InsClustering: Instantly Clustering LiDAR Range Measures for Autonomous Vehicle

You Li1, Clément Le Bihan2, Txomin Pourtau2, Thomas Ristorcelli2

Abstract—LiDARs are usually more accurate than cameras in distance measuring. Hence, there is strong interest to apply LiDARs in autonomous driving. Different existing approaches process the rich 3D point clouds for object detection, tracking and recognition. These methods generally require two initial steps: (1) filter points on the ground plane and (2) cluster non-ground points into objects. This paper proposes a field-tested fast 3D point cloud segmentation method for these two steps. Our specially designed algorithms allow instantly process raw LiDAR data packets, which significantly reduce the processing delay. In our tests on Velodyne UltraPuck, a 32 layers spinning LiDAR, the processing delay of clustering all the 360° LiDAR measures is less than 1ms. Meanwhile, a coarse-to-fine scheme is applied to ensure the clustering quality. Our field experiments in public roads have shown that the proposed method significantly improves the speed of 3D point cloud clustering whilst maintains good accuracy.

I. INTRODUCTION

Perception systems of autonomous vehicles usually consist of cameras, LiDARs and radars [1] [2]. As an active sensor, LiDAR illuminates the surroundings by using laser devices. Distances are measured through analyzing the laser returns. Currently, the most popular LiDARs are mechanical spinning LiDARs (e.g. Velodyne HDL64), which scan a wide scenario through physically rotating lasers and receivers. LiDAR is much better than the camera in the accuracy of distances but poor in object recognition due to the sparsity of point clouds. Radar can work in all-weather conditions and is better in moving object detection and speed estimation. However, the resolution of radar is insufficient for complex environments. Therefore, LiDAR is always regarded as an important sensor to ensure safety for high level autonomous driving in complex environments.

In the perception systems for autonomous vehicles, LiDARs are generally used for object detection, tracking and recognition. Sometimes, LiDARs are used for Occupancy grid mapping [3] for path planning. [4] proposed a classic LiDAR perception pipeline to extract obstacles from 3D point cloud: (1) clustering: point clouds are firstly separated by ground segmentation. Then, non-ground points are grouped into different obstacles. (2) object recognition and tracking: obstacles are classified into semantic classes [5], [6] (e.g. pedestrian, vehicle, truck etc) and tracked through Bayesian filters [7].

In this paper, we focus on point cloud clustering, i.e. ground segmentation and object clustering. As the first step of the whole LiDAR perception pipeline, segmentation is required to be as fast and accurate as possible. For a Velodyne LiDAR working at 600RPM (rotation per minute), the processing time of the perception algorithms is usually expected to be less than 100ms. Considering that tracking and classification tasks for multiple objects are more complicated and require more time, the time budget for point cloud segmentation is quite limited.

The proposed approach "InsClustering" is able to instantly process the received raw UDP packets from a popular LiDAR: Velodyne UltraPuck. Range image in spherical coordinates is utilized to process the raw packets for ground segmentation and object clustering. Finally, a coarse-to-fine scheme is applied to maintain the clustering accuracy, especially mitigate the over-segmentation problem. The main contributions of this method are:

- A fast 3D point cloud segmentation for autonomous driving scenarios. The time delay of completing the segmentation of a full 360° scan of a 32 layers spinning LiDAR is less than 1ms on our platform (Intel i7-7820, 16G Memory, not rely on GPU), which is faster than two SOTA methods DepthClustering [8], and SLR [9].
- Good accuracy performance, especially in ameliorating over-segmentation problem, compared to the two SOTA methods.

II. LITERATURE REVIEW

As the first step of applying LiDAR into autonomous driving, the ground segmentation and object clustering have been well researched. The related methods can be roughly divided into:

2.5D grids based methods: By projecting 3D points into 2D xy plane, 2.5D grids (plus height information) are created. In the early works of [7], [10], the number of points, maximum height within a 2.5D polar grid and angles between consecutive grids are used for ground segmentation. The non-ground points are grouped through a chunk clustering method in [10]. The same 2.5D polar grids structure is kept in [4], but line segments are fitted to judge whether a grid belongs to ground or not. The left non-ground grids are clustered by the connected component labeling (CCL).

