PAPER *Special Section on Intelligent Transport Systems*

An Open Multi-Sensor Fusion Toolbox for Autonomous Vehicles

Abraham MONRROY CANO[†a)], **Eijiro TAKEUCHI**[†], **Shinpei KATO**[††], ***Nonmembers***, ***and*** **Masato EDAHIRO**[†], ***Member***

SUMMARY We present an accurate and easy-to-use multi-sensor fusion toolbox for autonomous vehicles. It includes a 'target-less' multi-LiDAR (Light Detection and Ranging), and Camera-LiDAR calibration, sensor fusion, and a fast and accurate point cloud ground classifier. Our calibration methods do not require complex setup procedures, and once the sensors are calibrated, our framework eases the fusion of multiple point clouds, and cameras. In addition we present an original real-time ground-obstacle classifier, which runs on the CPU, and is designed to be used with any type and number of LiDARs. Evaluation results on the KITTI dataset confirm that our calibration method has comparable accuracy with other state-of-the-art contenders in the benchmark.

key words: *LiDAR, cameras, sensor fusion, calibration, autonomous driving, ground detection*

1. Introduction

The field of autonomous vehicles has undergone amazingly fast development in recent years. Demonstrations from software giants such as Google, NVIDIA and Intel show that we are closer than ever to achieve the deployment of fully autonomous driving systems on general roads. Nevertheless, these solutions are completely closed, not providing feedback to the research community.

The importance of the benefits to society of the deployment of autonomous vehicle technology has been widely discussed [1]. Reduction of road accidents, safe mobility for the elderly, independent transportation for the disabled, and reduction of traffic congestion are just a few of the potential benefits promised by the adoption of autonomous driving technology.

An autonomous vehicle requires several sensors to understand its surroundings, and act accordingly in different scenarios [2]. Images from camera devices, range data from LiDARs, speed information from radars, and other sensor data is fused to achieve single-digit centimeter-level accuracy of the object detection. Thanks to the fast development of autonomous driving technologies, the cost of these sensors is rapidly reducing.

A lot of research has been carried out in the calibration, fusion and classification field. However, most existing work focus on a unique sensor type (i.e. rotating LiDAR), or are not designed to function with multiple sensors simultaneously.

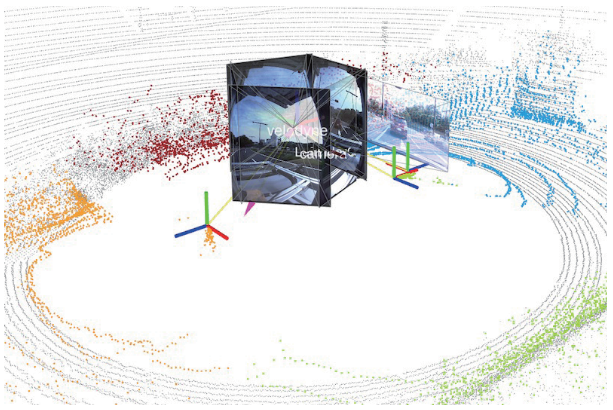


Fig. 1 Successful calibration result of six wide-angle cameras, one rotating LiDAR and four MEMS (micro electro-mechanical systems) LiDARs. Each LiDAR is shown in different color.

The integration of multiple LiDARs and cameras allows the system to benefit from redundant, complementary and timely information. Moreover, multiple sensors distributed around the vehicle improve the field of view, and hence the safety of the system and its users. Nevertheless, increasing the number of sensors requires better synchronization, fast and reliable fusion techniques, and optimized processing methods due to the increased bandwidth. Figure 1 shows a successful integration of multiple LiDAR and camera sensors of different types using this work.

In this paper, we present an open and easy-to-use perception framework for multi-LiDAR, multi-camera calibration, fusion, and a point cloud ground classifier. Our experience with autonomous vehicles has showed that these phases are the foundation for any higher level perception system, whether it is based on traditional machine learning [3] or deep learning. New approaches such as [4]–[6] are good examples of state-of-the-art systems that require a solid calibration and fusion framework. In summary, the main contributions of this work are as follows:

- The proposal of a reliable LiDAR-to-LiDAR extrinsic calibration based on 3D shape matching.
- Implementation of an easy-to-use "target-less" Camera-to-LiDAR extrinsic calibration method built around a user interface.

Manuscript received March 14, 2019.
Manuscript revised June 25, 2019.

[†]The authors are with Graduate School of Information Science, Nagoya University, Nagoya-shi, 464-8603 Japan.
[††]The author is with Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 113-8656 Japan.
a) E-mail: amonrroy@ertl.jp

DOI: 10.1587/transfun.2019TSP0005

Table 1 Feature comparison among tested Camera-LiDAR algorithms.

Method	Open-source	Data Available	ROS Compatible	Target-Type	LiDAR Type	Camera Type
Geiger, et al. [18]	X	O	X	Chessboard	Rotating	Single
Naroditsky, et al. [12]	X	X	X	Blackboard	Rotating	Single
Velas, et al. [13]	O	X	O	Custom board	Rotating	Single
Weimin, et al. [15]	O	O	X	Chessboard	Rotating	Single
Pandey, et al. [16]	O	O	X	Target-less	Rotating	Omni
This work	O	O	O	Target-less	any	any

- A real-time pixel to point cloud fusion algorithm.
- The proposal of an accurate, and real-time ground classifier for point cloud that works regardless the type and number of LiDARs.

This paper is arranged as follows: Section 2 presents previous work on the fields related to each of the parts of our toolbox. Section 3 contains the theory and implementation details for each component. Section 4 shows the evaluations we carried out and their results. Afterwards, Sect. 5 analyses these results and presents our findings. Finally, Sect. 6 summarizes and suggests the follow-up work required to improve our framework.

2. Related Work

Extrinsic calibration can be defined as the process of calculating the relative position in space between sensor coordinate frames. This can be achieved by looking for co-observable features in the data from both sensors. Calibrating different sensors may use similar optimization methods. However, the features to search for are dependent on the sensor. In the following subsections we will summarize some of the most recent developments in each area.

2.1 LiDAR-LiDAR Extrinsic Calibration

To the best of our knowledge, at the time of writing, very few publications can be found that address the calibration of multiple multi-layer LiDAR sensors. In [7] a method to calibrate a single-layer LiDAR is presented. This work maximizes the mutual information entropy, through the estimation of the projection coefficients between the spaces.

The work presented in [8], shows a semi-automatic extrinsic calibration method for multiple LiDARs and cameras. In this work, features are inserted in the field of view of the sensors with the help of spheres. These are detected with the help of the PointCloud Library (PCL) [9] segmentation and sphere fitting toolbox. The rigid body transformation is calculated using the Iterative Closest Point (ICP) algorithm [10].

2.2 Camera-LiDAR Extrinsic Calibration

While there is extensive work in this field, most of it focuses on the calibration of a single LiDAR and a camera. In this subsection we can classify the available methods into two:

1. The target category requires predefined and specific setups to ease the identification of shared features between the sensors. Under this class we can find notable mentions such as [11], [12], [13], [14], [8], and [15], which take advantage of the purposely inserted features, fixing the number of targets and optimizing the calibration algorithm under these constraints. The work presented by [8], [11], and [15] require the construction of a setup consisting of several specific targets, and scatter them across the shared field of view of the camera and the LiDAR. This approach works well in laboratories and closed environments. However, its application is difficult in the field. Moreover, all the mentioned work, except in [15], are not open-sourced, reducing their impact and reachability.
2. The target-less methods focus on finding inherent features to the scene in both sensors. In this category [16] is a notable mention. It requires an initial rough position estimate between the sensors, provided by the user. In this position, it generates an image projection of the LiDAR reflectance values using the given camera projection matrix. It then slowly modifies the transformation to try to match the generated reflectivity image with the camera gray-scale image. To measure the error, it compares the camera brightness histogram, and the LiDAR's reflectivity histogram. Since this method relies completely on the reflectivity values, it requires manual pre-calibration of the LiDAR unit [17], instead of using the parameters given by the manufacturer. This extra step involves the use of specific equipment, making this method difficult to deploy and test in practical situations.

Table 1 presents a feature summary comparison of the Camera-LiDAR calibration methods mentioned. From this we can compare characteristics such as integration, required target type, and sensing devices. After a quick analysis, we can identify the need of a method that works with different types of cameras and LiDARs, while maintaining the target-less property. Target-less methods offer the possibility to calibrate without the need of a special setup, or target. This feature allows users to reduce the time required to obtain the calibration parameters.

