3D Point Cloud Processing and Learning for Autonomous Driving

Chen, Siheng; Liu, Baoan; Feng, Chen; Vallespi-Gonzalez, Carlos; Wellington, Carl

TR2020-066 June 03, 2020

Abstract

We present a review of 3D point cloud processing and learning for autonomous driving. As one of the most important sensors in autonomous vehicles, light detection and ranging (LiDAR) sensors collect 3D point clouds that precisely record the external surfaces of objects and scenes. The tools for 3D point cloud processing and learning are critical to the map creation, localization, and perception modules in an autonomous vehicle. While much attention has been paid to data collected from cameras, such as images and videos, an increasing number of researchers have recognized the importance and significance of LiDAR in autonomous driving and have proposed processing and learning algorithms to exploit 3D point clouds. We review the recent progress in this research area and summarize what has been tried and what is needed for practical and safe autonomous vehicles. We also offer perspectives on open issues that are needed to be solved in the future.

IEEE Signal Processing Magazine

3D Point Cloud Processing and Learning for Autonomous Driving

Siheng Chen, Baoan Liu, Chen Feng, Carlos Vallespi-Gonzalez, Carl Wellington

Abstract

We present a review of 3D point cloud processing and learning for autonomous driving. As one of the most important sensors in autonomous vehicles, light detection and ranging (LiDAR) sensors collect 3D point clouds that precisely record the external surfaces of objects and scenes. The tools for 3D point cloud processing and learning are critical to the map creation, localization, and perception modules in an autonomous vehicle. While much attention has been paid to data collected from cameras, such as images and videos, an increasing number of researchers have recognized the importance and significance of LiDAR in autonomous driving and have proposed processing and learning algorithms to exploit 3D point clouds. We review the recent progress in this research area and summarize what has been tried and what is needed for practical and safe autonomous vehicles. We also offer perspectives on open issues that are needed to be solved in the future.

I. Introduction and Motivation

As one of the most exciting engineering projects of the modern world, autonomous driving is an aspiration for many researchers and engineers across generations. It is a goal that might fundamentally redefine the future of human society and everyone's daily life. Once autonomous driving becomes mature, we will witness a transformation of public transportation, infrastructure and the appearance of our cities. The world is looking forward to exploiting autonomous driving to reduce traffic accidents caused by driver errors, to save drivers' time and liberate the workforce, as well as to save parking spaces, especially in the urban area [1].

A. Autonomous driving: History and current state

It has taken decades of effort to get closer to the goal of autonomous driving. From the 1980s through the DARPA Grand Challenge in 2004 and the DARPA Urban Challenge in 2007, the research on autonomous driving was primarily conducted in the U.S. and Europe, yielding incremental progresses in driving competence in various situations [2]. In 2009, Google started a research project on self-driving cars, and later created Waymo to commercialize the accomplishment based on their early technical success. Around 2013-2014, the rise of deep neural networks brought on the revolution of practical computer vision and machine learning. This emergence made people believe that many technical bottlenecks of autonomous driving could be fundamentally solved. In 2015, Uber created the Uber Advanced Technologies Group with the aim to enable autonomous vehicles to complete scalable ride-sharing services. This aim has become a common deployment strategy within the industry. Currently, there are numerous high-tech companies, automobile manufacturers, and start-up companies working on autonomous-driving technologies, including Apple, Aptiv, Argo AI, Aurora, Baidu, GM Cruise, Didi, Lyft, Pony.ai, Tesla, Zoox, the major automobile companies, and many others [3]. These companies have ambitious goals to achieve SAE level 4[1] in the near future. Although there has been significant progress across many groups in industry and academia, there is still much work to be done. The efforts from both industry and academia are needed to achieve autonomous driving. Recently, there have been many discussions and hypotheses about the progress and the future of autonomous driving; however, few thoughts from those who push industrial-level self-driving technologies from the frontline are publicly accessible. In this article, we provide a unifying perspective from both practitioners and researchers.

S. Chen is with Mitsubishi Electric Research Laboratories, Cambridge, MA, USA. Email: schen@merl.com. B. Liu is with Precivision Technologies, Inc., Pittsburgh, PA, USA. Email: baoanliu@precivision.tech. C. Feng is with New York University, Brooklyn, NY, USA. Email: cfeng@nyu.edu. C. Vallespi-Gonzalez and C. Wellington are with Uber Advanced Technologies Group, Pittsburgh, PA, USA. Emails: cvallespi@uber.com, cwellington@uber.com.

[1]SAE International, a transportation standards organization, introduced the J3016 standard, which defines six levels of driving automation; See details in https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic. It ranges from SAE Level Zero (no automation) to SAE Level 5 (full automation). One turning point occurs between Levels 2 and 3, where the driving responsibility shifts from a human driver to an autonomous system, and another turning point occurs between Levels 3 and 4, where the human no longer drives under any circumstances.

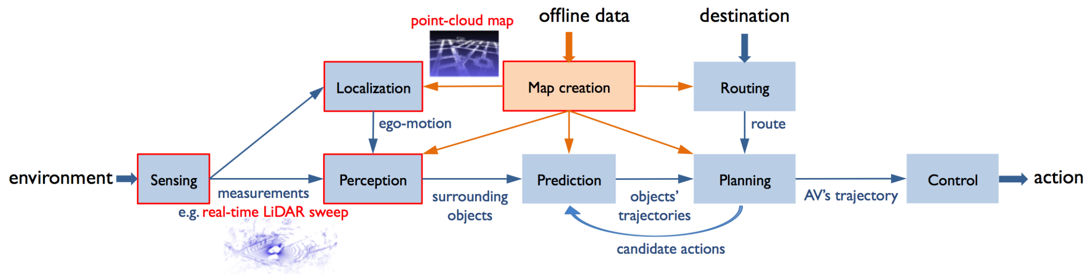


Fig. 1: High-level architecture of a typical autonomous system. A high-definition map is built offline. At runtime, the online system is given a destination. The system then senses its environment, localizes itself to the map, perceives the world around it and makes corresponding predictions of future motion for these objects. The motion planner uses these predictions to plan a safe trajectory for an autonomous vehicle (AV) to follow the route to the destination that is executed by the controller. Note that two types of 3D point clouds are used in this autonomous system: a point-cloud map, created by the map creation module and consumed by the localization module, and a real-time LiDAR sweep, collected by the sensing module and consumed by the localization and perception modules.