Spherical coordinates (range image) based methods: Instead of projecting onto a 2D ground plan, 3D points can be viewed through Spherical coordinates (r, ϕ, θ) according to characteristics of scanning LiDAR. For mechanical scanning
LiDAR, vertical angle for each laser beam is fixed, azimuth angle is decided by the scanning time and motor speed. Therefore, every range reading can be represented by $P_{i,j} = (\rho_{i,j}, \phi_{i,j}, \theta_{i,j})$, where $i$ refers to a certain laser beam, $j$ is the azimuth angle index. Avoiding the creation of 2.5D polar grids, this approach naturally fills the range readings into a predefined range image data buffer and allows quick access to a point and its neighbours. For example, [8] firstly segments the ground points in each column of a range image, and group the remaining points by criterions of distance and angle. For 32 beams LiDAR, they reached 4ms in an Intel i5 U5200 2.2 GHz CPU, C++ implementation under ROS. The ground segmentation in [9] is realized based on 2.5D grids, while for the object clustering, they firstly apply the clustering in each scan-line (actually the row in range image), and then merge the clusters scan-line by scan-line.

**Graph based methods:** this is more general than 2.5D grids and range image, usually coupled with graph based algorithms. For instance, [11] represents 3D point clouds by an undirect graph. Local plane normals are compared to cut the edges of graph to realize ground segmentation. However, it is not a real-time method. [12] proposed two different segmentation methods, Gaussian Process Incremental Sample Consensus (GP-INSAC) and a mesh-based method inspired by [11]. However, these two methods are not real-time. [13] models the ground as a spatio-temporal conditional random field, dividing the surrounding into connected grids. Heights of each grid are estimated concurrently by an EM variant algorithm implemented for GPU. Although the results are promising, this method is not real time.

**Deep learning based semantic segmentation:** In [14], the authors designed a neural network to semantically segment 3D point clouds. However, due to the sparsity of LiDAR data w.r.t distance, this method doesn’t work well for autonomous driving scenarios. [15] applies convolutional neural networks for semantic segmentation of objects from LiDAR point clouds. This method is capable of real-time processing thanks to the use of the range image.

### III. Methodology

#### A. Sensor model

The proposed method follows the principle of range image, as reviewed in Sec. II. All the laser beams fired from a spinning LiDAR can be considered to be emitted from an origin point $O$ in a spherical coordinate system: $(\rho, \varphi, \theta)$ (illustrated in Fig. 1 (a)), where $\rho$ refers to range measure, $\varphi$ and $\theta$ are the vertical and azimuth angles respectively. A range image $I_{N \times M}$ can be easily constructed as shown in Fig. 1(a). We define a point $p$ in range image $I_{N \times M}$:

$$p_{i,j} = [\rho_{i,j}, \varphi_{i,j}, \theta_{i,j}]^T \quad i \in [1, N], j \in [1, M]$$ (1)

Two state-of-the-art methods, SLR [9] and DepthClustering [8] both use depth image for point cloud clustering. While SLR process the range image row by row, and DepthClustering is column by column. Fig. 1 (b) shows an real example of a range image for the used Velodyne UltraPuck LiDAR.

#### B. InsClustering

The range image as used in SLR [9] and DepthClustering [8] provides a nature and memory-friendly representation for processing a spinning LiDAR’s outputs. Nevertheless, being different to a global shutter camera that captures an image at once, a spinning LiDAR continuously outputs the range measures during its scanning process. Generating a range image causes the problem of latency:

**Latency problem:** Waiting for a LiDAR to finish a full scan and then processing the range image leads to delay of object detection. In the worst case, when a Velodyne LiDAR is spinning at 600RPM (100ms for 360° scan), suppose there is an obstacle that appears at the very beginning of a scan. Then, we will wait almost 100ms for the range image creation and then pay for the algorithm processing time to finally receive the detected obstacle. Therefore, due to this defect of range image solution, a perception system would have late obstacle detections, which is critical for autonomous driving functions, such as autonomous emergency braking (AEB).