2.3 Camera-LiDAR Fusion

Having both sensors extrinsically calibrated, we can perform fusion at a point-to-pixel level. This enables both sensors to complement each other. The camera can integrate distance information for each of the 3D points projected on the image, while the LiDAR can extract color information to each of the points projected onto the camera field of view.

The work presented in [19] achieves fusion using distinct shots taken with a single camera pointing towards the LiDAR field of view. Having the camera intrinsics known and using photogrammetry techniques, they obtain the colored point cloud. However, this approach requires several pre-computations and is not capable of real-time performance. A different approach can be found in [13]. In this work, the authors integrate the intrinsic and extrinsic calibration over several phases. Nevertheless, they neither present the fusion method nor their findings on computation time.

2.4 Point Cloud Ground Classifier

The last part of our calibration framework involves the ground removal. This step is essential to ease the identification of obstacles on the road. Most work on this field ([20], [21], [22], and [23]) is focused on airborne applications, with LiDARs attached to a flying drone and pointing towards the ground. These methods can also be applied to both the rotating and solid-state LiDARs usually found in autonomous vehicles. However, these methods are not designed to be executed in real time.

Ground classification is often confused with road classification. This kind of research is focused on autonomous driving applications. The work by [24], and [25] present the current state-of-the-art developments. However, these require large labeled datasets, a frequently ignored limitation for its real application. For this reason, most of this work does not perform well when the resolution of the LiDAR sensor is changed, requiring a new dataset.

3. Theory and Implementation

In this section, we will explain each of the contributing parts of the framework, the theory behind each one, and its corresponding open-source implementation.

During the remaining sections we will employ the following coordinate frame conventions, which observe the ROS (Robot Operating Systems) [26] standard. In summary:

- All systems are right handed.
- LiDAR coordinates: x axis points forward, y axis extends to the left, and z faces upwards.
- Camera frames follow: z points forward, x extends to the right, and y faces downwards.

3.1 LiDAR-LiDAR Extrinsic Calibration

LiDAR sensors obtain the distance to an object by illuminating it with a pulsed laser, and measuring the time required to reflect and return. The 3D version of these sensors are offered by manufacturers as multi-layered, electro mechanical based, or solid-state devices. In all these cases, the device is equipped with one or more laser transceivers. Either by moving a single or multiple lasers, or pulsing the array constantly, the device generates a 3D point cloud, instead of a single 2D scan. This allows the device to generate 3D shapes with high resolution. Taking advantage of this feature, our LiDAR-LiDAR extrinsic calibration is based on a shape-based approach. Moreover, as originally shown in [27], the transformations obtained by this approach produce considerably lower error, while also performing up to three times faster than point-to-point based algorithms (i.e. ICP).

LiDAR-to-LiDAR calibration can be applied in a variety of sensor configurations. For example, new LiDAR technologies provide faster readings, reduced cost, non-mechanical solutions, velocity reading included on each point, etc. However, these sensors tend to have a narrower field of view, compared to their traditional mechanical counterparts. For this reason, extrinsic calibration is required when using several of these sensors. Another setting for multi-LiDAR application would be the integration of multiple LiDARs on bigger vehicles, such as SUVs, trucks, buses, etc., where a single rotating LiDAR would not be able to cover the entire field of view.

3.1.1 Theory

In order to extrinsically calibrate the sensors, both must have shared features as suggested by [7]. Our strategy employs the Normal Distributions (ND) method [28], [27] to match shapes, instead of a direct PDF (Probability Density Function) projection matching as presented in [7]. The ND algorithm tries to successively find the rigid transformation (Eq. (1)) between the b and a point cloud coordinate frames where R_{ab} is a rotation matrix, represented by the rotation on the three coordinate axes as shown in Eq. (2). The translation vector between these two spaces is represented by $\boldsymbol{t}$ as shown in Eq. (3).

$$p_a = R_{ab} * p_b + t_{ab} \tag{1}$$

$$R(\alpha,\beta,\gamma) = R(\alpha) \times R(\beta) \times R(\gamma) = \begin{pmatrix} cos\alpha & -sin\alpha & 0 \\ sin\alpha & cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} cos\beta & 0 & sin\beta \\ 0 & 1 & 0 \\ -sin\beta & 0 & cos\beta \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\gamma & -sin\gamma \\ 0 & sin\gamma & cos\gamma \end{pmatrix} \tag{2}$$

$$\boldsymbol{t} = (\, t_x \ \ t_y \ \ t_z \,) \tag{3}$$

To solve this problem, and assuming we have an initial estimate pose (x, y, z, raw, $pitch$, and yaw), the ND algorithm uses the Newton algorithm to maximize Eq. (4), when comparing the point clouds we wish to align, where w, p_i, and $\sum_i$ are the desired pose (matching parameter), the mean vector, and the covariance matrix respectively.

$$Fitness(\boldsymbol{X}, w) = \sum_{i}^{N-1} exp(\frac{-(x_i - p_i)^T \sum_i^{-1} (x - p_i)}{2}) \tag{4}$$

This calculation involves: 1) the partitioning of the point cloud P_a in smaller k regions, called Voxels. Voxels are quantized versions of a point cloud, obtained by dividing the space into cubic lattices. The points are assigned to the appropriate Voxel, by rounding the coordinate value of each point p_i; 2) The computation of the normal vectors $n_{k,i}$ for each of the M_k points inside the Voxels; 3) The probability distribution for the resulting normal vectors, which requires the calculation of the average p and covariance matrix Σ_k as shown in Eqs. (5) and (6), respectively.

$$\boldsymbol{p_k} = \frac{1}{M_k} \sum_{i=0}^{M_k - 1} x_{ki} \tag{5}$$

$$\boldsymbol{\Sigma_k} = \frac{1}{M_k} \sum_{i=0}^{M_k - 1} (x_{ki} - p_k)(x_{ki} - p_k)^T \tag{6}$$

Finally, the normal distribution for each Voxel is estimated as:

$$e(x)_k \approx e(\frac{(x - p_k)^t \sum_k^{-1} (x - p_k)}{2}) \tag{7}$$

3.1.2 Implementation

Our implementation follows the original design by [27], integrated in the PCL (Point Cloud Library) library [9]. The parameters we used are: 0.1 and 1.0 meters for the voxel sizes on the downscaling step, for the point clouds to be aligned; the termination value for the score as 0.01 (ϵ); the step size for the optimization as 0.1; and 400 as the maximum number of iterations before forcing the termination of the algorithm when the algorithm cannot converge. As defined in [27], the algorithm requires a rough estimated initial pose. In summary, the inputs required by our tool are:

- The point clouds generated by the LiDAR sensors to be aligned. These can be obtained directly from the sensor, in an online fashion, or from previously recorded data.
- An optional voxel size for both point clouds. A larger value will accelerate the process, but will decrease the accuracy due to a larger quantization.
- A rough estimation of the transformation between the sensors, in the form $(x, y, z, raw, pitch, yaw)$.

The LiDAR-to-LiDAR calibration tool is implemented as a ROS node. This approach allows us to add an abstraction layer between the sensors and the algorithm, enabling our node to support any kind of LiDAR sensor, as long as we can add its respective driver. Figure 2 shows a high-level diagram of the node.

The resulting transformation matrix is then registered

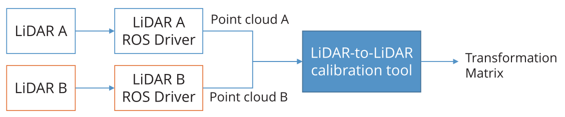


Fig. 2 High-level diagram of the LiDAR-to-LiDAR calibration tool.

in the ROS Transformations Tree (TF). The TF eases the conversion between the coordinate frames in a synchronized manner. In the specific case of LiDAR-to-LiDAR calibration, this transformation is static. This means that the transformation will not change over time, since both sensors are fixed to the vehicle's chassis.

3.2 Camera-LiDAR Extrinsic Calibration

We decided to implement our method as a semi-automatic calibration tool. The user is required to select the corresponding points between the image and point cloud, using a point-and-click approach. This method removes the necessity to setup a special room, or create particular markers or targets to identify the corresponding points. Moreover, this allows our method to work with not only RGB cameras, but also other types of projective cameras, such as infrared ones.

