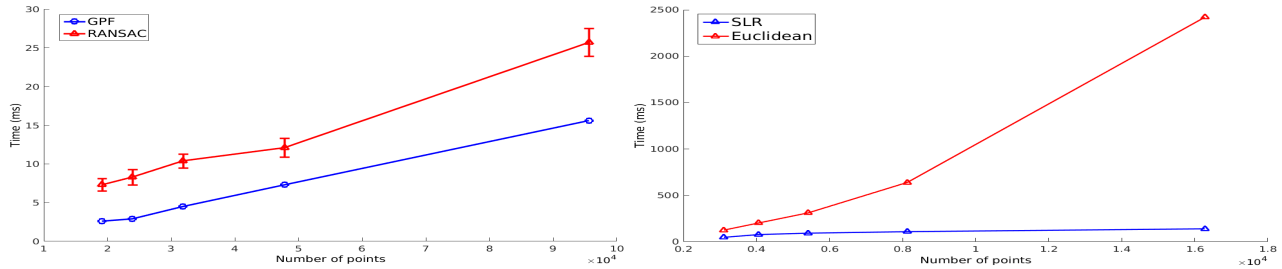
In order to test the proposed pipeline, we experimented on the KITTI dataset for ground-points extraction and clustering. We provide performance comparisons with indicative algorithms that are used in similar scenarios, show the final results and stress the speed differences in our proposed solutions.

The calibration of the parameters for this application is rather intuitive as it mostly reflects distances between 3D points. In our experiments, the parameters for the GPF were set as N_{segs} = 3, N_{iter} = 3, N_{LPR} = 20, Th_{seeds} = 0.4m and Th_{dist} = 0.2m. For SLR the parameters were Th_{run} = 0.5m and Th_{merge} = 1m, while the radius for the euclidean cluster extraction used for comparison was set to 0.5m.

A. Ground Plane Fitting

The GPF is compared to a RANSAC plane fitting implementation which is used in similar scenarios. The 3D point labelling results are similar and our method performs faster (Fig. 5a).

In Fig. 6 the comparison between the results of the GPF for fitting a single plane versus fitting multiple planes are shown. By splitting the initial point cloud into multiple segments, and fitting a plane to each segment, it is possible to correct erroneous labeling for the ground points.



(a) GPF algorithm scalability with respect to the number of points.

(b) SLR algorithm scalability with respect to the number of points.

Fig. 5: Average running time performance of the GPF versus RANSAC (a), and SLR versus the Euclidean Cluster Extraction algorithm (b) with increasing number of points. RANSAC does not have a stable running time because of its random character.

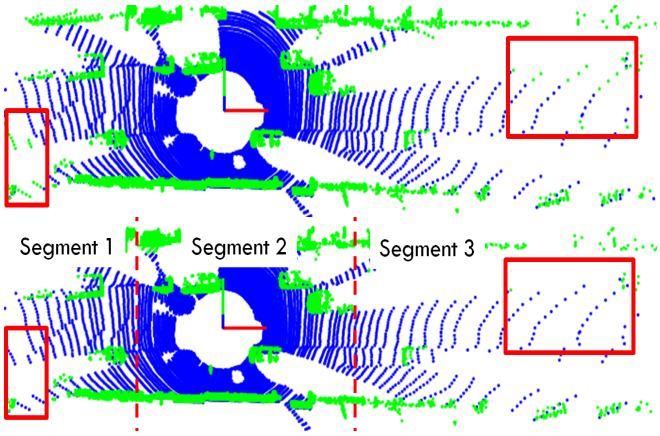


Fig. 6: *Top:* Output of the GPF for fitting a single plane to the entire point cloud. *Bottom:* Output of the GPF for fitting a plane to each one of the segments. The red boxes indicate areas with wrong labeling in the case of a single plane fit and the correct labelling in the multiple planes fit. In both images the blue points belong to the ground and the green to non-ground points.