In industry, an autonomous system usually includes a series of modules with complicated internal dependencies. Most modules are still far from being perfect due to a number of technical bottlenecks and the long-tail issues [4]. Additionally, a small error from one module can cause problems in subsequent modules and potentially result in a substantial failure at the system level. There has been some initial research on end-to-end systems where the entire system is trained end-to-end and information can flow from sensors directly to the final planning or control decisions. These systems offer the promise to reduce internal dependency challenges; however, these systems often lack explainability and are difficult to analyze. Although significant progress has been made, there remain many open challenges in designing a practical autonomous system that can achieve the goal of full self-driving.

B. A tour of an autonomous system

An autonomous system typically includes the sensing, map creation, localization, perception, prediction, routing, planning, and control modules [5]; see Figure 1. A high-definition map is created offline. At runtime, the online system is given a destination. The system then senses its environment, localizes itself to the map, perceives the world around it and makes corresponding predictions of future motion for these objects. The motion planner uses these predictions to plan a safe trajectory for an autonomous vehicle (AV) to follow the route to the destination that is executed by the controller.

Sensing module. To ensure reliability, autonomous driving usually requires multiple types of sensors. Cameras, radio detection and ranging (RADAR), light detection and ranging (LiDAR) and ultrasonic sensors are most commonly used. Among those sensors, LiDAR is particularly interesting because it directly provides a precise 3D representation of a scene. Although the techniques for 3D reconstruction and depth estimation based on 2D images have been significantly improved with the development of deep learning based computer vision algorithms, the resulting estimations are still not always precise or reliable. Besides algorithmic constraints, fundamental bottlenecks also include inherent exponential range error growth in depth estimation, poor performance in low light, and the high computational cost of processing high-resolution images. On the other hand, LiDAR measures 3D information through direct physical sensing. A real-time LiDAR sweep consists of a large number of 3D points; called a 3D point cloud[2]. Each 3D point records the range from the LiDAR to an object's external surface, which can be transformed into the precise 3D coordinate. These 3D point clouds are extremely valuable for an autonomous vehicle to localize itself and detect surrounding objects in the 3D world. *The vast majority of companies and researchers rely heavily on LiDAR to build a reliable autonomous vehicle [6].* This is why we believe that advanced techniques for 3D point cloud processing and learning are indispensable for autonomous driving.

Map creation module. Map creation is the task of creating a high-definition (HD) map, which is a precise heterogeneous map representation of the static 3D environment and traffic rules. A HD map usually contains two

[2]The measurements from RADAR and Ultrasound are also called 3D point clouds, but we focus on 3D point clouds collected by LiDAR.

map layers: a *point-cloud map*, representing 3D geometric information of surroundings, and a *traffic-rule-related semantic feature map*, containing road boundaries, traffic lanes, traffic signs, traffic lights, etc. These two map layers are aligned together in the 3D space and provide detailed navigation information. As one map layer, the point-cloud map is a dense 3D point cloud and mainly used for providing localization prior. Different from common maps designed for humans, an HD map is designed for autonomous vehicles. The map creation module is crucial because an HD map provides valuable prior environmental information; see details in Section III.

Localization module. Localization is the task of finding the ego-position of an autonomous vehicle relative to a reference position in the HD map. This module is crucial because an autonomous vehicle must localize itself in order to use the correct lane and other important priors in the HD map. One of the core techniques is 3D point cloud registration; that is, estimating the precise location of an autonomous vehicle by matching real-time LiDAR sweeps to the offline HD map; see details in Section IV.

Perception. Perception is the task of perceiving the surrounding environment and extracting information that is related to navigation. This module is crucial because the perception module is the visual system of an autonomous vehicle, which should detect, track and classify objects in the 3D scene. It used to be considered as the technical bottleneck of autonomous driving. Recently, with large-scale training data and developments of advanced machine learning algorithms, the overall performance of the perception module has achieved tremendous improvement. Some core techniques include 2D object detection and 3D object detection. 2D object detection becomes relatively mature, while 3D object detection is based on real-time LiDAR sweeps and becomes an increasingly hot research topic; see details in Section V.

Prediction. Prediction is the task of predicting the future potential trajectories of each object in the 3D scene. This module is crucial because an autonomous vehicle needs to know the possible future behaviors of nearby objects to plan a safe trajectory.

Routing. Routing is the task of designing a high-level path from the starting position to the destination for an autonomous vehicle. The output of this module provides a high-level guideline for the planning module.

Planning. Motion planning is the task of designing a trajectory for an autonomous vehicle based on the state of current cars, surrounding environment and the destination. This module is crucial because an autonomous vehicle needs to know how to react to the surrounding environment.

Control. Control is the task of executing the commands from the planning module. It takes charge of controlling the actuators of the steering wheel, throttle, and brakes.

C. Overview of 3D point cloud processing and learning

As mentioned earlier, LiDAR provides indispensable 3D information for autonomous driving. We now move on to the processing and learning techniques that convert raw measurements into useful information.