3.2.1 Theory

The Camera-LiDAR extrinsic calibration follows the same idea as the LiDAR-LiDAR extrinsic calibration. Find the relative transformation between both spaces with the help of shared features between spaces. However, in the case of cameras and LiDARs, these do not represent the data in the same number of dimensions. To find the relationship between them, we need to add an extra step. Knowing that LiDARs represent the points in an orthogonal 3D space using euclidean coordinates (x, y, z); and that cameras represent 2D points in a perspective space (u, v), we can relate these spaces through a linear transformation in the form:

$$\boldsymbol{p_{cam}} = \boldsymbol{P} * \boldsymbol{R} * \boldsymbol{t} * \boldsymbol{p_{lidar}} \tag{8}$$

where:

$\boldsymbol{p_{cam}}$	represents the final projected points in image space.
$\boldsymbol{P}$	is the camera projection matrix, also known as the camera intrinsics, or the intrinsics parameters.
$\boldsymbol{R}$	is the rotation matrix on 3D space as mentioned in Eq. (2).
$\boldsymbol{t}$	is the translation vector between the camera and the LiDAR, described on Eq. (3).
$\boldsymbol{p_{lidar}}$	is the LiDAR point cloud in 3D space.

Multiplying all of these, we derive the following matrix:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{9}$$

Equation (9) represents the complete relationship between the LiDAR and the camera spaces, where u and v represent the coordinates in pixel space, and x, y and z represent the 3D space in the LiDAR frame.

To obtain the transformation parameters m, we need to solve the generated system of equations. However, it is important to note that knowing in advance the intrinsic parameters of the setup helps to simplify this problem. Knowing the projection parameters, we would only need to find the Euler rotations (α, β, γ) and the translation vector $\boldsymbol{t}$. Since our objective is to obtain the extrinsic calibration we will assume the intrinsic matrix $\boldsymbol{P}$ is known. To obtain these parameters we followed the well-established intrinsic calibration methods integrated in OpenCV [29].

Having simplified the parameters calculation, we proceed to estimate the pose between the intrinsically calibrated camera to the LiDAR sensor. This problem is commonly known as Perspective-n-Point (PnP). To solve it we employed the Efficient PnP method [30]. This algorithm estimates the R and t matrices minimizing the re-projection error using the Gauss-Newton method, given the corresponding 2D-3D points (in our case, p_{cam} and p_{lidar}), and the camera intrinsics (P). Even if the method requires as low as five points to calculate the parameters, we ask the user to provide at least nine corresponding points, as shown in [30].

3.2.2 Implementation

As previously mentioned, to feed the point correspondences between the points in LiDAR space and camera space, we developed a simple, yet effective point-and-click user interface (UI) solution. This UI shows the image and the point cloud in parallel. The user can quickly identify the correspondences between image and the projected point cloud by clicking on the screen. This approach removes any specific setup or marker prerequisite, and enables our method to work virtually anywhere, as long as the user is able to identify features between the image and point cloud.

Similarly to the LiDAR-LiDAR calibration tool, the camera-LiDAR extrinsic calibration is a ROS node. It can connect to any camera and LiDAR supported by the system. Figure 3 shows the components of the calibration tool.

Figure 4 presents the user interface displayed to allow the selection of the corresponding points between 3D and 2D space.

Finally, the node is designed to work with both a real sensor or use previously recorded data. This feature allows the selection of points located at different instants in time, easing the selection of better matching features between both sensors.

3.3 Image-Cloud Fusion

With the Camera-LiDAR extrinsic calibration complete, we can fuse the data from both sensors. The range information from the LiDAR can be projected to the 2D space. Similarly, the color information contained in the image can be back-projected to the LiDAR space. Figure 5 illustrates this concept. The point cloud projected to the image is displayed using a color map representing the distance between the LiDAR and the object being observed. The red color represents a closer distance, while a blue color represents a further one. This kind of point cloud representation is specially useful when detecting and tracking objects [4]–[6]. For this reason, we decided to present an open real-time solution to this problem. Our approach uses a hashing function to temporarily store the 3D-2D correspondences. This enables us to back-project the color information in constant time $O(1)$, at the cost of $O(n)$ storage space.

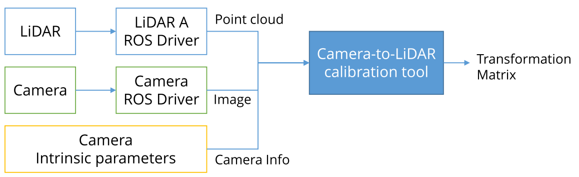


Fig. 3 High-level diagram of the camera-to-LiDAR calibration tool.

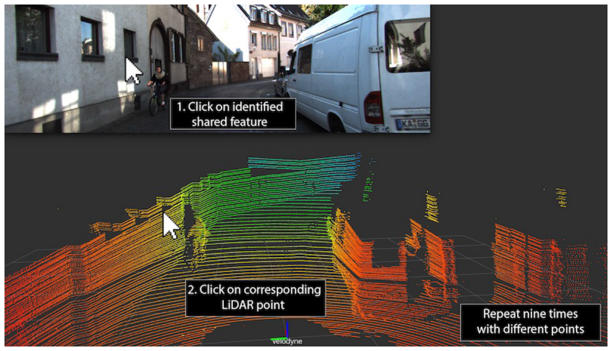

1. Click on identified shared feature
2. Click on corresponding LiDAR point
Repeat nine times with different points
velodyne

Fig. 4 UI presented to the user to ease the selection of corresponding points.

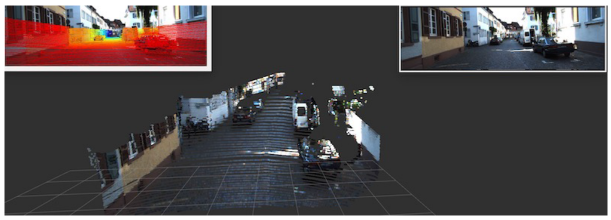


Fig. 5 Result of the real time Image-Cloud fusion using a Velodyne HDL-64 and a wide-angle camera.

3.3.1 Theory

With the help of Eq. (8) and having known the matrices P, R and t from the calibration steps, we can obtain the correspondence between the LiDAR points and the image pixels. The distance for each point is calculated on the ground plane as $\sqrt{x^2 + y^2}$.

In order to back-project the color information from the image to the LiDAR's point cloud, we use an unordered map data structure. We define the unordered map as a list of ordered pairs $<(u, v), (x, y, z)>$, that use the point coordinates in both spaces as the hashing function for indexing. Defining the map this way, allows us to find the correspondences in constant time $O(1)$.

The implementation of the node was done following

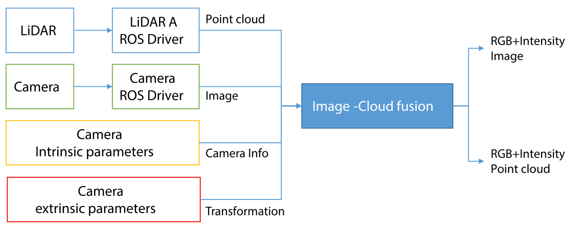


Fig. 6 High-level diagram of the image-cloud fusion.

the ROS messaging standard. An outline of the node is shown in Fig. 6.

3.4 Ground Classification

The final component of the framework presented in this work is a ground classifier for point cloud data. We call it the Ray Ground Filter. It is a binary classifier (*Ground* or *No-Ground*) for the points in the point cloud. As previously mentioned in Sects. 1 and 2, this step is important to ease the implementation of higher level perception methods, since it determines the points considered as an obstacle for its classification and tracking.

3.4.1 Theory

The first phase of the Ray Ground Filter algorithm involves converting the point cloud from Euclidean space to an alternative quantized polar space. The new space is partitioned into a fixed number of radial dividers or rays. The steps involved in this conversion are:

1. Remove the points which are above the height of the LiDAR h_{lidar}. It is important to remove objects above the vehicle that are not considered obstacles (i.e. bridges, signals, etc.).

$$P_e = \{(x, y, z) \in \mathbb{R} \times \mathbb{R} \times \mathbb{R} \,|\, z < h_{lidar}\} \tag{10}$$

2. Convert the points from 3D Euclidean space (x, y, z) to 2D Polar space (r, θ), while also storing the point's height (z) and a reference to the original point, where $\theta \in [0, 2\pi)$.

$$\begin{aligned} P_p &= \{(\theta, r, z)\}\text{, such that} \\ \theta &= atan(y, x) \\ r &= \sqrt{x^2 + y^2} \\ z &= z \end{aligned} \tag{11}$$

3. Quantize θ into N radial dividers.

$$N = ceil(2\pi/\alpha * i) \,|\, \alpha \in [0, 2\pi] \tag{12}$$

where α is a value decided by the user. We use a 100th of a degree (1.74 e-04 rad).