To cope with this problem, the proposed method InsClustering adapts traditional range image processing framework to directly process raw data flow from a spinning LiDAR. As denoted by the name, this method can instantly process the received LiDAR data and generate object detections without extra delay. The principle is shown in Fig. 2.

1) **Handle raw data packets:** In single return mode, the utilized Velodyne UltraPuck continuously outputs a stream of data packets $P_t$ via UDP protocol. The 1248 bytes long data packet $P$ comprises 12 data blocks, and each data block contains an azimuth angle and 32 range measures corresponding to the 32 lasers fired simultaneously. In fact, $P$ is a part of range image $I_{N \times M}$. Although in UltraPuck, the 32 lasers are not in the same azimuth angle, $P$ can still be viewed as several columns of $I_{N \times M}$, after accumulating enough data packets.
Fig. 2: Difference of processing range image and raw data flow. The end time of processing of range image based approach is later than the one of packet flow based approach.

Algorithm 1: Processing on LiDAR raw data flow

Input: Velodyne data packets flow
Output: Ground labels and clusters

/* Step 1: ground segmentation */
1 for every \( B_{N+M}^i \), built from data packets do
2 Coarse and fine ground segmentation in \( B_{N+M}^i \) as step 1 and 2 in Algorithm 2
end

/* Step 2: Clustering */
4 for every obstacle point \( p \{I^i\} \) in received \( B_{N+M}^i \), do
5 Coarse clustering within a searching region \( S(p \{I^i\}) \).
6 end
7 Refine the cluster buffer \( CB \) according to Algorithm 3
8 Output the refined clusters in \( CB \) when necessary.

Instead of processing \( I_{N+M} \), the new approach instantly processes a buffer of received data packets \( P^i \), and thus ameliorates the latency (as shown in Fig. 2). Let \( B_{N+M}^i \) be one packet buffer created by a number of sequentially received data packets, where \( N \) is the number of rows in \( I_{N+M} \), \( M' \) is the buffer length. After creating the packet buffer, the proposed algorithm can directly process it. Algorithm 2 presents the pseudocode of InsClustering.

Within InsClustering, ground segmentation and a coarse clustering are performed simultaneously for each received packet buffer. Since the size of obstacle could be bigger than the length of packet buffer, a cluster buffer is maintained to refine previous generated clusters and output completed clustering results.

2) Ground segmentation: A fast coarse segmentation is first performed in each input packets buffer \( B_{N+M}^i \), which is \( M' \) columns of a range image \( I_{N \times M} \). The algorithm starts from a virtual point at the very beneath of the LiDAR. The virtual point is decided by the vehicle’s height and it is always assumed to be on the ground. Then, following the increasing order of vertical angles, the algorithm sequentially accesses point by point. Inspired by [16], any input point is firstly evaluated to be a change point or not, based on its slope. A change point usually reveals a start of an obstacle. For a point \( p_{i,j} \), its slope \( \alpha_{i,j} \) is defined as \((z_{i+1,j} - z_{i,j})/(\rho_{i+1,j}^x - \rho_{i,j}^x)\). If \( \alpha_{i,j} \) is bigger than a threshold \( t_\alpha \), \( p_{i,j} \) is classified as a change point. This is described as isChangePoint() in Algorithm 2.

If \( p_{i,j} \) is not a change point, its previous class \( l_{i-1,j} \) is an important clue for decision. In [16], if a point following the change point has a big height decrease, it will be labeled as ground again. Nevertheless, we observe that, due to the occlusion, the points far from a change point are hard to be classified by simple height changes. Therefore, a temporary label uncertain point is given in this case. \( p_{i,j} \)'s range difference \( \Delta \rho_{i,j}^y \) is calculated with respect to its next point \( p_{i+1,j} \). If \( \Delta \rho_{i,j}^y \geq t_{\Delta \rho^y} \), the point will be labeled as uncertain. Otherwise, it will be labeled as change follow. This is described as function isClose() in Algorithm 2.