Usages in autonomous driving. Two types of 3D point clouds are commonly used in an autonomous vehicle: a real-time LiDAR sweep and a point-cloud map, which is one layer in the HD map; see Figure 1. A point-cloud map provides prior environmental information: the localization module uses a point-cloud map as a reference in 3D point cloud registration to determine the position of the autonomous vehicle, and the perception module uses a point-cloud map to help split the foreground and the background. On the other hand, real-time LiDAR sweeps are consumed by the localization module to register against the point-cloud map, and by the perception module to detect surrounding objects in the 3D scene. Therefore, 3D point cloud processing and learning are critical to build the map creation, localization and perception modules in an autonomous system.

Recent progress in academia. Sensors capture data and data feeds algorithms. During the development of RADAR, acoustic sensors and communication systems, 1D signal processing experienced a rapid growth during the past century, leading to a revolutionary impact on digital communication systems. With the popularization of cameras and televisions, 2D image processing experienced a rapid growth during the past 30 years, resulting in a significant change to photography, entertainment, and surveillance. With the increasing needs from industrial robotics, autonomous driving and augmented reality, 3D sensing techniques is experiencing rapid development recently. At the same time, the algorithms to process and learn from 3D point clouds are starting to get much attention in academia. The following discussion is divided into two parts: 3D point cloud processing, which handles 3D point clouds from a signal-processing perspective, and 3D point cloud learning, which handles 3D point clouds from a machine-learning perspective.

3D point cloud processing. 3D point cloud processing is the process of analyzing and modifying a 3D point cloud to optimize its transmission, storage and quality through various mathematical and computational algorithms. Even though the processing algorithms could be significantly different, many processing tasks are naturally extended from 1D signal processing and 2D image processing. For example, 3D point cloud compression is the 3D counterpart of image compression that aims to reduce the cost for storage or transmission of a 3D point cloud; 3D point cloud denoising is the 3D counterpart of image denoising that aims to remove noise from a 3D point cloud; 3D point cloud registration is the 3D counterpart of image registration that aims to align two or more 3D point clouds of the same scene; and 3D point cloud downsampling and upsampling are the 3D counterpart of image scaling that aims to change the resolution (number of points) in a 3D point cloud.

3D point cloud learning. 3D point cloud learning is the process of interpreting and understanding a 3D point cloud. With the powerful tools of deep neural networks, computer vision researchers aim to extend the success from images and videos to 3D point clouds. Two primary learning problems are 3D point cloud recognition and segmentation. Similarly to the cases for 2D images, 3D point cloud recognition aims to classify a given 3D point cloud into a predefined class category and 3D point cloud segmentation aims to partition a given 3D point cloud into multiple segments. Due to the irregular format of 3D point clouds, one of the biggest challenges for designing a learning algorithm is to formulate efficient data structures to represent 3D point clouds. Some algorithms transform 3D point clouds to regular 3D voxels, so that 3D convolutions can be used for the analysis; however, they have to make a trade-off between resolution and memory. To handle raw point clouds directly, PointNet [7] uses point-wise multilayer perceptrons (MLPs) and max-pooling to ensure the permutation invariance. After that, a series of 3D deep learning methods follow PointNet as their base networks.

Relations between academia and industry. The technical transition from 1D time-series to 2D images is quite natural, because both types of data are supported on regular-spacing structures; however, the technical transition from 2D images to 3D point clouds is not straightforward because those points are irregularly scattered in a 3D space. Numerous popular methods to handle 3D point clouds are proposed heuristically by practitioners. Therefore, there is a substantial room for both researchers and practitioners to collaborate and solve fundamental tasks on 3D point cloud processing and learning, so that we can accelerate the progress of autonomous driving.

D. *Outline*

The outline of this article is as follows: Section II presents key ingredients of 3D point cloud processing and learning. It starts by explaining common properties of a 3D point cloud, followed by various approaches to represent a 3D point cloud. It then presents modern methods to process and learn from a 3D point cloud. Sections III, IV, and V cover the state-of-the-art methods and challenges about 3D point cloud processing and learning in the map creation, localization and perception modules of an autonomous system, respectively. We specifically consider these three modules because they heavily rely on 3D point clouds to achieve reliable performance. In each module, we discuss *what* this module is specifically working on; *why* 3D point cloud processing and learning are significant for this module; and *how* 3D point cloud processing and learning make a difference in this module. Section VI concludes with discussion and pointers to future directions. In the supplementary material, we compare the perspectives between academia and industry in Section I, illustrate the latest qualitative results in Section II, and overview a series of elementary tasks about 3D point clouds that have received much attention in academia in Section III.

II. Key Ingredients of 3D Point Cloud Processing and Learning

In this section, we introduce basic tools of 3D point cloud processing and learning. We start with the key properties of 3D point clouds. We next evaluate some options for representing a 3D point cloud. Finally, we review a series of popular tools to handle 3D point clouds. Those tools have received great attention in academia. Even some of them might not be directly applied to an autonomous system, it is still worth mentioning because they could inspire new techniques, which are potentially useful to autonomous driving.

A. *Properties*

As discussed in Section I-C, we consider two typical types of 3D point clouds in autonomous driving: real-time LiDAR sweeps and point-cloud maps.

