An improved feature extractor for the Lidar Odometry and Mapping (LOAM) algorithm

Clayder Gonzalez
Department of Electrical Engineering
Universidad de Chile
Santiago, Chile
Email: claydergc@gmail.com

Martin Adams
Department of Electrical Engineering
Universidad de Chile
Santiago, Chile
Email: martin@ing.uchile.cl

***Abstract*—This work proposes an improved feature extractor for the Lidar Odometry and Mapping (LOAM) algorithm, which is currently the highest ranked algorithm in the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) visual odometry ranking. This article proposes and justifies the substitution of LOAM's current feature extraction method with the Curvature Scale Space (CSS) based feature extraction algorithm for the processing of 3D Point Cloud Data (PCD). The justification is based on in improvement of the repeatability of the detection of robust features for LOAM and an improvement in the associated computational cost. The LOAM's feature extractor and CSS feature extractor were tested and compared with simulated and real data including the KITTI visual odometry dataset using the Optimal Sub-Pattern Assignment (OSPA) and Absolute Trajectory Error (ATE) metrics. The results showed that LOAM based on the CSS feature extractor out performed that based on the original LOAM feature extractor with respect to these metrics.**

I. Introduction

Feature extraction is a critical task in feature-based Simultaneous Localization and Mapping (SLAM), which is one of the most important problems in the robotics community [2] [17]. An algorithm that solves SLAM using Lidar based features is the Lidar Odometry and Mapping (LOAM) algorithm [19]. This algorithm is currently regarded as the best performing SLAM algorithm according to the KITTI visual odometry benchmark [6].

The LOAM algorithm solves the SLAM problem through a feature matching approach and its feature extraction algorithm detects features classifying points of a point cloud as smooth or non-smooth. This classification is based on an equation that defines the level of smoothness for each point. However, this equation does not consider the range noise of the sensor. Therefore, if the lidar range noise is high, the LOAM feature extractor can confuse smooth and non-smooth points, causing the feature matching task to fail.

This work proposes the replacement of the original LOAM feature extraction algorithm with the Curvature Scale Space (CSS) algorithm [11]. The choice of this algorithm was made after studying various feature extractors in the literature. The CSS algorithm can potentially improve the feature extraction task in noisy environments because of its various levels of Gaussian smoothing. The replacement of the original LOAM feature extractor with the CSS algorithm was achieved by adapting the CSS algorithm to Velodyne 3D LiDAR data.

The LOAM and CSS feature extractors were tested and compared with simulated and real data including the KITTI visual odometry dataset using the Optimal Sub-Pattern Assignment (OSPA) [15] and Absolute Trajectory Error (ATE)[1] metrics. For all these datasets the CSS feature extraction performance was better than that of the LOAM algorithm in terms of the OSPA and ATE metric values.

II. LOAM Revisited

The LOAM algorithm can be divided into several blocks as shown in Figure 1. The Lidar block returns a raw point cloud

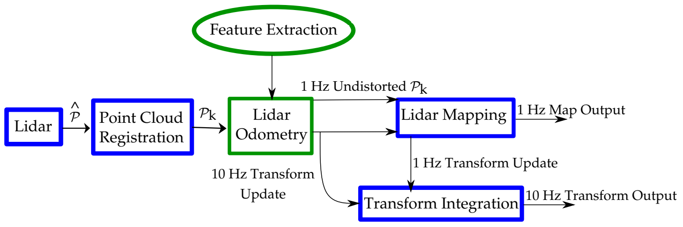


Fig. 1: The LOAM software system adapted from [19].

$\hat{\mathcal{P}}$. Then, the Point Cloud Registration block is responsible for organizing the point cloud by attaching to it a property called sweep k. This results in a point cloud $\mathcal{P}_k$. According to the original paper [19], a sweep k is a non-negative integer that labels each point cloud, based on its elevation angle - i.e. every 360 degrees. Therefore, when the algorithm starts, the first 360 degrees of rotation produces a point cloud $\mathcal{P}_1$, the second rotation a point cloud $\mathcal{P}_2$, etc. Additionally, the Point Cloud Registration block divides each $\mathcal{P}_k$ based on its scan. Since the Velodyne lidars (VLP-16, HDL-32, HDL-64) are composed of several individual lasers, a scan is the set of points returned by one laser. Subsequently, the Lidar Odometry block contains the feature extraction task. Based on these features, the LOAM algorithm estimates the motion of the lidar. The motion is estimated by matching the features belonging to two consecutive 3D scans to find the optimal transform between them. It is important to note that this task depends on the number of estimated features and how accurately located they are. The Lidar Odometry block constantly feeds the Lidar Mapping block with data at 1Hz and the Transform Integration block with data at 10Hz. These two frequencies are important because they determine how fast the map is

[1]http://www.rawseeds.org/rs/methods/view//9.

generated and how fast the transform is updated. Consequently, the feature extraction task is important for the Lidar Mapping and Transform Integration blocks.