4. Create a list for each of the N radial dividers.

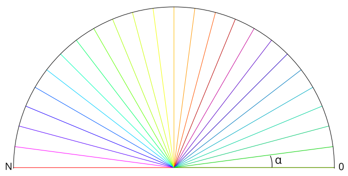


Fig. 7 Diagram of the radial dividers (rays) used in the ray ground algorithm. Showing only 0 to π for simplicity.

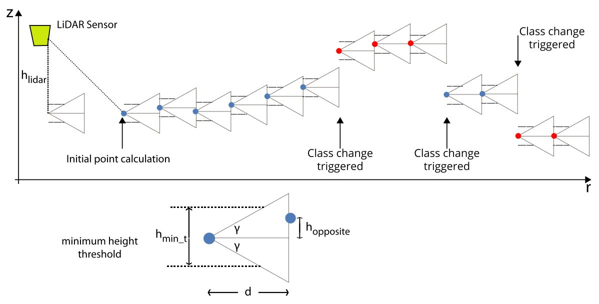


Fig. 8 Ray ground filter classification diagram following triangle geometries inside a ray. The points shown in blue color were classified as ground, while the red ones as obstacles.

5. Assign the points to the quantized angle and add it to the list. Figure 7 shows each the radial dividers generated and differentiated by color.

$$\forall p \in P_p,\ n = floor(p_\theta/\alpha) \tag{13}$$

6. Order all the points inside of each of the rays (unordered lists).

At this point we achieve a new point cloud formed by N lists. Each one of these lists contain points of a new type $(x, y, z, i, r, \theta, n)$, where (x, y, z) are the original coordinates; i represents the intensity of the laser reflectivity; θ is the angle in polar coordinates; and n is the index of the quantized radial divider or ray to which this points belong; and k is the number of points in each of the ordered rays.

With all the points organized and ordered, we proceed to classify them into two classes *Ground* and *No-Ground* using a triangle geometry, as shown in Fig. 8. The adjacent side of the triangle is defined as the distance (d_i) between two consecutive points in the ray.

$$d_i = r_i - r_{i-1}, i \in (0, k] \tag{14}$$

The first point in the ray is classified using the same geometry shape. However, the initial point is a virtual one located below the LiDAR sensor h_{lidar}. The opposite side is calculated according to a given angle threshold (γ), while d and γ define the maximum height difference to which the next point in the ray might be $(h_{opposite})$.

If the next point falls inside the triangle geometry, the point is considered to be of the same class as the previous point.

$$\begin{aligned} h_{i,opposite} &= tan(\gamma) * d_i \\ \text{if } h_{i,opposite} &< h_{min}, \\ \text{then } h_{i,opposite} &= h_{min} \end{aligned} \tag{15}$$

Otherwise, it triggers a change of class. However, the points located closer to the sensor tend to be adjacent due to the laser arrangement, triggering a class change on uneven terrain. For this reason, if $h_{opposite}$ is less than a given value h_{min_t}, the change is not generated (Fig. 8).

If a change of class is detected when the previous class was *No-Ground*, then the point needs to be re-classified to ensure it is *Ground*. To achieve this, we re-calculate the geometry of the triangle using the distance between the last point classified as *Ground* and the current one.

3.4.2 Implementation

The classifier was implemented as a ROS node. The node can receive a point cloud from any LiDAR sensor. It requires as inputs:

- A point cloud of type (x, y, z) or (x, y, z, i).
- The height of the LiDAR sensor h_{lidar}.
- The quantization angle for the radial divisions α.
- The angle γ which defines the maximum height of a consecutive point.
- The minimum height threshold between points h_{min}.

To accelerate the processing, the conversion from (x, y, z) to (θ, r, z), and the quantization is performed in parallel. The resulting radial dividers are stored in N vectors. If the host architecture supports OpenMP, the node will perform conversion and sorting on each one of the CPU cores, reducing the load and accelerating execution.

Figure 9 presents the high-level architecture design of the classifier. Finally, the result of ground classification on the KITTI dataset on a Velodyne HDL-64 can be seen in Fig. 10. Blue colored points denote the ones classified as ground, while the red points indicate those belonging to an obstacle.

Due to nature of the algorithm, it can virtually work with any type and/or number of LiDARs. It also does not require a large dataset for a training phase. Moreover, the algorithm was implemented completely using the CPU, allowing it to work on low-power and embedded systems without a GPU.

4. Evaluation

4.1 LiDAR-LiDAR Extrinsic Calibration

LiDAR to LiDAR evaluation using well-established resources is difficult. Currently there are no public datasets including the assessment of multiple LiDARs. For this reason, we decided to prepare two vehicles with different settings, obtain the ground truth, and evaluate our method.

The first setup is built on top of a Toyota Prius sedan. We mounted a rotating LiDAR (Velodyne HDL-64, with a 360 degrees field of view), and four low-cost smaller Micro-Electro-Mechanical-based Scanning LiDARs (MEMS). Each MEMS LiDAR had 140 degrees of horizontal field of view. To obtain the position and Euler rotations, we used a chessboard pattern as a guide to manually identify points between each of the LiDAR frames using the changes in intensity. Having identified the corresponding points, we calculated the rigid-transformation matrix between the rotating LiDAR, and each of the solid-state ones using the Singular-value decomposition (SVD) method. The manually obtained measurements are shown in Fig. 11.

The second setup consisted of five rotating Velodyne HDL-16 LiDARs mounted on a Toyota Alphard minivan. The ground truth was calculated in the same way as in the Prius setup. Figure 12 shows the setup and the obtained measurements.

Using our LiDAR-to-LiDAR extrinsic calibration tool, we collected a database of 20 different experiments. The average resulting parameters are shown in Table 2 and Table 3, for the Prius and Alphard setups respectively. Figure 13 displays the average accuracy and standard deviation for the calculations compared to the manually obtained points.

Finally, the extrinsic calibration obtained using our

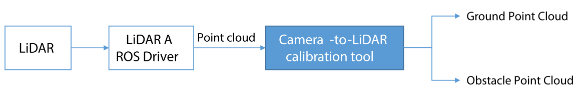


Fig. 9 High-level diagram of the ray ground classifier component.

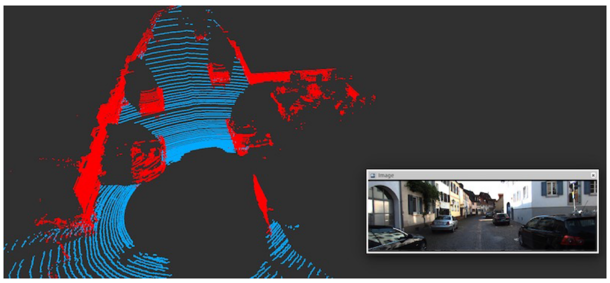


Fig. 10 Ray ground filter on KITTI dataset. Image shown only for reference.

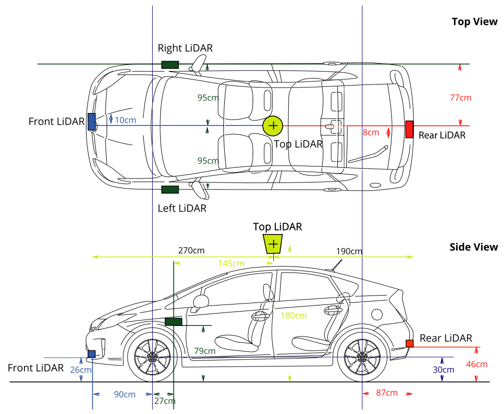


Fig. 11 Manually obtained measurements used as ground truth to evaluate our method. Prius setup, Velodyne HDL-64 and four narrow MEMS LiDARS.

Table 2 Prius Setup. LiDAR-LiDAR extrinsic calibration quantitative results between each of the narrow field of view LiDARs, and a rotating LiDAR. All units are in meters. Reporting absolute error.