Real-time LiDAR sweeps. Because of the sensing mechanism, for each 3D point in a real-time LiDAR sweep, we can trace its associated laser beam and captured time stamp. One real-time LiDAR sweep can naturally be organized on a 2D image, whose x-axis is the time stamp and y-axis is the laser ID. We thus consider each individual real-time LiDAR sweep as an *organized 3D point cloud*. For example, a Velodyne HDL-64E has 64 separate lasers and each laser fires thousands of times per second to capture a 360-degree field of view. We thus obtain a set of 3D points associated with 64 elevation angles and thousands of azimuth angles[3]. Each collected 3D point is associated with a range measurement, an intensity value and a high precision GPS time stamps. Note that for a global-shutter image, the pixel values are collected by a charge-coupled device (CCD) at the same time; however, for a real-time LiDAR sweep, the 3D points are collected at various time stamps. For the same laser, firings happen sequentially to collect 3D points; for different lasers, firings are not synchronized either; thus, the collected 3D points are not perfectly aligned on a 2D regular lattice. Since the arrangement of 64 lasers follows a regular angular spacing, the point density of a real-time LiDAR sweep changes over the range; that is, we collect many more 3D points from nearby objects than from far-away objects. Moreover, a real-time LiDAR sweep naturally suffers from the occlusion; that is, we get 3D points only from the sides of objects facing the LiDAR. To summarize, some key properties of a real-time LiDAR sweep include:

- Pseudo 3D. A real-time LiDAR sweep arranges 3D points approximately on a 2D lattice. Due to the non-perfect synchronization, 3D points are not perfectly aligned on a 2D lattice. Meanwhile, unlike a 3D point cloud obtained from multiple views, a real-time LiDAR sweep only reflects a specific view; we thus consider its dimension *pseudo 3D*;
- Occlusion. Each individual real-time LiDAR sweep records the 3D environment almost from a single viewpoint[4]. A front object would occlude the other objects behind it; and
- Sparse point clouds. Compared to a 2D image, a real-time LiDAR sweep is usually sparse representations of objects, especially for far-away objects. It cannot provide detailed 3D shape information of objects.

Point-cloud maps. To create a point-cloud map, one needs to aggregate real-time LiDAR sweeps scanned from multiple autonomous vehicles across time. Since there is no straightforward way to organize a point-cloud map, we consider it as an *unorganized 3D point cloud*. For example, for a 200×200 square meter portion of an HD map, one needs to aggregate the LiDAR sweeps around that area for 5-10 trials, leading to over 10 millions 3D points. Since LiDAR sweeps could be collected from significantly different views, an HD map after aggregation gets denser and presents a detailed 3D shape information. To summarize, some key properties of a point-cloud map include:

- Full 3D. A point-cloud map aggregates multiple LiDAR sweeps from various views, which is similar to 3D data collected by scanning an object on a turntable. A point-cloud map captures information on more objects' surfaces, providing a denser and more detailed 3D representation;
- Irregularity. 3D points in a point-cloud map are irregularly scattered in the 3D space. They come from multiple LiDAR sweeps and lose the laser ID association, causing an unorganized 3D point cloud;
- No occlusion. A point-cloud map is an aggregation of 3D points collected from multiple viewpoints. It depicts the static 3D scene with much less occlusion;
- Dense point clouds. A point-cloud map provides a dense point cloud, which contains detailed 3D shape information, such as high-resolution shapes and the surface normals; and
- Semantic meanings. As another layer in the HD map, a traffic-rule-related semantic feature map contain the semantic labels of a 3D scene, including road surfaces, buildings and trees. Since a traffic-rule-related semantic feature map and a point-cloud map are aligned in the 3D space, we can trace the semantic meaning of each 3D point. For example, 3D points labeled as trees in a point-cloud map would help improve perception as LiDAR points on leaves of trees are usually noisy and difficult to be recognized.

B. Matrix representations

Representations have always been at the heart of most signal processing and machine learning techniques. A good representation lays the foundation to uncover hidden patterns and structures within data and is beneficial

[3]In a real-time LiDAR sweep, the vertical resolution is usually much lower than the horizontal resolution.

[4]Since the autonomous vehicle could move in real-time, the viewpoint of LiDAR would change slightly.

for subsequent tasks. A general representation of a 3D point cloud is through a set, which ignores any order of 3D points. Let $\mathcal{S} = \{(\mathbf{p}_i, \mathbf{a}_i)\}_{i=1}^N$ be a set of N 3D points, whose ith element $\mathbf{p}_i = [x_i, y_i, z_i] \in \mathbb{R}^3$ represents the 3D coordinate of the ith point and $\mathbf{a}_i$ represents other attributes of the ith point. A real-time LiDAR sweep usually includes the intensity $\mathbf{a}_i = r_i \in \mathbb{R}$ and a point-cloud map usually includes surface normals $\mathbf{n}_i \in \mathbb{R}^3$; thus, $\mathbf{a}_i = [r_i, \mathbf{n}_i] \in \mathbb{R}^4$. For generality, we consider the feature of the ith point as $\mathbf{x}_i = (\mathbf{p}_i, \mathbf{a}_i) \in \mathbb{R}^d$.

For efficient storage and scientific computation, a matrix (or tensor) representation is appealing. Let f be the mapping from a set of 3D points $\mathcal{S}$ to a matrix (or tensor) X with a pending shape. A matrix representation of a 3D point cloud is thus $\mathrm{X} = f(\mathcal{S})$. We next discuss a few typical approaches to implement the mapping $f(\cdot)$.

Raw points. The most straightforward matrix representation of a 3D point cloud is to list each 3D point in the set $\mathcal{S}$ as one row in the matrix. Consider

$$\mathrm{X}^{(\mathrm{raw})} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{bmatrix}^T \in \mathbb{R}^{N \times d}, \tag{1}$$

whose ith row $\mathrm{X}_i^{(\mathrm{raw})} = \mathbf{x}_i \in \mathbb{R}^d$ is the features of ith point in the 3D point cloud.

The advantages of the raw-point-based representation are that i) it is simple and general; ii) it preserves all the information in the original set of 3D points; however, the shortcoming is that it does not explore any geometric property of 3D points. This representation is generally used in the map and the localization module of an autonomous system, where high precision is needed.