A. *The LOAM feature extractor*

Before describing the LOAM feature extraction method, it is important to mention that this task processes each scan by dividing it into four identical sub-regions. A sub-region is a subset of points of each scan. Therefore, the LOAM feature extractor is responsible for classifying each point of all sub-regions as smooth (flat) or non-smooth (sharp) according to a threshold. This threshold is represented by C_{LOAM} and it is defined as

$$C_{\mathrm{LOAM}} = \frac{1}{|\mathcal{R}| \cdot ||p_{(i,k)}||} || \sum_{j \in \mathcal{R}, j \neq i} (p_{(i,k)} - p_{(j,k)}) ||, \qquad (1)$$

where $\mathcal{R}$ is the set of points from a sub-region of a scan ($\mathcal{R} \subset \mathcal{P}_k$). Additionally, $p_{(i,k)}$ and $p_{(j,k)}$ represent the ith and jth point vectors of the sweep k of $\mathcal{R}$. Additionally, $||\ ||$ represents the L_2 norm.

III. Feature Extractors for Point Clouds Commonly Used in SLAM

In the literature, general feature extractors, which are those that process any kind of point cloud, and feature extractors that are commonly used in feature-based SLAM solutions are presented. Among the general feature extractors, algorithms such as the Harris corner detector [7], the Scale Invariant Feature Transform (SIFT) keypoint detector [9] and the Intrinsic Shape Signatures (ISS) detector [18] are often used. According to F. Silvio [5], among these algorithms, the one that has the best performance in terms of repeatability of detected features is the ISS algorithm.

Among the feature extractors that are commonly used in feature-based SLAM solutions, notable algorithms such as the Split and Merge (SM) algorithm [13], the Split and Merge Fuzzy (SMF) algorithm [3], the Iterative End Point Fit (IEPF) algorithm [4], the Hough Transform (HT) algorithm [8], the Curvature Scale Space (CSS) algorithm [11] and the Adaptive Curvature Estimation (ACE) algorithm [12] exist. Most of these algorithms are used with 2D point clouds and they commonly extract lines, corners and circles. According to P. Nuñez et al. [12], among these algorithms, the one that has the best performance (recognition rate, accuracy of feature locations and processing time of the algorithm) is the ACE algorithm.

A. *General Feature Extractor Validity in LOAM*

To adopt the previously mentioned general feature extractors it is necessary to remember that Lidar Odometry block needs to feed the Transform Integration block with its data at a frequency of 10 Hz (Figure 1). Additionally, it is important to consider that these feature extractors are based on an algorithm called *radius search*. The radius search algorithm is responsible for obtaining a set of neighbors for each point within a sphere of a certain radius.

To analyze the complexity of the Harris, SIFT and ISS algorithms in terms of extracting features, an experiment was carried out in which these algorithms were evaluated with a point cloud composed of $n = 21572$ points. This point cloud was recorded in an indoor environment and obtained with a Velodyne VLP-16 LiDAR. The experiment consisted of varying the radius of the search volume for each algorithm and then recording the time each algorithm took to perform the task and noting how many features each algorithm returned. This experiment was carried out with the Point Cloud Library (PCL) [14] and a computer Intel Core i7-7500U CPU @ 2.70GHz $\times$ 4.

Figure 2 shows the time (in seconds) each algorithm takes to process the point clouds and how many features are extracted by each algorithm. The time is represented on

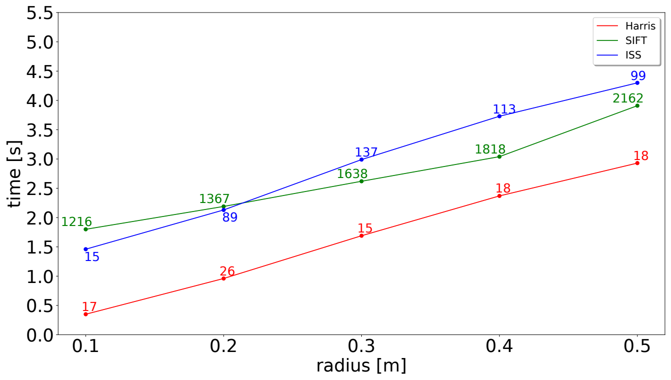


Fig. 2: Processing time of general feature extractors based on their search radius.

the vertical axis from 0 to 5.5 seconds and the number of features extracted by each algorithm for each radius is located next to the colored points. In this figure, it can be seen that the feature extractors take approximately 0.25 to 4.3 seconds to extract features from the point cloud. Additionally, except for the SIFT algorithm, the algorithms return less than 150 features. This is a low quantity of features compared to the LOAM feature extractor, which returns 3000 features per sweep, approximately. Hence, including any of these algorithm in the LOAM algorithm would be too slow for the Lidar Odometry block.

B. *SLAM based Feature Extractor Validity in LOAM*

SLAM based feature extraction algorithms were included in the comparison by P. Nuñez et al. [12]. The feature extractors had to detect corners, circular segments, line segments and edges. Table I gives a comparison of these algorithms in terms of run time, True Positive Rate (TPR) and False Positive Rate (FPR).

TABLE I: Feature Extraction Algorithm Comparisons According to P. Nuñez et al. [12].

Algorithm	SM	SMF	IEPF	HT	CSS	ACE
Time (ms)	6.7	14.2	3.8	31.2	37.7	9.4
TPR	0.76	0.79	0.77	0.71	0.92	0.91
FPR	0.19	0.19	0.21	0.23	0.02	0.02

An important characteristic of this table is that the first four algorithms have a low TPR. This occurs because these feature extractors used polygons to represent circular segments. On the other hand, the SM, SMF, IEPF and ACE algorithms' processing time values were the lowest and the HT and CSS algorithms' processing time values were the highest. With this information, we observed that the CSS and ACE algorithms had the best performance.