Fine ground segmentation: After the above step, change and ground points have high probabilities to be truly from obstacles or ground respectively. The uncertain points and change follow points need more investigations. In this step, 2D line segments fitting are utilized to refine the coarse ground segmentation. The points within one packet buffer are projected into 2D plane \( z - \rho^y \) for line fitting. Although these points are not in the same plane in reality, this is still a good approximation. This 2D approximation allows a faster line fitting, comparing to the 3D cases as in [9]. 2D line segments in \( z - \rho^y \) plane are:

\[
L_s : z = a_s \rho^y + b_s, \rho^y \in [\rho_{0,s}^y, \rho_{1,s}^y], s \in [1, S],
\]  

Where \( S \) is the total number of line segments in one block, \( a_s \) and \( b_s \) are slope and intercept, \( \rho_{0,s}^y, \rho_{1,s}^y \) are the start and end of the \( s \)th line segment.

Line segments are fitted sequentially [17] from the ground points or uncertain points. To assure the line segments are flat, the following results are valid only if \( a_s \) and \( b_s \) are within certain ranges. Then, distances from all points except the change points to the nearest line segment are calculated. If a distance exceeds a certain threshold \( t_{2Plane} \), the point is labeled as Obstacle. Otherwise, it is classified as ground. Notice that all the change points are directly labeled as obstacle points.
3) Clustering: We use connected component labeling (CCL) to quickly group points into initial clusters $\mathcal{C}$. From left to right, the algorithm sweeps each column from bottom to top. Given a point $p$, its search region $S(p)$ is defined as previous $N$ columns along the sweeping order. Its connectivity to a point $p'$ within $S(p)$ is decided by evaluating range difference $\Delta r^{xy}$. Although $\Delta r^{xy}(p, p')$ is not the Cartesian distance, we found it works very well when $S(p)$ is not big. If $\Delta r^{xy}(p, p')$ is smaller than a threshold $t_{\Delta r^{xy}}$, $p$ and $p'$ are connected and grouped into the same cluster. Otherwise, if all the points within $S(p)$ are not connected to $p$, a new cluster starting from $p$ is created. $S(p)$ is usually defined as several previous columns, i.e., 5, in received packet buffers. This method is described in Algorithm 3.

Refinement: The initial clustering can quickly group adjacent points together. However, it leaves the problem of over-segmentation untouched. The main cause of over-segmentation is the various reflectivities of different parts in one object. For example, due to windshield is transparent for LiDAR, vehicle’s window parts are easily over-segmented. Another common reason is the occlusion caused by small objects, such as a pole on the roadside.

To find potentially over-segmented clusters, we explore the linkages of initial clusters $\mathcal{C}$, based on their positions in polar coordinates. Given $\mathcal{C}_1$ and $\mathcal{C}_2$, they may be linked by contain, overlap or neighbour relation, as demonstrated in Fig 3. The conditions of these three linkages are defined as:

- **Contain**: $[\tilde{c}_1, \tilde{c}_2] \in [\tilde{c}_1^*, \tilde{c}_1^*]$ or $[\tilde{c}_2, \tilde{c}_2^*] \in [\tilde{c}_2^*, \tilde{c}_2^*]$
- **Overlap**: $\tilde{c}_1 \in [\tilde{c}_1^*, \tilde{c}_1^*], \tilde{c}_2 \notin [\tilde{c}_1^*, \tilde{c}_1^*]$ or vise versa.
- **Neighbour**: $\text{gap}(\tilde{c}_1, \tilde{c}_2) < t_{\text{merge}}$, where $\text{gap}(\cdot)$ computes the number of gap columns between two clusters.

The linked initial clusters are merged if they are close enough. The distance between two clusters is defined as the smallest minimum mutual point Cartesian distances within search regions respectively:

$$d(\tilde{c}_1, \tilde{c}_2) = \min_{n} d(p, p'), p \in S(\tilde{c}_1|\tilde{c}_2), p' \in S(\tilde{c}_2|\tilde{c}_1) \tag{3}$$

The search region is defined as the overlapping area or adjacent parts, as shown in Fig 3.