3D voxelization. To enjoy the success of 2D image processing and computer vision, we can discretize the 3D space into voxels and use a series of voxels to represent a 3D point cloud. A straightforward discretization is to partition the 3D space into equally-spaced nonoverlapping voxels from each of three dimensions; see Figure 2 (a). Let a 3D space with range H, W, D along the X, Y, Z axes, respectively. Each voxel is of size h, w, d, respectively. The (i, j, k)th voxel represents a 3D voxel space, $\mathcal{V}_{i,j,k} = \{(x, y, z) | (i-1)h \le x < ih, (j-1)w \le y < jw, (k-1)d \le z < kd\}$. We then use a three-mode tensor to represent this 3D point cloud. Let $\mathrm{X}^{(\mathrm{vox})} \in \mathbb{R}^{H \times W \times D}$, whose (i, j, k)th element is

$$\mathrm{X}_{i,j,k}^{(\mathrm{vox})} = \begin{cases} 1, & \text{when } \mathcal{V}_{i,j,k} \cap \mathcal{S} \neq \varnothing; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The tensor $\mathrm{X}^{(\mathrm{vox})}$ records the voxel occupancy.

The advantages of the 3D-voxelization-based representation are that (i) the resulting voxels are associated with a natural hierarchical structure and all the voxels have a uniform spatial size; and (ii) we can use off-shelf tools, such as 3D convolutions to analyze data; however, the shortcomings are that (i) it does not consider specific properties of organized 3D point clouds; (ii) it usually leads to an extremely sparse representation where most voxels are empty; and (iii) it involves a serious trade-off between the resolution and the memory. This representation can be used in the perception module of autonomous driving, as well as the storage of 3D point clouds.

Range view. As discussed in Section II-A, a real-time LiDAR sweep is essentially a series of range measurements from a single location with certain angular field of view; see Figure 2 (b). We can approximately organize the 3D points in a real-time LiDAR to a 2D range-view image. Each pixel in the range-view image corresponds to a frustum in the 3D space. The pixel value is the range from the LiDAR to the closest 3D point inside the frustum. Specifically, we partition the 3D space along the azimuth angle $\alpha \in [0, 2\pi)$ and the elevation angle $\theta \in (-\pi/2, \pi/2]$ with the resolution of azimuth angle α_0 and the resolution of elevation angle θ_0. The (i, j)th pixel corresponds to a frustum space, $\mathcal{V}_{i,j} = \{(x, y, z) | \alpha_0(i-1) \le \mathrm{acos}(\frac{x}{\sqrt{x^2+y^2}}) < \alpha_0 i, \theta_0(j-1) \le \mathrm{atan}(\frac{z}{\sqrt{x^2+y^2}}) + \frac{\pi}{2} < \theta_0 j\}$. We then use a 2D matrix to represent a 3D point cloud. Let $\mathrm{X}^{(\mathrm{FV})} \in \mathbb{R}^{H \times W}$, whose (i, j)th element is

$$\mathrm{X}_{i,j}^{(\mathrm{FV})} = \begin{cases} \min_{(x,y,z) \in \mathcal{V}_{i,j} \cap \mathcal{S}} \sqrt{x^2 + y^2 + z^2}, & \mathcal{V}_{i,j,k} \cap \mathcal{S} \neq \varnothing; \\ -1, & \text{otherwise.} \end{cases} \tag{3}$$

We consider the smallest range value in each frustum space. When no point falls into the frustum space, we set a default value as -1. Note that the range-view-based representation could also use nonuniform-spaced elevation angles according to the LiDAR setting.

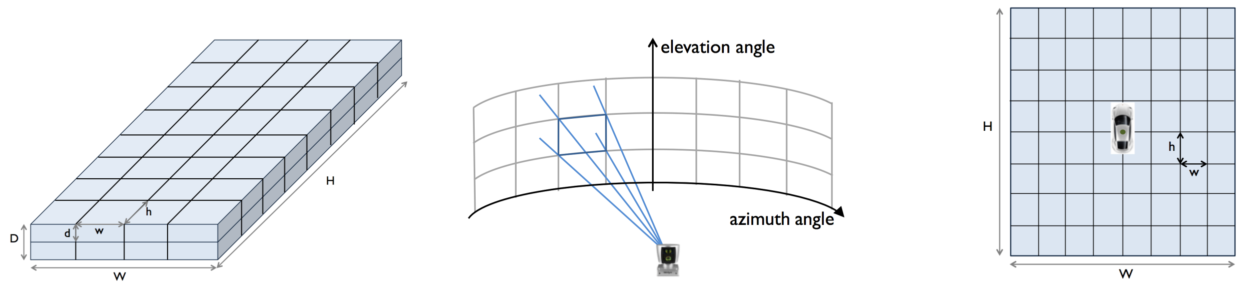
The advantages of the range-view-based representation are that (i) it naturally models how LiDAR captures 3D points, reflecting a 2D surface in the 3D space; (ii) Most frustum spaces associated have one or multiple 3D points, leading to a compact range-view image; however, the shortcoming is that it is difficult to model an unorganized point cloud, such as the point-cloud map in an HD map. This representation can be used in the perception module.

Bird's-eye view. The bird's-eye-view (BEV)-based representation is a special case of 3D voxelization by ignoring the height dimension. It projects 3D voxels to a BEV image; see Figure 2 (c). Let a 3D space with range H, W along the X, Y axes, respectively. Each pixel is of size h, w, respectively. The (i, j)th pixel in the BEV image represents a pillar space, $\mathcal{V}_{i,j} = \{(x, y, z) | (i-1)h \le x < ih, (j-1)w \le y < jw\}$. We then use a 2D matrix to represent a 3D point cloud. Let $\mathrm{X}^{(\mathrm{BEV})} \in \mathbb{R}^{H \times W}$, whose (i, j)th element is

$$\mathrm{X}^{(\mathrm{BEV})}_{i,j} = \begin{cases} 1, & \text{when } \mathcal{V}_{i,j} \cap \mathcal{S} \neq \varnothing; \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The matrix $\mathrm{X}^{(\mathrm{BEV})}$ records the occupancy in the 2D space. Note that there are a few variations of the BEV-based representations. For example, instead of using a binary value, MV3D [8] uses a few statistical values in each pillar space to construct $\mathrm{X}^{(\mathrm{BEV})}$.