Since these processing time values are compatible with those necessary in the Lidar Odometry Block, we proceed to choose between the CSS and ACE feature extractors. The use of the ACE feature extractor was first considered, however it was discarded because the point cloud data obtained from Velodyne lidars presented a higher range noise compared to the data processed in the original paper [12], which introduced the ACE algorithm. Therefore, the CSS feature extractor was chosen to replace the original LOAM feature extractor.

IV. Methodology and Implementation

To record the data we used the mobile robot Husky A-200, a computer with Intel Core i7-7500U CPU @ 2.70GHz $\times$ 4, a Velodyne VLP-16 LiDAR and a Topcon Hyper V GPS only for trajectory ground truth verification purposes. Then, the data obtained with this hardware was compared with that obtained from two versions of the LOAM algorithm, one which includes the LOAM feature extractor and the other which replaces it with the CSS feature extractor. Finally, the estimated feature positions and their trajectories produced by both algorithms were compared with ground truth features and trajectories, respectively. For the comparison of features, the Optimal Sub-Pattern Assignment (OSPA) metric was used. It is important to note that in this case the comparison was only carried out for the simulated data. Finally, the comparison of trajectories was carried out using the Absolute Trajectory Error (ATE) metric for both simulated and real data based experiments.

Before implementing the CSS algorithm, it is important to note that this feature extractor was used to extract only non-smooth features. Smooth features are extracted using the original LOAM feature extractor (Equation 1). Therefore, the first step necessary to implement the CSS algorithm is to divide the whole point cloud from the Velodyne VLP-16 into its 16 elevational scans. Figure 3 shows a zoomed view of a segment showing its 16 elevational scan components as different colors. This segment corresponds to $\mathcal{S}_1$, which can be seen from a top view in Figure 4. The subsequent step is to divide the point clouds from each scan into segments. The objective of this is to facilitate the detection of curvature extrema with the CSS algorithm and to decrease its computational complexity. The algorithm used in this work is called the Adaptive Breakpoint Detector [3].

The adaptive breakpoint detector detects a breakpoint based on the comparison of the current point and the previous point. For these points, the Euclidean distance between them is compared with a threshold. If the Euclidean distance is greater than the threshold, then a new segment is created.

An example of how a point cloud from one scan projected on an xy plane is divided into segments of different colors is shown in Figure 4. In this figure, $\mathcal{S}_j$ represents a set of points of the jth segment ($\mathcal{S}_j \subset \mathcal{P}_k$).

At this point, we have several point cluster segments per scan. With the data in this form, we can use the CSS algorithm to detect the curvature extrema. To use this algorithm, first we have to parameterize the curve segment as proposed by [10] to obtain different t_i values that vary from 0 to 1 as shown in equations 2, 3 and 4.

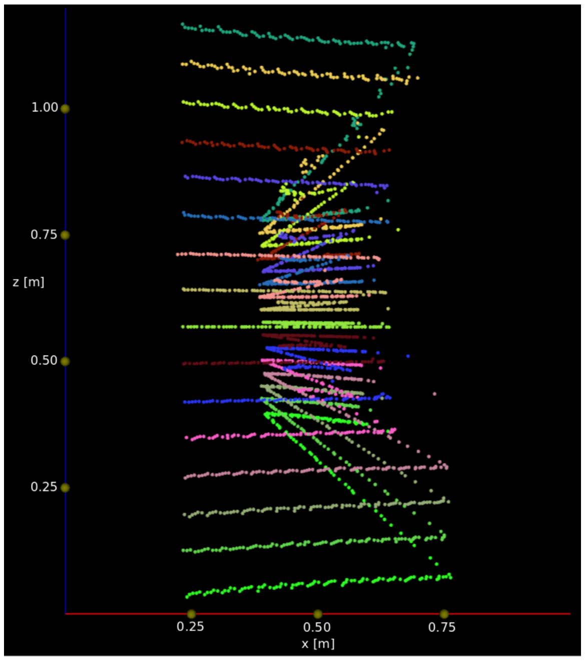


Fig. 3: Zoomed view of a segment of a 3D point cloud.

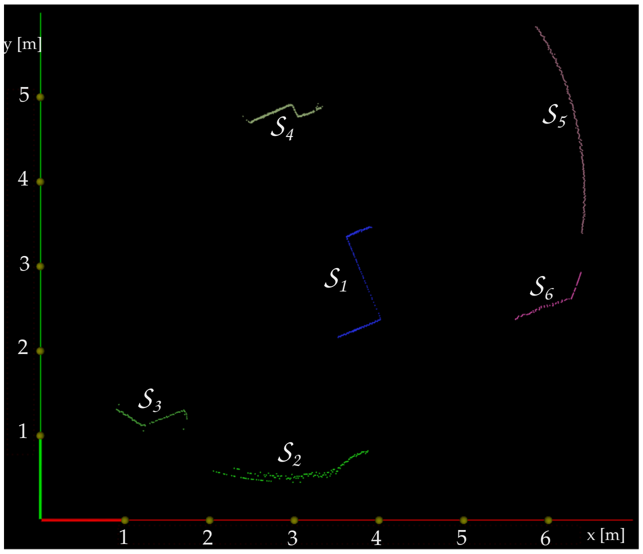


Fig. 4: Scan of a point cloud divided into six segments.