### IV. Experiments

The proposed method is implemented in C++ under ROS Kinetic framework, within a PC with an Intel CPU i7-7820, 16G memory. A Velodyne VLP32C is mounted on the roof of a Renault ZOE. We modified an open-source Velodyne driver to directly output raw data packets. The proposed method is compared with two other open-source methods: (1) Scan Line Run (SLR) [9] and (2) DepthClustering (DC) [8].
OSR and USR are defined in Eq. 4. TPR, FNR, (Under-segmentation suppression rate) into several ones. In contrast, under-segmentation detections. False Negatives (FNs) are the missed detections. False Positives (FPs) are the phantom detections. True Positives (TPs) are the correct segmentation. Over-segmentation (OSR) is the number of total “true objects” (usually acquired by annotation). True Positives (TPs) and False Negatives (FNs) are the missed detections. Over-segmentation means an object is segmented into several ones. In contrast, under-segmentation is the case of merging several objects into one segmentation. In general, higher OSR or USR means less over-segmentations or under-segmentation, which is preferred.

B. Test in Rouen dataset

The proposed methods are evaluated for a dataset recorded by our platform, in the center of Rouen, France. The recorded dataset contains 20 minutes around 12K frames of point clouds. We annotate all the objects being vertical to the ground. 4860 objects from 200 frames within 30 meters are manually extracted. This dataset also contains raw LiDAR data packets that enable the test of the proposed method.

Table I: Quantitative evaluation of three methods in Rouen dataset.

|                | Our    | SLR    | Depth Clustering |
|----------------|--------|--------|------------------|
| Precision      | 0.987  | 0.968  | 0.952            |
| Recall         | 0.991  | 0.987  | 0.990            |
| TPR            | 0.917  | 0.892  | 0.861            |
| FNR            | 0.008  | 0.010  | 0.007            |
| USR            | 0.982  | 0.985  | 0.988            |
| OSR            | 0.963  | 0.914  | 0.831            |

Table II: Benchmarking average processing time

|                | All       | Ground filtering | Cluster merging |
|----------------|-----------|------------------|-----------------|
| SLR            | 30.8ms    | 13.8ms           | 17.0ms          |
| Depth Clustering| 5.5ms     | 2.3ms            | 3.2ms           |
| Proposed InsClustering | 265us | 98us             | 167us           |

1) Parameter settings: The range images sizes for SLR and our method are 32 * 1800, for depth clustering, the size is 32 * 900. The parameters of SLR are tuned as: $N_{seg} = 4, N_{iter} = 5, N_{LPR} = 20, Th_{seeds} = 0.5m, Th_{dist} = 0.2m, Th_{run} = 0.3m$ and $Th_{merge} = 1m$, which are slightly different to the settings in [9]. For depth clustering, the parameters are optimized as: the window size for Savitsky-Golay smoothing: 5, ground removal angle threshold: $8^{\circ}$, clustering angle tolerance is $11^{\circ}$.

The packet buffer $B$ is set to 5 data packets. Other parameters are: $t_{\alpha} = 0.5, t_{\Delta_{x,y}} = 2m$. For fine ground segmentation, the block size $B = 20$, and $t_{se2line} = 0.2m$. In initial clustering, $t_{\Delta_{x,y}}$ is $1m$. For the search region $S(p)$, if $S(p)$’s range is less than 20m, $S(p)$ is the previous 5 columns. Otherwise, $S(p)$ is its previous 10 columns in the range image. In the refinement step, $t_{neighbour}, d(\hat{C}_i, \hat{C}_j)$, $t_{merge}$ are set to 5, the third minimum mutual point distance, and 0.8m respectively.

2) Results analysis: From the results shown in Tab. I our method achieves the best scores in Precision, Recall, and $TPR$. Depth clustering gets the least $FNR$. Our method is the best in OSR, which means less in over-segmentation. However, the price is the lowest USR, which means our method is apt to merge nearby objects. In opposite, depth clustering has the highest USR score while the lowest OSR that this method tends to split objects. SLR has balanced performances in all the metrics. As for the processing speed (in Tab. I), InsClustering is much faster than other methods that are based on range image. The reason is that, by processing directly on raw data flow, we separate the total processing time for a whole range image into many subsets. Hence, when the LiDAR finishes a circle of scanning, the processing of our method (raw data version) is almost finished. While for a range image based method, it is just the beginning.
3) Qualitative Examples: To directly understand the performance of the proposed method, we qualitatively demonstrate the segmentation results in Rouen dataset, as shown in Fig. 5 where some failure examples for over-segmentation and under-segmentation are demonstrated.