The advantages of the BEV-based representation are that (i) it is easy to apply 2D vision-based techniques; (ii) it is easy to merge with information from the HD map. For example, drivable areas and the positions of intersections encoded in the HD map can be projected to the same 2D space and fused with LiDAR information; (iii) it is easy to use for subsequent modules, such as prediction and motion planning, and (iii) objects are always the same size regardless of range (contrasting with the range-view-based representation), which is a strong prior and makes the learning problem much easier; however, the shortcoming of this voxelization is that (i) it also involves a serious trade-off between resolution and memory, causing excessive quantization issues of getting detailed information on small objects; (ii) it does not consider the specific properties of organized 3D point clouds and cannot reason the occlusion; and (iii) it causes the sparsity issue because most pixels are empty. This representation can be used in the perception module of autonomous driving.



(a) 3D voxel-based representation. (b) Range-view-based representation. (c) Bird's-eye-view-based representation.

Fig. 2: Common approaches to discretize the 3D space. The 3D voxel-based representation is to discretize the 3D space into equally-spaced nonoverlapping voxels from each of the three dimensions; the range-view-based representation is to discretize the 3D space along the azimuth angle and the elevation angle; and the bird's-eye-view-based representation is to discretize the 3D space along the X, Y axes, omitting the height dimension.

C. Representative tools

3D point clouds have been studied across various communities, such as robotics, computer graphics, computer vision and signal processing. We introduce a few representative tools to process and learn from 3D point clouds. We mainly emphasize deep-neural-network-based approaches because of their practical usages in autonomous driving.

Non-deep-learning methods. Before the emergence of deep learning, there have been many traditional methods to handle 3D point clouds for various tasks. However, unlike deep neural networks, those conventional methods can hardly be described in a single methodological framework. This is because hand-crafted tools are specifically designed to cater to the needs of each individual task. For example, in 3D point cloud segmentation and 3D shape detection, traditional techniques have been developed based on either region growth with simple geometric heuristics, or graph-based optimization, or robust estimation methods, such as RANSAC [9]. As another important task, 3D keypoint matching is closely related to 3D point cloud registration and 3D point cloud recognition. To tackle this task, many statistics-based methods have been developed in a hand-crafted fashion and aim to describe the geometric structures around 3D keypoints or objects; see a more comprehensive discussion in [10].

Convolutional neural networks. The motivation of using convolutional neural networks is to leverage off-shelf deep learning tools to process 3D point clouds. As regularized versions of multilayer perceptrons, convolutional neural networks (CNNs) employ a series of convolution layers and are commonly applied to analyzing images and videos. A convolution layer operates a set of learnable filters on input data to produce the output that expresses the activation map of filters. The beauty of a convolution layer is weight-sharing; that is, the same filter coefficients (weights) are applied to arbitrary positions in a 2D image, which not only saves a lot of learnable weights, but also ensures shift invariance, and helps avoid overfitting to limited training data. As a general and mature learning framework, CNNs and common variations are widely used in various computer vision tasks, including classification, detection, and segmentation, and have achieved state-of-the-art performance in most tasks.

Based on the success of CNNs in images and videos, CNNs have been applied to 3D point cloud data as well. Multiple representations have been used, including the 3D-voxelization-based representation (2), the range-view-based representation (3) and the BEV-based representation (4). A benefit of using CNNs to handle a 3D point cloud is that a convolution operator naturally involves local spatial relationships. In PointNet, each 3D point is processed individually; while in CNNs, adjacent voxels or pixels are considered jointly, providing richer contextual information. The basic operator is a 3D convolution for the 3D voxelization-based representation and a 2D convolution for the range-view-based representation and the BEV-based representation, respectively. Without loss of generality, consider a 4-mode tensor $\mathrm{X} \in \mathbb{R}^{I \times J \times K \times C}$, after convolving with C 3D filters $\mathrm{H} \in \mathbb{R}^{k \times k \times k \times C}$, the (i, j, k, c')th element of the output $\mathrm{Y} \in \mathbb{R}^{I \times J \times K \times C'}$ is

$$\mathrm{Y}_{i,j,k,c'} \ = \ \sum_{c=0}^{C-1} \sum_{\ell=0}^{k-1} \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \mathrm{H}_{i-\ell, j-m, k-n, c'} \, \mathrm{X}_{\ell, m, n, c} \, .$$

For simplicity, we omit the boundary issue. 3D convolution is expensive in both computation and memory usage.

Because of the discretization, many techniques and architectures developed for 2D images can be easily extended to handle 3D point clouds. Even though the discretization causes inevitable loss of information, CNNs usually provide reliable performances and are widely used in many tasks. As discussed previously, one critical issue about discretizing a 3D point cloud is that a resulting 3D volume or 2D image is sparse. A huge amount of computation is wasted in handling empty voxels.

To summarize, CNNs handle a 3D point cloud in a discretized representation. This approach inevitably modifies the exact 3D position information, but still provides strong and promising empirical performances because of the spatial relationship prior and the maturity of CNNs. It is thus widely used in the industry.