$$d_i = \begin{cases} 0, & \text{if } i = 1 \\ \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}, & \text{otherwise,} \end{cases} \tag{2}$$

$$D_p = \sum_{i=1}^{n} d_i, \tag{3}$$

$$t_i = \begin{cases} 0, & \text{if } i = 1 \\ \frac{1}{D_p} \sum_{j=1}^{i} d_j, & \text{otherwise.} \end{cases} \tag{4}$$

In the previous equations, the sub-index i varies from 1 to n, where n is the number of points of the curved segment

$\mathcal{S}_j$. Additionally, t_i represents the ith element of the set that represents the variable t. After the parameterization, the point cloud segment is expressed in its Euclidean components x and y as a function of t using the equation

$$C_{\text{EUCLID}} = \{x(t), y(t)\}. \tag{5}$$

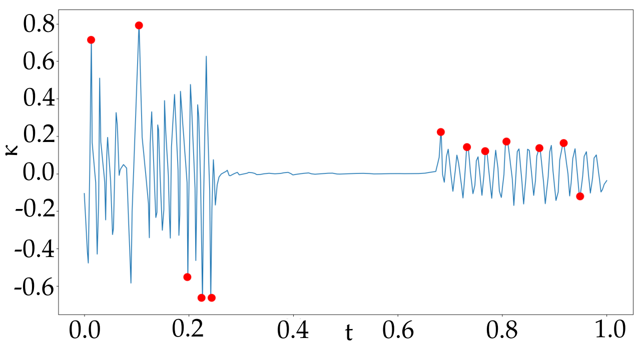
It is then necessary to convolve $x(t)$ and $y(t)$ with the Gaussian derivatives, which is also known as Gaussian smoothing. These convolutions are represented by $\dot{X}(t,\sigma)$, $\ddot{X}(t,\sigma)$, $\dot{Y}(t,\sigma)$ and $\ddot{Y}(t,\sigma)$. Finally, the results of previous convolutions are used to compute the curvature $\kappa(t,\sigma)$ for each point of each segment in the 16 scans using the equation

$$\kappa(t,\sigma) = \frac{\dot{X}(t,\sigma)\ddot{Y}(t,\sigma) - \dot{Y}(t,\sigma)\ddot{X}(t,\sigma)}{(\dot{X}(t,\sigma) + \dot{Y}(t,\sigma))^{3/2}}. \tag{6}$$

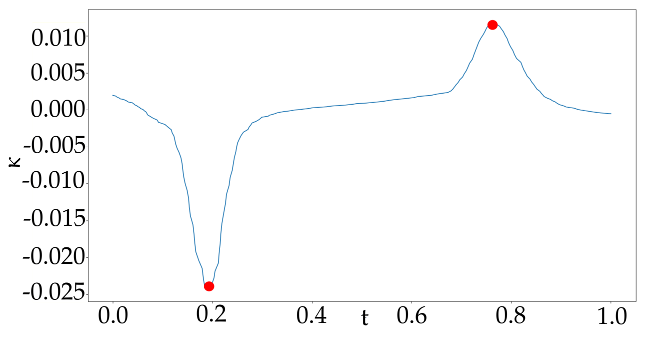
The following task is to find the curvature extrema in the functions κ. This task should be time efficient. Therefore, an algorithm that does not increase the computational complexity of the scale space is required. In this work, we used a simple approach proposed by R. Madhavan and H.Durrant-Whyte [10] given by

$$|\kappa_{i-1}| < |\kappa_i| > |\kappa_{i+1}| \quad \text{and} \quad |\kappa_{i-2}| < |\kappa_i| > |\kappa_{i+2}|. \tag{7}$$

In the previous conditions, the curvature extrema is found by comparing the current curvature value, κ_i, with its two neighbors on the left κ_{i-1}, κ_{i-2} and right κ_{i+1}, κ_{i+2}. The result of finding the curvature extrema can be seen in Figure 5. In this figure, the curvature extrema are marked with red dots for two curvature functions $\kappa(t,\sigma)$ with different values of σ (Figs. 5(a) and 5(b)).



(a) Curvature function κ of segment $\mathcal{S}_1$ for $\sigma = 0.1$



(b) Curvature function κ of segment $\mathcal{S}_1$ for $\sigma = 0.8$

Fig. 5: Curvature functions κ of segment $\mathcal{S}_1$ for two values of σ.

With the curvature extrema, we can create the scale space as shown in Figure 6. To extract the features in the scale space,

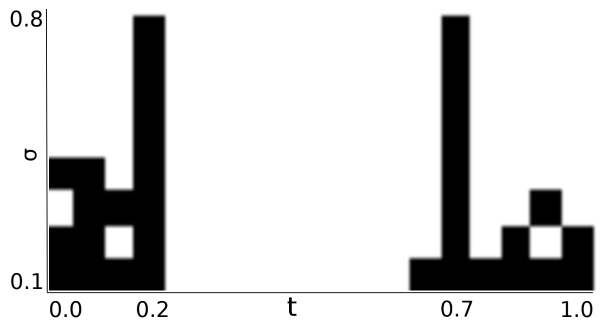


Fig. 6: Curvature scale space created with the functions κ shown in Figure 5.

we have to search for the curvature extrema that survived the Gaussian smoothing. Therefore, we need to iterate the whole scale space. To do this, we have to consider an important note made by Asada and Brady [1], which is also mentioned in [10]. This note states that curvature changes found at different scales, σ, are reliably localized at σ_{min}. However, if they are found at σ_{min} and do not persist across many scales, they are less likely to determine the shape of the curve. Finally, the segment $\mathcal{S}_1$ is composed of two features, one at $t = 0.2$ and the other at $t = 0.7$.