V. CONCLUSIONS

In this paper, we propose the InsClustering: a fast point cloud clustering method designed for processing raw data flow from spinning LiDAR. Thanks to this design, our method is proved much faster than two SOTA methods. Meanwhile, a coarse-to-fine framework is adopted in both ground segmentation and object clustering. Experimental results show the fast speed and good accuracy.

REFERENCES

[1] Y. Li and J. Ibanez-Guzman, “Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems,” IEEE Signal Processing Magazine, vol. 37, no. 4, 2020.
[2] Y. Li, Y. Qiao, and Y. Ruichek, “Multiframe-based high dynamic range monocular vision system for advanced driver assistance systems,” IEEE Sensors Journal, vol. 15, pp. 5433–5441, 2015.
[3] E. Capellier, F. Davoine, V. Frement, and Y. Li, “Evidential grid mapping, from asynchronous lidar scans and rgb images, for autonomous driving,” in 21st International Conference on Intelligent Transportation Systems (ITSC), 2018.
[4] M. Himmelsbach et al., “Lidar based 3D Object Perception,” in 1st International Workshop on Cognition for Technical Systems, 2008.
[5] A. Teichman, J. Levinson, and S. Thrun, “Towards 3D object recognition via classification of arbitrary object tracks,” in IEEE International Conference on Robotics and Automation, 2011.
[6] E. Capellier et al., “Evidential deep learning for arbitrary LIDAR object classification in the context of autonomous driving,” in IEEE Intelligent Vehicles Symposium (IV), 2019.
[7] A. Petrovskaya and S. Thrun, “Model based vehicle detection and tracking for autonomous urban driving,” Journal of Autonomous Robots, Springer, vol. 26, pp. 123–139, 2009.
[8] I. Bogoslavskiy et al., “Fast range image-based segmentation of sparse 3d laser scans for online operation,” in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016.
[9] D. Zermas et al., “Fast Segmentation of 3D Point Clouds: A Paradigm on LiDAR Data for Autonomous Vehicle Applications,” in IEEE International Conference on Robotics and Automation (ICRA), 2017.
[10] J. L. Leonard, “A perception-driven autonomous urban vehicle,” Journal of Field Robotics, vol. 25, pp. 727–774, 2008.
[11] F. Moosmann, O. Pink, and C. Stiller, “Segmentation of 3D Lidar Data in non-flat Urban Environments using a Local Convexity Criterion,” in IEEE Intelligent Vehicles Symposium (IV), 2009.
[12] B. Douillard et al., “On the segmentation of 3D LIDAR point clouds,” in IEEE International Conference on Robotics and Automation (ICRA), 2011.
[13] L. Rummelhard et al., “Ground estimation and point cloud segmentation using spatiotemporal conditional random field,” in IEEE Intelligent Vehicles Symposium (IV), 2017.
[14] C. R. Qi et al., “Pointnet: Deep Learning on Point Sets for 3D Classification and Segmentation,” in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
[15] B. Wu et al., “SqueezeSeg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud,” in IEEE International Conference on Robotics and Automation (ICRA), 2018.
[16] P. Chu, “A fast ground segmentation method for 3d point cloud,” Journal of Information Processing Systems, vol. 13, 2017.
[17] V. Nguyen, S. Gächter, A. Martinelli, N. Tomatis, and R. Siegwart, “A comparison of line extraction algorithms using 2d range data for indoor mobile robotics,” Autonomous Robots, 2007.
[18] M. O. Shin et al., “Real-time and accurate segmentation of 3-d point clouds based on gaussian process regression,” IEEE Transactions on Intelligent Transportation Systems, vol. 18, pp. 3363–3377, 2017.