	Rotating to Front LiDAR			Rotating to Left LiDAR			Rotating to Right LiDAR			Rotating to Rear LiDAR		
Parameter	**Ground Truth**	**Our method**	**Error**	**Ground Truth**	**Our method**	**Error**	**Ground Truth**	**Our method**	**Error**	**Ground Truth**	**Our method**	**Error**
X [meters]	2.70	2.67724	0.0228	1.450	1.4387	0.0113	1.450	1.44314	0.0069	-1.90	-1.8782	0.0218
Y [meters]	-0.10	-0.10886	0.0089	0.950	0.9574	0.0074	-0.950	-0.9678	0.0178	0.08	0.0737	0.0063
Z [meters]	-1.54	-1.56121	0.0212	-1.100	-1.0274	0.0726	-1.100	-1.1095	0.0095	-1.34	-1.3665	0.0265
α [rads]	1.57	1.5708	0.0008	3.140	3.096	0.0440	3.140	3.13085	0.0091	1.57	1.5708	0.0008
β [rads]	0.00	0.0074	0.0074	0.000	-0.0537	0.0537	-3.140	-3.1368	0.0032	3.13	3.1153	0.0147
γ [rads]	1.57	1.5708	0.0008	1.570	1.5243	0.0457	-1.570	-1.5773	0.0073	-1.57	-1.5708	0.0008

Table 3 Alphard Setup. LiDAR-LiDAR extrinsic calibration quantitative results between each the top LiDAR, and a five rotating LiDAR covering blind spots. All units are in meters. Reporting absolute error.

	Top to Front LiDAR			Top to Right LiDAR			Top to Left LiDAR			Top to Rear LiDAR		
Parameter	**Ground Truth**	**Our method**	**Error**	**Ground Truth**	**Our method**	**Error**	**Ground Truth**	**Our method**	**Error**	**Ground Truth**	**Our method**	**Error**
X [meters]	2.75	2.75793	0.00793	-1.70	-1.7198	0.0198	-1.74	-1.74005	0.00005	-2.14.	-2.1429	0.0029
Y [meters]	0.05	0.04227	0.01227	-0.95	-0.95033	0.00033	0.93	0.9391	0.0091	-0.05	-0.0453	0.0153
Z [meters]	-1.31	-1.31587	0.00587	-1.3500	-1.39294	0.04294	-1.45	-1.46752	0.016752	-1.23.	-1.23756	0.00756
α [rads]	0.00	0.00584	0.00584	0.03	0.0394	0.0094	0.0	-0.0273	0.0273	0.	-0.0267	0.0267
β [rads]	0.05	0.06284	0.01284	0.00	-0.0032	0.0032	0.0	-0.02206	0.02206	0.	0.0206	0.0206
γ [rads]	0.00	0.01505	0.01505	-1.57	-1.5782	0.0082	1.58	1.58036	0.00036	3.1415	3.14124	0.00034

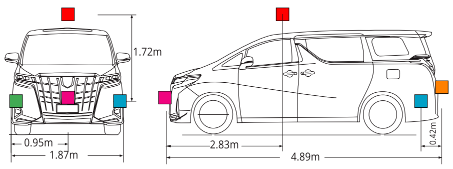


Fig. 12 Manually obtained measurements used as ground truth to evaluate our method. Alphard setup, five Velodyne VLP-16.

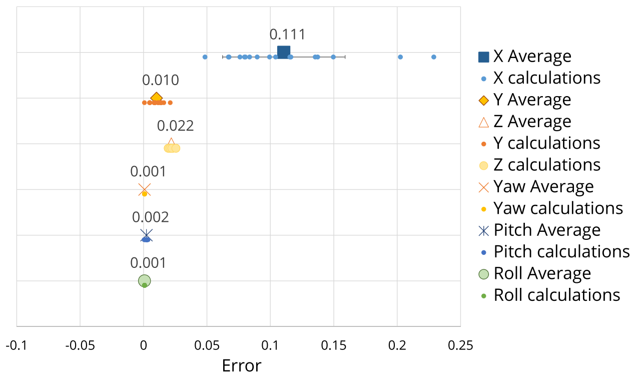


Fig. 13 Prius Setup. Average absolute error for 20 repeated calculations of the extrinsic parameters between LiDARs. X, Y, Z error is reported in meters, while Yaw, Pitch, Roll in radians.

method can be seen in Figs. 14 and 15 for the Prius and Alphard setups respectively. For the Prius Setup, white points represent the point cloud belonging to the rotating LiDAR, while the colored ones belong to the other low-resolution MEMS LiDARs. In a similar way, for the Alphard setup, white points represent the point cloud generated by the top rotating lidar, while the colored ones belong to the other rotating sensors located to the front, left, right, and rear of the vehicle.

4.2 Camera-LiDAR Extrinsic Calibration

To evaluate our method, we took as reference the regarded KITTI dataset [18]. The authors of the dataset calibrated

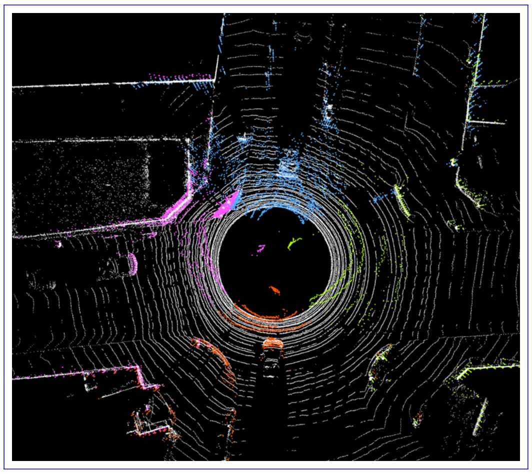


Fig. 14 Prius Setup. Result of extrinsic calibration between rotating LiDAR (shown in white), and four MEMS narrow view LiDAR sensors (displayed in blue, green, pink, and red).

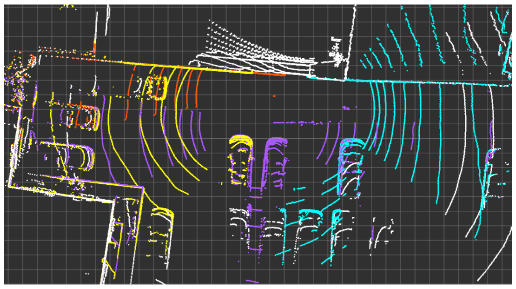


Fig. 15 Alphard Setup, result of extrinsic calibration of five rotating LiDARs (top: white; front: cyan; right: purple; left: orange; and rear: yellow).

their sensors using the method described in [11]. This approach is used in other work as standard baseline.

To calibrate the LiDAR and the camera we randomly selected a frame, as shown in Fig. 4. Then, we proceeded to point and click the correspondent points in the image and

Table 4 Camera-LiDAR extrinsic calibration results comparison. Units in meters and radians.

Parameter	KITTI Ground Truth	Geiger, et.al.[11]	Our method	Absolute Error
X [meters]	0.27	0.2717	0.3017	0.0317
Y [meters]	-0.06	-0.076	-0.089	0.029
Z [meters]	-0.08	-0.04	-0.0937	0.0137
Yaw [rads]	1.57	-1.5632	-1.5625	0.039
Pitch [rads]	0.0	0.0006	0.0012	0.0356
Roll [rads]	-1.57	-1.5559	-1.5622	0.358

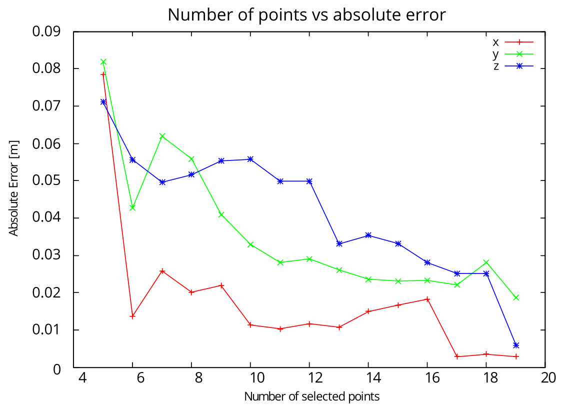


Fig. 16 Translation absolute error compared with number of selected points on the Alphard Setup.