V. Results

To obtain the results, we divided the experiments into three parts based on simulated data, real data and the KITTI dataset. In all the experiments the ATE metric was used to determine the LOAM trajectory error. In the case of the simulated data, the OSPA metric was also used to determine the mapping error. From here on, the LOAM algorithm using its original feature extractor will be referred to as the Original LOAM algorithm and the LOAM algorithm using the CSS feature extractor as the CSS LOAM algorithm. Additionally, it is important to note that the implementation of the original LOAM algorithm was based on the code uploaded by the original author to the ROS Indigo website[2].

A. Simulated data

The purpose of the simulated data experiments is to control the range noise added to the measurements of the simulated lidar. The tested range noise standard deviation (σ_r) values were 0.01 m, 0.02 m and 0.03 m, however, only the results for $\sigma_r = 0.02$ m will be shown. Additionally, for this scenario, we used the Gazebo simulation software to simulate the motion of the Husky A-200 robot platform and the data acquisition of the Velodyne VLP-16 LiDAR.

This scenario was composed of the robot platform mounted with the lidar located at $(x\ y\ z)^T = (0\ {-10}\ 0)^T$. Additionally, a set of walls forming 12 corners was used together with boxes of cubic shape forming 28 corners. All the corners represent sharp features. The layout of this scenario can be seen in Figure

[2]http://docs.ros.org/indigo/api/loam_velodyne/html/files.html.

7 in which the ground truth features are marked with light blue stars and the predefined trajectory is represent with a blue dashed line.

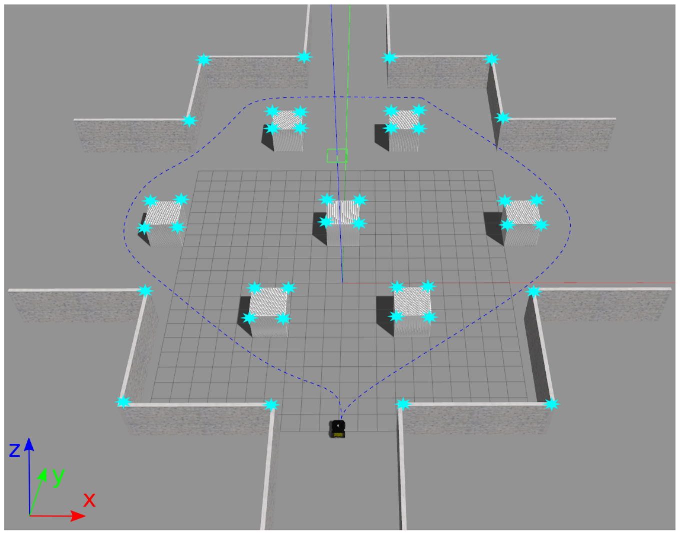


Fig. 7: Scenario with simulated data.

The first metric used was the OSPA metric with cutoff parameter $c = 5$m and power $p = 2$ [15], to determine the LOAM mapping error as shown in Figure 8 with $\sigma_r = 0.02$ m. In this figure, the metric values are separated by an average

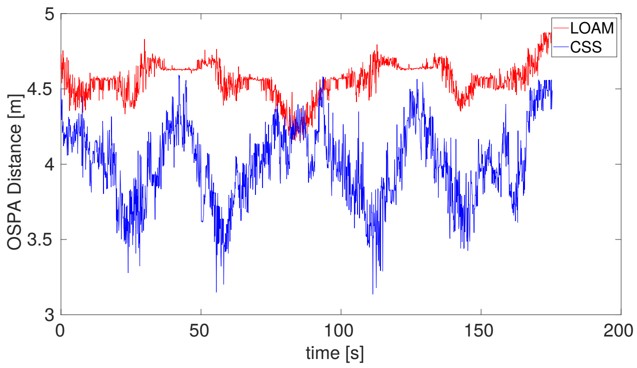
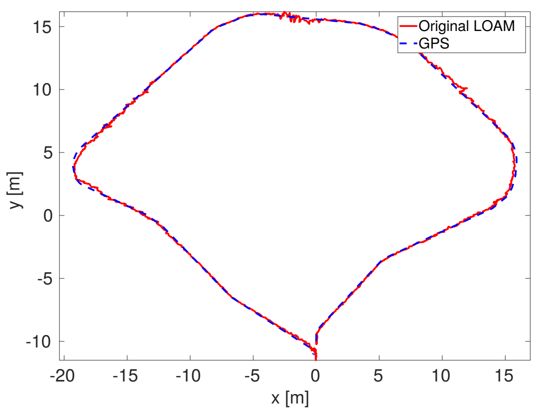