PointNet-based methods. The motivation of using PointNet-based methods is to directly handle raw 3D points by deep neural networks without any discretization. PointNet [7] is a pioneering work that achieves this goal. Raw 3D point clouds are inherently unordered sets, and PointNet was designed to respect this property and produce the same output regardless of the ordering of the input data. The key technical contribution of PointNet is to use a set of shared point-wise multi-layer perceptrons (MLPs) followed by global pooling to extract geometric features while ensuring this permutation-invariant property of raw 3D data. Even though the architecture is simple, it has become a standard building block for numerous 3D point cloud learning algorithms and achieves surprisingly strong performance on 3D point cloud recognition and segmentation.

PointNet considers the raw-point-based representation $\mathrm{X}^{(\mathrm{raw})}$ (1). Let $\mathrm{H} \in \mathbb{R}^{N \times D}$ be a local-feature matrix, where the ith row H_i represents the features for ith point, and $\mathbf{h} \in \mathbb{R}^D$ be a global-feature vector. A basic computational block of PointNet works as

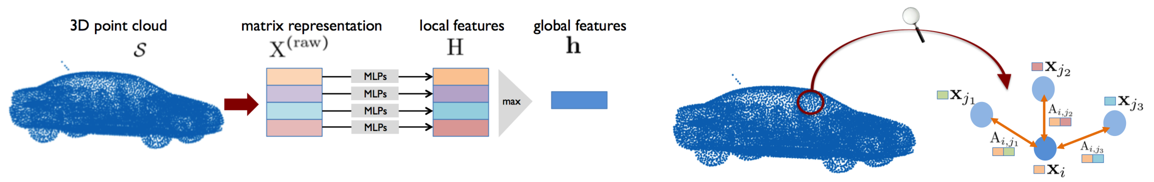
$$\begin{aligned} \mathrm{H}_i &= \mathrm{MLP}^{(\mathrm{L})}\left(\mathrm{X}_i^{(\mathrm{raw})}\right) \in \mathbb{R}^D, \quad \text{for } i = 1, \cdots, N, \\ \mathbf{h} &= \mathrm{maxpool}\left(\mathrm{H}\right) \in \mathbb{R}^D, \end{aligned} \tag{5}$$

where $\mathrm{X}_i^{(\mathrm{raw})}$ is the ith 3D point's feature, and $\mathrm{MLP}^{(\mathrm{L})}(\cdot)$ denotes a L-layer MLPs, which map each 3D point to a feature space, and $\mathrm{maxpool}(\cdot)$ performs downsampling by computing the maximum values along the column (the point dimension); see Figure 3 (a). Note that each 3D point goes through the same MLPs separately.

Intuitively, the MLPs propose D representative geometric patterns and test if those patterns appear around each 3D point. The max-pooling records the strongest response over all the 3D points for each pattern. Essentially, the global-feature vector $\mathbf{h}$ summarizes the activation level of D representative geometric patterns in a 3D point cloud, which can be used to recognize a 3D point cloud. Meanwhile, since each 3D point goes through the same MLPs separately and the max-pooling removes the point dimension, the entire computational block is permutation invariant; that is, the ordering of 3D points does not influence the output of this block. To some extent, PointNet for 3D point cloud learning is similar to principal component analysis (PCA) for data analysis: it is simple, general and effective. Just like principal component analysis, PointNet extracts global features in a 3D point cloud.

To summarize, PointNet-based methods handle 3D point clouds in the raw-point-based representation and ensure the permutation invariance. The effectiveness has been validated in various processing and learning tasks.

Graph-based methods. The motivation of using graph-based methods is to leverage the spatial relationships among 3D points to accelerate the end-to-end learning of deep neural networks. One advantage of CNNs is that a convolution operator considers local spatial relationships; however, those relationships are between adjacent voxels (or adjacent pixels), not original 3D points. To capture the local relationships among 3D points, one can introduce a graph structure, where each node is a 3D point and each edge reflects the relationship between each pair of 3D points. This graph structure is a discrete proxy of the surface of an original object. A matrix representation of a graph with N nodes is an adjacency matrix $\mathrm{A} \in \mathbb{R}^{N \times N}$, whose (i,j)th element reflects the pairwise relationship between the ith and the jth 3D points; see Figure 3 (b). Graph-based methods usually consider the raw-point-based representation (1). Each column vector in $\mathrm{X}^{(\mathrm{raw})}$ is then data supported on the graph A; called a graph signal.



(a) PointNet. (b) Graph-based methods.

Fig. 3: Illustration of representative tools. Plot (a) shows that PointNet uses a set of shared point-wise multi-layer perceptrons (MLPs) followed by max-pooling to extract geometric features that exhibit the permutation-invariant property of raw 3D point clouds. Plot (b) shows that graph-based methods introduce a graph structure to capture the local relationships among 3D points. In the graph, each node is a 3D point and each edge reflects the relationship between each pair of 3D points.

There are several ways to construct a graph, such as a K-nearest-neighbor graph, an ϵ-graph and a learnable graph. A K-nearest-neighbor graph is a graph in which two nodes are connected by an edge, when their Euclidean distance is among the K-th smallest Euclidean distances from one 3D point to all the other 3D points. An ϵ-nearest-neighbor graph is a graph in which two nodes are connected by an edge, when their Euclidean distance is smaller than a given threshold ϵ. Both K-nearest-neighbor graphs and ϵ-graphs can be efficiently implemented by using efficient data structures, such as Octree [11]. A learnable graph is a graph whose adjacency matrix is trainable in an end-to-end learning architecture.