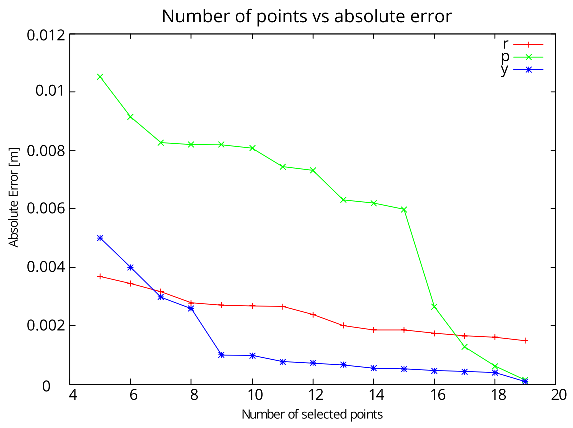


Fig. 17 Rotation absolute error compared with number of selected points on the Alphard Setup.

the point cloud. Table 4 shows our results compared to the measurements reported by [18]. However, it is important to note that our approach lets the user to select data from as many frames as needed over the time. This helps to ease the matching process over the image field, and at different ranges.

4.3 Image-Cloud Fusion

Due to the nature of the back-projection calculation, the results of Image-Cloud fusion depend on the accuracy of both the camera intrinsics, and extrinsics parameters. For this reason, to evaluate this section of the framework, we decided to carry out instead a performance evaluation. We measured the average performance of our tool on four different camera and LiDAR setups. Table 5 shows the arrangements used for this purpose. Figure 18 presents qualitative fusion results for each arrangement. All these tests were performed on a desktop computer running Ubuntu 16.04, ROS Kinetic, with an Intel Core i7-6700K CPU with 4 cores, and 16 GB of RAM. The node was compiled with OpenMP to distribute the execution on the multi-core system.

Table 5 Fusion measurements for different sensors and cameras on an Intel Core i7-6700K CPU with 4 cores. A. Baumer VLG-22C@ 20 Hz; B. PointGrey Grasshopper3@15 Hz; C. PointGrey Grasshopper3@15 Hz; D. PointGrey Flea2 @ 15 Hz

Camera	Image Resolution	Velodyne LiDAR	Point cloud size	Execution (ms)
A	1288x960	HDL-64	~ 120,000	116
B	800x600	HDL-32	~ 70,000	78
C	1384x1036	VLP-16	~ 25,000	50
D	1384x1032	HDL-64	~ 120,000	128

Table 6 Classification performance of the ray ground classifier on the point clouds by sensor type.

Measurement	HDL-64	HDL-32	VLP-16
Accuracy	0.8247	0.8429	0.8461
TP	0.8429	0.7737	0.6954
FP	0.1952	0.1025	0.0653
TN	0.8047	0.8974	0.9346
FN	0.1570	0.2262	0.3045
Precision	0.8258	0.8560	0.8619
Execution time (ms)	55	20	11

4.4 Ray Ground Classifier

To evaluate the classification accuracy, we decided to manually label 12 point clouds. We chose four random point clouds for each of the most commonly used Velodyne sensors: HDL-64, HDL-32 and VLP-16. Each scan was taken from previously recorded log data from various urban settings. For the Velodyne HDL-64 case, we used the point clouds provided by the KITTI dataset. The average results for the point clouds classified can be seen in Table 6. In this table we also included the average execution time for each sensor model. The execution time was measured on a desktop computer running on an Intel Core i7-6700K CPU with 4 cores, and 16 GB of RAM with OpenMP enabled.

Finally, qualitative results on three different sensors can be seen on Fig. 19. The last sub figure shows the Ray Ground classifier working on four merged Velodyne VLP-16 LiDARs. These were previously calibrated using our method presented in Sect. 3.1.2.

5. Discussion

5.1 LiDAR-LiDAR Extrinsic Calibration

Like any other approximation algorithm, the ND algorithm may fail to converge under the maximum number of iterations. This is due to the absence of differentiable features, leading to the cost-function to become insensitive to the rotational parameters. In Fig. 13, we can observe that when providing a feature rich environment, the ND algorithm can

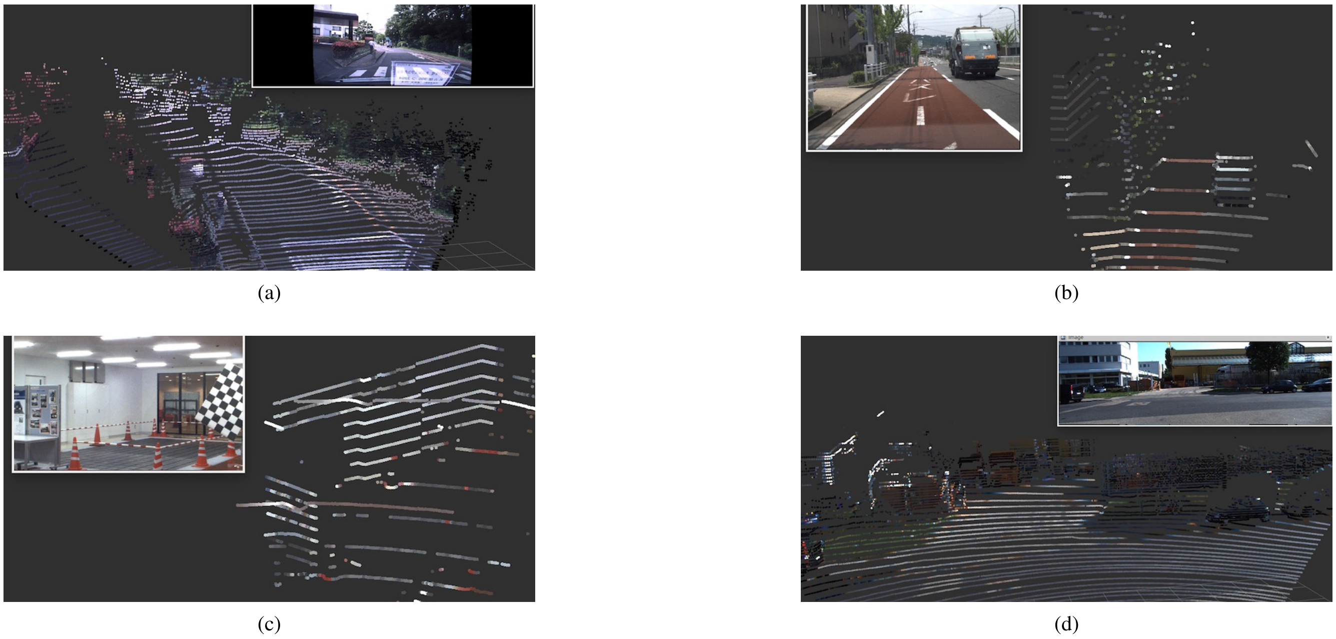


Fig. 18 Fusion results on different setups a) BaumerVLG-22 and HDL-64; b) Grasshopper3 and HDL-32; c) Grasshopper and VLP-16; d) Flea2 and HDL-64 (KITTI).