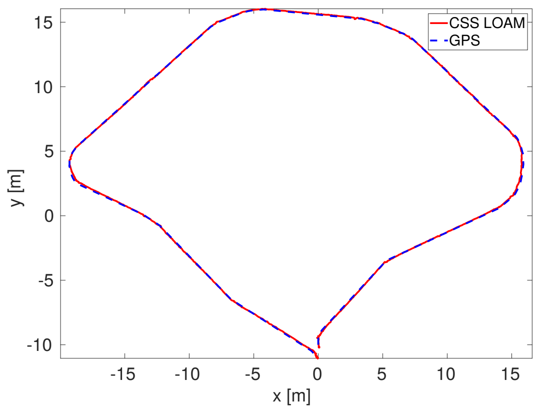
Fig. 8: LOAM feature estimation error using the OSPA metric ($\sigma_r = 0.02$ m).

value of 0.56 meters and reach a maximum difference of 1.62 meters.

The second metric used in this scenario was the ATE trajectory metric. As occurred with the previous metric, the most important finding occurred when using $\sigma_r = 0.02$ m. Figures 9a and 9b show the estimated trajectories of the original LOAM and the CSS LOAM algorithms, respectively. In these figures, the estimated trajectory with the original LOAM algorithm presents high error values in the area near to the point $x = 0$, $y = 15$. This trajectory error is also represented in the error in y at time 100 s shown in Figure 10a. In Figure 10a, the mean error of the original LOAM algorithm was equal to 0.12 meters, while in Figure 10b the mean error of the CSS LOAM algorithm was 0.06 meters.



(a) Estimated trajectory using the original LOAM algorithm.



(b) Estimated trajectory using the CSS LOAM algorithm.

Fig. 9: Estimated trajectories compared to ground-truth using simulated data ($\sigma_r = 0.02$ m).

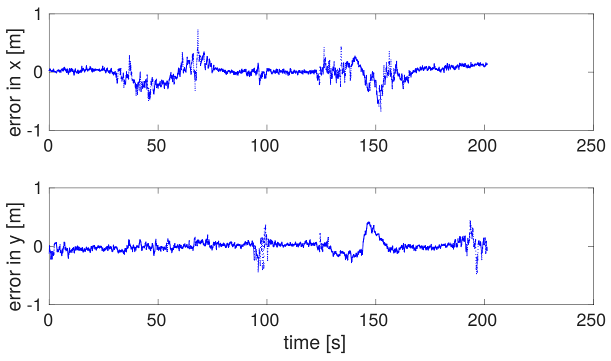
B. *Real data*

The environment used to conduct the real experiment was the surroundings of Universidad de Chile - Campus Beauchef, located in Santiago de Chile's downtown area. This scenario is mainly composed of buildings and trees which can feed the LOAM and CSS feature extraction algorithms with a variety of shapes.

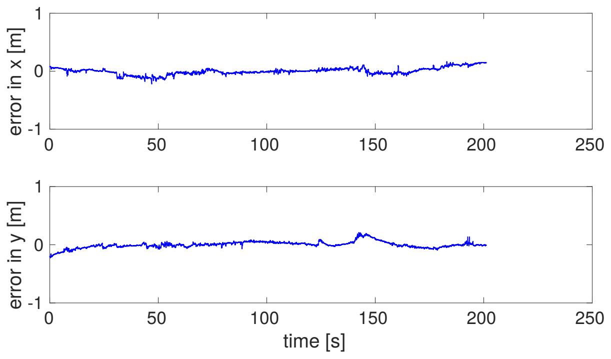
Figure 11 shows the complete trajectory of the original LOAM algorithm (red line, Figure 11a) and the CSS LOAM algorithm (red line, Figure 11b) compared to the ground truth trajectory generated by the GPS (blue dashed line). In Figs. 11(a) and (b), the estimated trajectories of the circular area located at the bottom are shown separately as zoomed views. This is to show that the trajectory of the CSS LOAM algorithm is aligned better with the blue line than the trajectory of the original LOAM algorithm.

C. *KITTI Dataset*

The results of the original LOAM algorithm and the CSS LOAM algorithm using the KITTI visual odometry dataset varied in some sequences. It is important to note that the KITTI dataset provides the point cloud data as individual files corresponding to each full scan of the environment. The original LOAM algorithm processes each of these files at its own rate, i.e. not necessarily in real time. Therefore to more



(a) Errors in x, y produced by the Original LOAM algorithm.



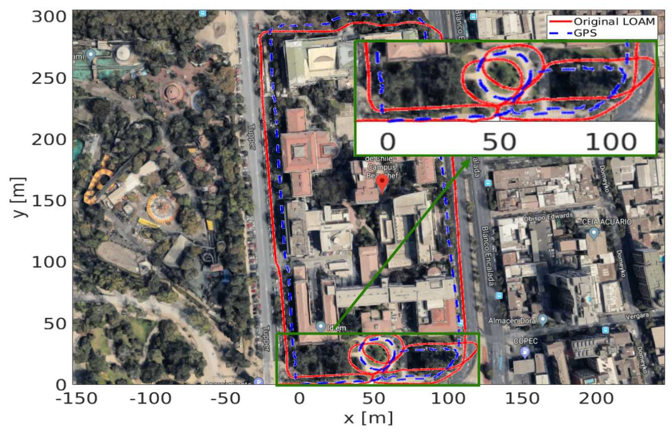
(b) Errors in x, y produced by the CSS LOAM algorithm.