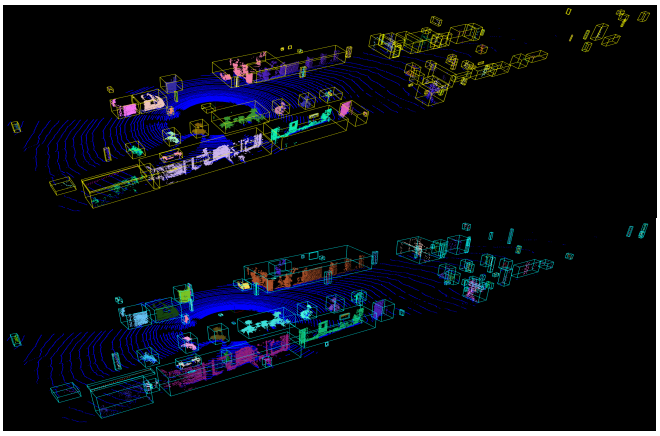


Fig. 7: *Top:* Output of the GPF and SLR pipeline. *Bottom:* Output of the GPF and Euclidean Cluster Extraction pipeline for comparison.The KITTI-dataset point cloud is cropped on the two sides along the y-axis for reduction of points. The blue points belong to the ground and the rest of the colors depict non-ground clusters.

B. Scan-Line Run

The resulting clusters can be seen in Fig. 7 inside yellow bounding boxes. The SLR algorithm takes advantage of the structure of the point cloud and performs much faster than algorithms that traverse the point cloud in an irregular way. The diagram seen in Fig. 5b shows an linear increase in time as the number of points increase. In comparison, the Euclidean Cluster Extraction (ECE) algorithm gets exponentially slower with the increase of the number of points. The clustering algorithms that need to search for neighbors within the whole dataset, base their performance on the total number of points of the cloud. On the other hand, the search for SLR is restricted by the size of its runs and considers neighbors only on one scan-line achieving better performance.

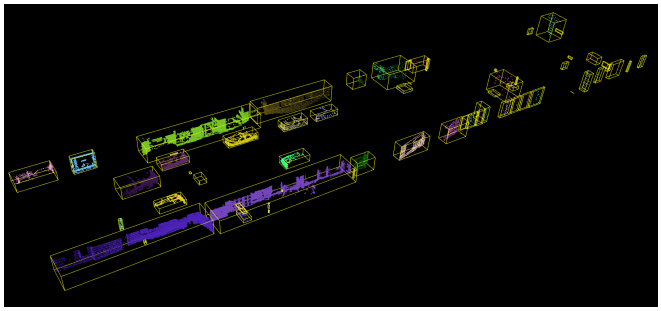
As seen in Fig. 8, reduction of the point density in the point cloud affects the SLR segmentation with a case of over-segmentation being presented in 8b. The algorithm behaves satisfactorily for objects in the proximity of the sensor as the adjacent vehicles have been correctly segmented. More results for the comparison between the SLR and the ECE are provided in Fig. 9, where the similar behavior of the two algorithms is noted. Small variations are present due to differences in the threshold parameters and the 3D point accessing order.

V. Conclusions and Future Work

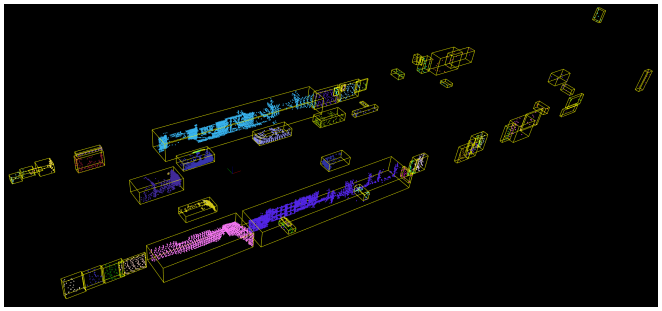
We have examined the problem of segmenting point cloud data for applications in autonomous driving vehicle and have proposed a pipeline that initially extracts the ground points and consequently groups the remaining points into clusters based on their distance. The proposed solution is tailored around the specific application and performs significantly faster than general purpose segmentation pipelines, which makes it ideal for real time operations on data gathered by LiDAR sensors.

The algorithm has been successfully tested in several sequences of the publicly available KITTI dataset and its performance has been verified for a plethora of different scenes and number of points. An additional step is to verify how the pipeline behaves when encountering rough, uneven terrain and rapid slope changes. For this we plan to collect our own data that will capture corner cases and will allow us to verify the robustness of our algorithms.