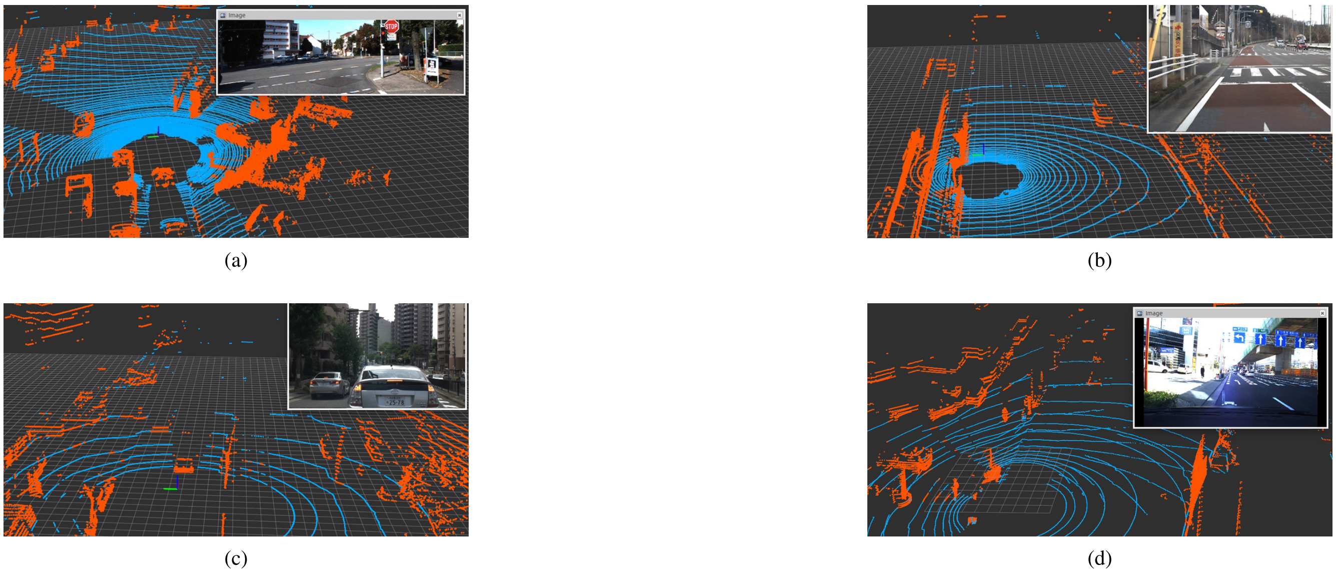


Fig. 19 Ground classification, using our ray ground algorithm on a) Velodyne HDL-64 (KITTI); b) Velodyne HDL-32; c) Velodyne VLP-16; d) four Velodyne VLP-16 calibrated. The points belonging to the road are shown in blue, while obstacles are painted in red.

obtain highly accurate results. The error on the x-axis, even if it is considerably low, presents a higher value than the rest of the parameters. While analyzing this difference, we found out that this was caused by an inaccurate internal laser calibration from the manufacturer. Figure 2 shows that the points hitting the wall (the x-axis), display an uneven wall surface.

During our indoor and outdoor experiments, we realized that selecting a complex and/or asymmetric environment leads to a faster convergence. Since our tool shows the calibration result after completing each set of iterations in real-time, the user can quickly identify if the selected setting is appropriate for calibration. Thus, from our experiments we conclude that it is important to have large objects (i.e. walls or other vehicles) in the shared field of view of the LiDARs being calibrated. It is important to note, that our method requires features to be shared between the sensor views, otherwise calibration will not be possible.

When calibrating more than two LiDAR sensors that all share an overlapping field of view, the calibration must be performed N-1 times, where N is the number of sensors. In situations when multiple LiDARs only share a field of view with some of the other sensors, the obtained solution might not converge to a local optima. This is because the NDT algorithm is designed to match only two point clouds at a time. To overcome this, one approach would be to complete calibration N-1 times, and afterwards repeat calibration N-1 times using a different order. If the same solution is obtained, the current solution is very likely to be a global optima. Nevertheless, this cannot be confirmed due to the lack of extra information from other sensors. If a different solution is obtained, then an average between the solutions might be used. To further improve the optimization of the calibration parameters, a bundle adjustment-like approach could be performed. This would allow to tune the individual calibration results to maximize global consistency.

5.2 Camera-LiDAR Extrinsic Calibration

As mentioned in Sect. 3.2, our method provides an easy and practical way to quickly obtain the extrinsic calibration between the Camera and LiDAR. It does not require the preparation of a specific setting, or marker of any kind. For this reason, it allows calibration of the sensors while directly connected to them, or the use of previously recorded data. Since our method depends completely on the points fed to the algorithm, its accuracy highly depends on the ability of the user to find proper correspondences. During our experiments we found it is easier to observe the shared features between sensors when there are objects nearby. This is due to the nature of the multi-layered laser and the perspective camera model. After several experiments we defined the following guidelines to ease the calibration process. Users familiar with these, were able to obtain the parameters in as short as one minute.

1) Capture a single scene with different objects scattered around the field of view. 2) It is easier to identify objects' corners, points hitting lowest and highest points of an identified object in the point cloud. 3) Since our method allows the selection of points regardless time of the the frame, using recorded log data eases the selection of these points while the car is running in a urban scenario. Taking advantage of this attribute is highly recommended in cases where a static scenario does not contain enough identifiable features. 4) As shown in Figs. 16 and 17, selecting more points will obtain a reduced error on the translation parameters. However, if the selected features are not easily identifiable, selecting more points will increase the final absolute error.

Finally, an additional advantage of our method is that it also allows the calibration regardless of the image format. Figure 20 shows a successful extrinsic calibration between a thermal vision camera and a Velodyne HDL-64.

5.3 Image-Cloud Fusion

The image-cloud fusion node is an example of an application of a successful integration of sensor fusion. It complements and integrates distance and color data between the sensors, providing extra information to other perception applications. We consider that this tool will help other researchers in the development of real-time perception systems. As previously mentioned, recent developments based on deep learning require the post-fusion data to be fed to the network [4]–[6]. All these techniques perform the data fusion off-line, forwarding it at a later phase. Our method provides a real-time solution to this problem in many applications. It is important to note that the performance might be reduced when using higher resolution images and LiDAR sensors.

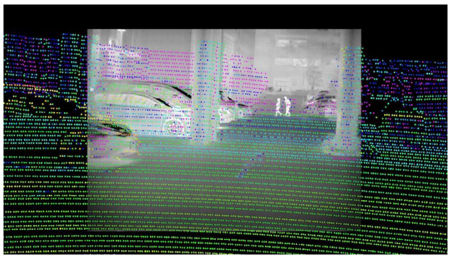


Fig. 20 Successful calibration of a thermal vision camera (FLIR ADK), and a Velodyne HDL-64.

5.4 Ray Ground Classifier

The Ray Ground Classifier presents an elegant and effective way to classify the ground on complex point clouds. Being a shape and rule based method, is possible to use it on different sensor types and setups. It requires few or none of its parameters to be tuned for successful application. Moreover, being implemented completely on the CPU, it can be ported to other embedded architectures. Our evaluation shows that it achieves a high accuracy across different sensor types as indicated in Table 6. Due to its nature, it does not require specific measurements, such as rings numbers or specific intensity ranges, making our approach sensor independent. Figure 19 presents qualitative results of the ground classification on four Velodyne VLP-16 LiDAR sensors, previously calibrated with the approach presented in Sect. 4.1.

During our experiments and evaluation, we found out that to obtain better classification results it is important to set correctly the height of the LiDAR (h_{lidar}), and the angle threshold of the maximum slope (γ). The first one is usually known and when it is not, it can be estimated by averaging the first point of each of the rays. The latter can be quickly estimated using the UI provided by ROS, while analyzing the height values in the ground of the point cloud. Finally, we also carried out experiments on steep roads. On these we found out the classifier to be reliable, when changing the slope parameter. However, we also found that keeping a high slope value on regular roads might cause misdetections.

6. Conclusion

We described an open-source multi-sensor fusion toolbox for autonomous vehicles. It is composed of a LiDAR-to-LiDAR extrinsic calibration algorithm, a Camera-LiDAR extrinsic calibration method, Multi-LiDAR fusion, Camera-LiDAR fusion, and a ground classification method.

Our LiDAR-to-LiDAR calibration algorithm successfully adapts a state-of-the-art matching algorithm. Through several experiments, we showed that provides calibration to the centimeter-level accuracy, without requiring a special setup. We also shared recommendations and limitations on Sect. 5.1 to ease the use of our calibration tool.

We introduced an easy-to-use Camera-to-LiDAR extrinsic calibration method. It applies the latest developments on camera pose estimation in computer vision. Instead of requiring the preparation of an specific setup, or the printing of predefined markers, a UI is provided, so the user feeds the corresponding points between the camera and LiDAR space to our algorithm. To accelerate this task, we also shared the guidelines we defined through experimentation in Sect. 3.2. Our method achieved error as low as one centimeter, comparable to other state-of-the-art developments in the field. It is important to note that our method will fail to obtain an accurate calibration if the intrinsic parameters are incorrect. The application of our method becomes challenging when using low resolution LiDARs in combination with telephoto lenses. Users can have a hard time to find shared features between images and point clouds. In these cases, targets might be inserted manually. We found that cones, or highly reflective materials are easy to identify between images and point clouds.

The third part of our framework, presented a real-time image and point cloud fusion. The tool integrates the range data to the image space. It also allows point cloud back-projecting the color information to the 3D space. To the best of our knowledge, it is the first real-time open-source available as the time of writing.