Fig. 10: Errors in x, y of the original LOAM algorithm and the CSS LOAM algorithm using simulated data ($\sigma_r = 0.02$ m).

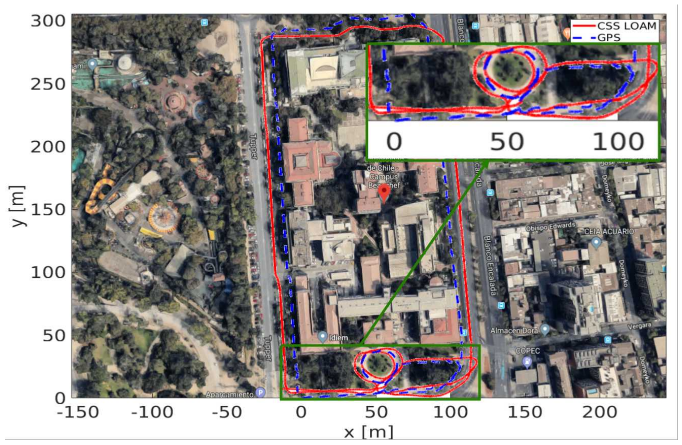
realistically evaluate the performances of both the original and CSS LOAM algorithms, the KITTI data set files were converted to data scan files, which were produced at the frame rate of the Velodyne HDL-64E lidar - i.e. 10 frames per second. The impact of this lidar data frame rate on the original LOAM algorithm is not analyzed in the KITTI ranking, but is considered here. It will be shown that the original LOAM algorithm is not fast enough to process full Velodyne HDL-64E data at the frame rate of the sensor.

An notable difference in results occurred when testing sequence 06 of the KITTI dataset. Figure 12 shows the trajectory of sequence 06 estimated with the original LOAM algorithm (red line, Figure 12a) and the CSS LOAM algorithm (red line, 12b) compared to ground truth (blue lines) with the real sampling frequency of the lidar, as stated before. Figure 13 shows the ATE error of the estimated trajectories. In these figures, we can observe that the lowest error in the x Euclidean component belongs to the CSS LOAM algorithm (Figure 13b). In the case of the y component error the CSS LOAM algorithm also shows a slight improvement over its original LOAM counterpart.

The explanation for the results shown in Figures 12 and 13 are related to the LOAM algorithms' processing times. As described in section II, the performance of LOAM's Lidar Odometry block depends on the number of features extracted. Therefore, we decided to see if this number affected the original implementation of the LOAM algorithm. Figure 14 shows that some point clouds ($\mathcal{P}_k$) of the KITTI dataset were not being processed (referred to as "Quantity of dropped $\mathcal{P}_k$") by the original implementation of the LOAM algorithm. This was producing incorrect trajectories when matching the features.

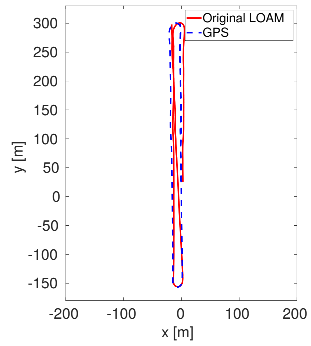


(a) Estimated trajectory using the original LOAM algorithm.

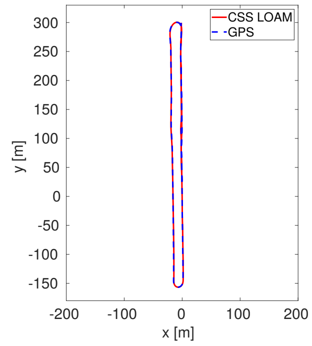


(b) Estimated trajectory using the CSS LOAM algorithm.

Fig. 11: Estimated trajectories using real data.



(a) Estimated trajectory using the original LOAM algorithm.

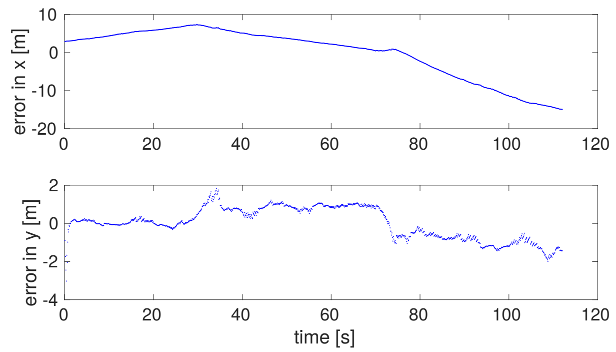


(b) Estimated trajectory using the CSS LOAM algorithm.

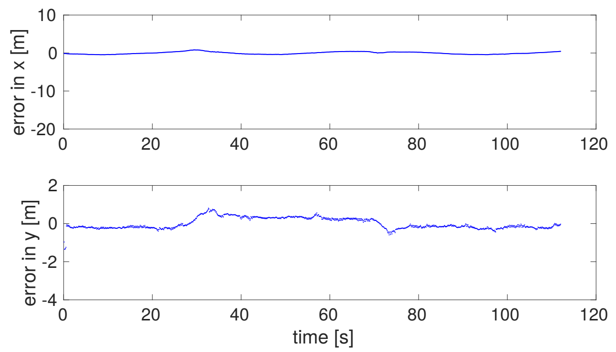
Fig. 12: Estimated trajectories compared to ground-truth in the sequence 06.