Segmentation is the first step in the autonomous scene understanding. Once the detection of the obstacles in the environment surrounding the vehicle is captured, the next

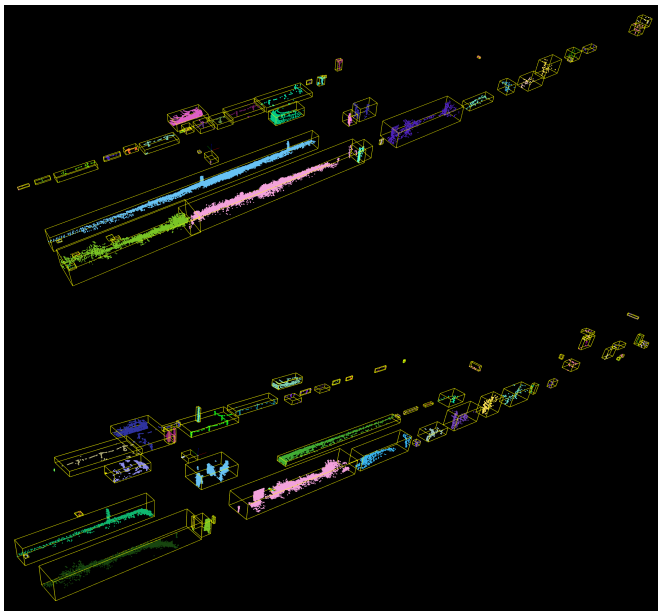


(a) Dense point cloud



(b) Sparse point cloud

Fig. 8: The results of SLR for the same LiDAR scan in the original form (a) and subsampled (b). Out of the 104,863 points, every fifth point is kept to reduce the number down to 20,973 points. The algorithm managed to handle the sparse point case.



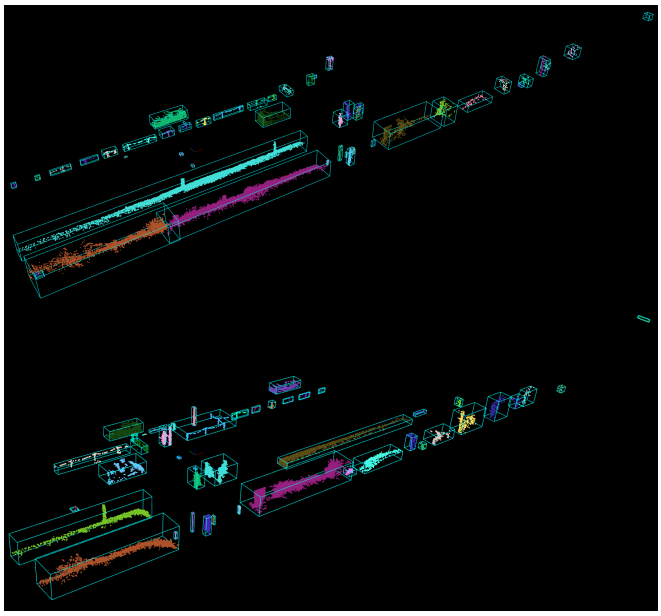


Fig. 9: Results of two LiDAR point clouds from the KITTI dataset. Each row depicts the same point cloud; *Left:* SLR segmentation. *Right:* ECE segmentation.

processing step is to identify the obstacles as static or dynamic and perform tracking on the dynamic ones.

ACKNOWLEDGEMENTS

For the completion of this work, Dimitris Zermas has been supported by DELPHI. In addition, this material has been partially supported by the National Science Foundation through grants #CNS-0934327, #CNS-1439728, #IIS-1427014, #OISE-1551059, #CNS-1531330, and #CNS-1544887.

REFERENCES

[1] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[3] M. Himmelsbach, F. v. Hundelshausen, and H. J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, June 2010, pp. 560–565.

[4] F. Moosmann, O. Pink, and C. Stiller, "Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion," in *Intelligent Vehicles Symposium, 2009 IEEE*, June 2009, pp. 215–220.

[5] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3d lidar point clouds," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2798–2805.

[6] T. Chen, B. Dai, R. Wang, and D. Liu, "Gaussian-process-based real-time ground segmentation for autonomous land vehicles," *Journal of Intelligent & Robotic Systems*, vol. 76, no. 3, pp. 563–582, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10846-013-9889-4

[7] R. Tse, N. Ahmed, and M. Campbell, "Unified mixture-model based terrain estimation with markov random fields," in *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, Sept 2012, pp. 238–243.

[8] J. Byun, K.-i. Na, B.-s. Seo, and M. Roh, *Drivable Road Detection with 3D Point Clouds Based on the MRF for Intelligent Vehicle*. Cham: Springer International Publishing, 2015, pp. 49–60. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07488-7_4

[9] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," Ph.D. dissertation, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[10] PCL. Point cloud library. [Online]. Available: http://www.http://pointclouds.org/

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[12] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[13] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Transactions on Image Processing*, vol. 17, no. 5, pp. 749–756, 2008.