The final part of our framework is an accurate geometric and rule-based ground classifier. Thanks to the nature of the algorithm, and the low number of configurable parameters, it can be used with virtually any kind of LiDAR sensor, including data composed by several LiDARs. Moreover, our method performed on average as fast as 55 ms for the high-resolution Velodyne HDL-64. It averaged 20 and 11 ms on the HDL-32 and VLP-16 sensors respectively, achieving real-time performance on the three tested sensors.

Our multi-sensor fusion toolbox is integrated in the autonomous driving framework known as Autoware [31]. The source code can be downloaded from the following URL ⟨https://gitlab.com/autowarefoundation/autoware.ai⟩. Future work on our toolbox will involve the full automation of the calibration methods. For instance, employing recent developments on the Structure from Motion field. We also plan to implement the fusion and classification algorithms on GPUs to further accelerate its execution.

References

[1] D.J. Fagnant and K. Kockelman, “Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations,” Transportation Research Part A: Policy and Practice, vol.77, pp.167–181, 2015.

[2] U. Ozguner, C. Stiller, and K. Redmill, “Systems for safety and autonomous behavior in cars: The DARPA grand challenge experience,” Proc. IEEE, vol.95, no.2, pp.397–412, 2007.

[3] A. Petrovskaya and S. Thrun, “Model based vehicle tracking for autonomous driving in urban environments,” Proc. Robotics: Science and Systems IV, Zurich, Switzerland, vol.34, 2008.

[4] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander, “Joint 3D proposal generation and object detection from view aggregation,” arXiv preprint arXiv:1712.02294, 2017.

[5] M. Simon, S. Milz, K. Amende, and H.M. Gross, “Complex-YOLO: Real-time 3D object detection on point clouds,” arXiv preprint arXiv:1803.06199, 2018.

[6] C.R. Qi, W. Liu, C. Wu, H. Su, and L.J. Guibas, “Frustum PointNets for 3D object detection from RGB-D data,” arXiv preprint arXiv:1711.08488, 2017.

[7] A. Alempijevic, S. Kodagoda, J. Underwood, S. Kumar, and G. Dissanayake, “Mutual information based sensor registration and calibration,” IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2006.

[8] M. Pereira, D. Silva, V. Santos, and P. Dias, “Self calibration of multiple LiDARs and cameras on autonomous vehicles,” Robot. Auton. Syst., vol.83, pp.326–337, 2016.

[9] R.B. Rusu and S. Cousins, “3D is here: point cloud library (PCL),” Robotics and automation (ICRA), 2011 IEEE International Conference on, pp.1–4, IEEE, 2011.

[10] P.J. Besl and N.D. McKay, “A method for registration of 3-D shapes,” IEEE Trans. Pattern Anal. Mach. Intell., vol.14, no.2, pp.239–256, Feb. 1992.

[11] A. Geiger, F. Moosmann, Ö. Car, and B. Schuster, “Automatic camera and range sensor calibration using a single shot,” Robotics and Automation (ICRA), 2012 IEEE International Conference on, pp.3936–3943, IEEE, 2012.

[12] O. Naroditsky, A. Patterson, and K. Daniilidis, “Automatic alignment of a camera with a line scan LiDAR system,” Robotics and Automation (ICRA), 2011 IEEE International Conference on, pp.3429–3434, IEEE, 2011.

[13] M. Vel’as, M. Španěl, Z. Materna, and A. Herout, “Calibration of RGB camera with velodyne LiDAR,” 2014.

[14] K. Kwak, D.F. Huber, H. Badino, and T. Kanade, “Extrinsic calibration of a single line scanning LiDAR and a camera,” Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pp.3283–3289, IEEE, 2011.

[15] W. Wang, K. Sakurada, and N. Kawaguchi, “Reflectance intensity assisted automatic and accurate extrinsic calibration of 3D LiDAR and panoramic camera using a printed chessboard,” Remote Sensing, vol.9, no.8, p.851, 2017.

[16] G. Pandey, J.R. McBride, S. Savarese, and R.M. Eustice, “Automatic targetless extrinsic calibration of a 3D LiDAR and camera by maximizing mutual information,” AAAI, 2012.

[17] J. Levinson and S. Thrun, “Unsupervised calibration for multi-beam lasers,” Experimental Robotics, pp.179–193, Springer, 2014.

[18] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” Conference on Computer Vision and Pattern Recognition (CVPR), 2012.

[19] A. Abdelhafiz, B. Riedel, and W. Niemeier, “Towards a 3D true colored space by the fusion of laser scanner point cloud and digital photos,” Proc. ISPRS Working Group V/4 Workshop (3D-ARCH), Citeseer, 2005.

[20] K. Zhang, S.C. Chen, D. Whitman, M.L. Shyu, J. Yan, and C. Zhang, “A progressive morphological filter for removing nonground measurements from airborne LiDAR data,” IEEE Trans. Geosci. Remote Sens., vol.41, no.4, pp.872–882, 2003.

[21] X. Meng, L. Wang, J.L. Silván-Cárdenas, and N. Currit, “A multidirectional ground filtering algorithm for Airborne LiDAR,” ISPRS

J. Photogrammetry and Remote Sensing, vol.64, no.1, pp.117–124, 2009.

[22] F. Pirotti, A. Guarnieri, and A. Vettore, "Ground filtering and vegetation mapping using multi-return terrestrial laser scanning," ISPRS J. Photogrammetry and Remote Sensing, vol.76, pp.56–63, 2013.

[23] D. Maturana and S. Scherer, "3D convolutional neural networks for landing zone detection from LiDAR," Robotics and Automation (ICRA), 2015 IEEE International Conference on, pp.3471–3478, IEEE, 2015.

[24] Z. Chen and Z. Chen, "RBNet: A deep neural network for unified road and road boundary detection," International Conference on Neural Information Processing, pp.677–687, Springer, 2017.

[25] X. Han, J. Lu, C. Zhao, S. You, and H. Li, "Semisupervised and weakly supervised road detection based on generative adversarial networks," IEEE Signal Process. Lett., vol.25, no.4, pp.551–555, 2018.

[26] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng, "ROS: An open-source robot operating system," ICRA Workshop on Open Source Software, p.5, Kobe, Japan, 2009.

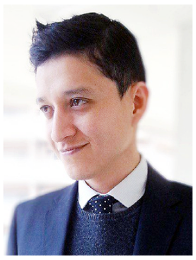
[27] M. Magnusson, A. Lilienthal, and T. Duckett, "Scan registration for autonomous mining vehicles using 3D-NDT," J. Field Robotics, vol.24, no.10, pp.803–827, 2007.

[28] E. Takeuchi and T. Tsubouchi, "A 3-D scan matching using improved 3-D normal distributions transform for mobile robotic mapping," Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on, pp.3068–3073, IEEE, 2006.

[29] G. Bradski, "The OpenCV library," Dr. Dobb's Journal of Software Tools, 2000.

[30] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: Efficient perspective-n-point camera pose estimation," Int. J. Comput. Vis., vol.81, no.2, pp.155–166, 2009.

[31] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," IEEE Micro, vol.35, no.6, pp.60–68, 2015.



Abraham Monrroy Cano received his B.E from the National University of Mexico and his M.S. degree from Nagoya University in 2008 and 2016, respectively. He is currently a Ph.D. student in the Graduate School of Informatics, Nagoya University. His research interests include deep learning, image processing, and autonomous vehicles.

Eijiro Takeuchi received his Bachelor, Masters and PhD degrees from the Intelligent Robot Laboratory, University of Tsukuba, Japan. From 2008 to 2014, he worked at Tohoku University, Japan, as Assistant Professor. Since 2014, he moved to Nagoya University, Japan, where he is currently an Associate Professor at the Graduate School of Informatics. His main interest is localization, mapping in Robotics and autonomous driving.

Shinpei Kato received the BS, MS, and PhD degrees from Keio University in 2004, 2006, and 2008, respectively. He is an associate professor in the School of Information Science, The University of Tokyo. He was also with the University of Tokyo, Carnegie Mellon University, and the University of California, Santa Cruz, from 2009 to 2012. His research interests include operating systems, real-time systems, and parallel and distributed systems.

Masato Edahiro received the PhD degree from Princeton University in 1999. He is a professor in the School of Informatics, Nagoya University. He was also at NEC Corporation from 1985 to 2010. His research interests include graph and network algorithms, and software for multi- and many-core processors.