VI. Conclusions and Future Work

This article addressed the problem of feature misdetection and inability to account for sensor range noise in the LiDAR Odometry and Mapping (LOAM) algorithm. As a solution,



(a) Errors in x, y produced by the Original LOAM algorithm.



(b) Errors in x, y produced by the CSS LOAM algorithm.

Fig. 13: Errors in x, y of the original LOAM algorithm and the CSS LOAM algorithm in the sequence 06.

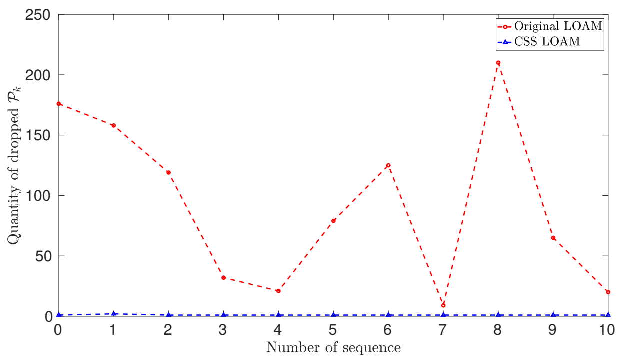


Fig. 14: Quantity of dropped point clouds, $\mathcal{P}_k$, per sequence.

the replacement the original LOAM feature extractor with the Curvature Scale Space (CSS) algorithm was proposed.

The original LOAM and CSS LOAM algorithms were compared using simulated and real data including the KITTI dataset. The Optimal Sub-Pattern Assignment (OSPA) and the Absolute Trajectory Error (ATE) metrics were used to gauge algorithmic performances. Both metrics demonstrated the superior performance of the proposed CSS LOAM algorithm.

A significant difference in the original and proposed CSS LOAM results occurred when using the KITTI dataset. The original LOAM algorithm produced high ATE errors in various data sequences. A major factor which contributed to this poor performance was the inability of the original LOAM algorithm to process data from the Velodyne HDL-64E at frame rate, resulting in "dropped point clouds" and therefore a loss of information. This problem did not occur with the CSS LOAM algorithm due to its lower computational complexity.

Finally, the inclusion of the CSS feature extractor in other SLAM algorithms such as the LeGO-LOAM algorithm [16] could be considered. The LeGO-LOAM algorithm is a modified version of the LOAM algorithm which is optimized for a horizontally placed LiDAR on a ground vehicle. The main difference between the LOAM algorithm and the LeGO-LOAM algorithm is that the LeGO-LOAM algorithm assumes that there is always a ground plane in the current 3D scan.

References

[1] Haruo Asada and Michael Brady. The Curvature Primal Sketch. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):2–14, 1986.

[2] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part I. *IEEE Robotics and Automation Magazine*, 13(3):108–117, 2006.

[3] Geovany Araujo Borges and Marie-José Aldon. Line Extraction in 2D Range Images for Mobile Robotics. *Journal of Intelligent and Robotic Systems*, 40:267–297, 2004.

[4] David H. Douglas and Thomas K. Peucker. *Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature*, chapter 2, pages 15–28. John Wiley & Sons, Ltd, 2011.

[5] Filipe Silvio and Alexandre Luis. A Comparative Evaluation of 3D Keypoint Detectors. In *9th International Conference on Computer Vision Theory and Applications*, pages 145–148, 2014.

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] C. Harris and M. Stephens. A Combined Corner and Edge Detector. *Procedings of the Alvey Vision Conference 1988*, pages 23.1–23.6, 1988.

[8] P.V.C. Hough. Method and means for recognizing complex patterns, 1962.

[9] David G Lowe. Distinctive Image Features from Scale Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[10] R. Madhavan and H. F. Durrant-Whyte. Natural landmark-based autonomous vehicle navigation. *Robotics and Autonomous Systems*, 2004.

[11] Farzinand Mokhtarian and Alan Mackworth. Scale-Based Description and Recognition of Planar Curves and Two-Dimensional Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(1), 1986.

[12] P. Núñez, R. Vázquez-Martín, J. C. del Toro, A. Bandera, and F. Sandoval. Natural landmark extraction for mobile robot navigation based on an adaptive curvature estimation. *Robotics and Autonomous Systems*, 56(3):247–264, 2008.

[13] Theodosios Pavlidis and Steven L. Horowitz. Segmentation of Plane Curves. *IEEE Transactions on Computers*, C-23(8):860–870, 1974.

[14] Radu Bogdan Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). *IEEE International Conference on Robotics and Automation*, pages 1 – 4, 2011.

[15] Dominic Schuhmacher, Ba Tuong Vo, and Ba Ngu Vo. A consistent metric for performance evaluation of multi-object filters. *IEEE Transactions on Signal Processing*, 56(8 I):3447–3457, 2008.

[16] Tixiao Shan and Brendan Englot. LeGO-LOAM : Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[17] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):1999–2000, 2002.

[18] Zhong Yu. Intrinsic shape signatures: A shape descriptor for 3D object recognition. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009*, pages 689–696, 2009.

[19] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference*, July 2014.