A general graph-based operation is a graph filter, which extends a classical filter to the graph domain and extracts features from graph signals. The most elementary nontrivial graph filter is called a *graph shift operator*. Some common options for a graph shift operator include the adjacency matrix A, the transition matrix $\mathrm{D}^{-1}\mathrm{A}$ (D is the weighted degree matrix, a diagonal matrix with $\mathrm{D}_{i,i} = \sum_j \mathrm{A}_{i,j}$ reflecting the density around the ith point), the graph Laplacian matrix $\mathrm{D} - \mathrm{A}$, and many other structure-related matrices. The graph shift replaces the signal value at a node with a weighted linear combination of values at its neighbors; that is, $\mathrm{Y} = \mathrm{A}\,\mathrm{X}^{(\mathrm{raw})} \in \mathbb{R}^N$, where $\mathrm{X}^{(\mathrm{raw})} \in \mathbb{R}^{N\times 3}$ is an input graph signal (an attribute of a point cloud). Every linear, shift-invariant graph filter is a polynomial in the graph shift,

$$h(\mathrm{A}) = \sum_{\ell=0}^{L-1} h_\ell \mathrm{A}^\ell = h_0 \mathrm{I} + h_1 \mathrm{A} + \ldots + h_{L-1} \mathrm{A}^{L-1},$$

where h_ℓ, $\ell = 0, 1, \ldots, L-1$ are filter coefficients and L is the graph filter length. A higher order corresponds to a larger receptive field on the graph vertex domain. The output of graph filtering is given by the matrix-vector product $\mathrm{Y} = h(\mathrm{A})\,\mathrm{X}^{(\mathrm{raw})}$. Graph filtering can be used in various processing tasks, such as 3D point cloud downsampling and denoising [12].

Inspired by the success of graph neural networks in social network analysis, numerous recent research incorporate graph neural networks to handle a 3D point cloud. As the first such work, [13] introduces two useful techniques: the edge convolution operation and learnable graphs. The edge convolution is a convolution-like operation to extract geometric features on a graph. The edge convolution exploits local neighborhood information and can be stacked to learn global geometric properties. Let $\mathrm{H} \in \mathbb{R}^{N \times d}$ be a local-feature matrix, where the ith row H_i represents the features for the ith point. A basic computational block works as $\mathrm{H}_i = \|_{(i,j) \in \mathcal{E}}\, g(\mathrm{X}_i^{(\mathrm{raw})}, \mathrm{X}_j^{(\mathrm{raw})}) \in \mathbb{R}^d$, where $\mathcal{E}$ is the edge set and $g(\cdot, \cdot)$ is a generic mapping, implemented by some neural networks, and $\|$ is a generic aggregation function, which could be the summation or maximum operation. To some extent, the edge convolution extends PointNet by inputting a pair of neighboring points' features. The edge convolution is also similar to graph filtering: both aggregates neighboring information; however, the edge convolution specifically models each pairwise relationships by a nonparametric function. [13] also suggests to dynamically learn a graph. It always uses a kNN graph, but the distance metric is the Euclidean distance in the high-dimensional feature space.

Subsequent research has proposed to use novel graph neural networks to handle 3D point cloud recognition and segmentation. As one of the most recent works in this area, [14] constructs the deepest yet graph convolution network (GCN) architecture, which has 56 layers. It transplants a series of techniques from CNNs, such as residual and dense connections, and dilated graph convolutions, to the graph domain.

To summarize, graph-based methods build graph structures to capture the distribution of a 3D point cloud and take advantage of local spatial relationships. This approach handles 3D point clouds in the raw-point-based representation, ensuring the permutation invariance. This approach is less mature: even though leveraging a graph improves the overall performance, graph construction is more art than science and takes extra computational cost [13]; additionally, deep architectures for graph-based neural networks still needs more exploration [14].

III. 3D Point Cloud Processing for High-Definition Map Creation

A. *Overview of high-definition map creation module*

To precisely represent the static 3D environment and traffic rules, a high-definition (HD) map usually contains two map layers: a point-cloud map, representing 3D geometric information of surroundings, and a traffic-rule-related semantic feature map, containing road boundaries, traffic lanes, traffic signs, traffic lights, the height of the curbs, etc. The main reason for creating an offline HD map is that understanding traffic rules in real-time is too challenging. For example, based on the current technology, it is difficult for an autonomous vehicle to determine the correct lane in real-time when driving into at an intersection with complicated lane merging and splitting. In contrast, all traffic rules and environmental information can easily be encoded in an HD map, which goes through an offline process with human supervision and quality assurance. An HD map provides strong and indispensable priors and fundamentally eases the designs of multiple modules in an autonomy system, including localization, perception, prediction and motion planning. Therefore, an HD map is widely believed to be an indispensable component of autonomous driving.

Priors for localization. The role of localization is to localize the pose of an autonomous vehicle. In an HD map, the point-cloud map and the traffic-rule-related semantic features, such as lane markers and poles, are usually served as localization priors for the map-based localization. These priors are used to register real-time LiDAR sweeps to the point-cloud map, such that one can obtain the real-time high-precision ego-motion of an autonomous vehicle.

Priors for perception. The role of perception is to detect all objects in the scene, as well as their internal states. The perception module can use an HD-map to serve as a prior for detection. For example, the positions of traffic lights in an HD map are usually served as perception priors for traffic light state estimation. With the point-cloud map as priors, one can separate a real-time LiDAR sweep into foreground and background points in real-time. We can then remove background points, which are those lying on the static scenes, such as road surfaces and the trunks of trees, and feed only foreground points to the perception module. This formalism can significantly reduce the computational cost and improve the precision of object detection.

Priors for prediction. The role of prediction is to predict the future trajectory of each object in the scene. In an HD map, 3D road and lane geometries and connectivities are important priors to the prediction module. These priors can be used to guide the predicted trajectories of objects to follow the traffic lanes.

Priors for planning. The role of motion planning is to determine the trajectory of an autonomous vehicle. In an HD map, traffic-rule-related semantic features such as lane geometries and connectivities, traffic light, traffic sign and the speed limit of lanes, are indispensable priors for the planning module. These priors are used to guide the designed trajectory to follow the correct lane and obey the stop signs and other traffic signs.

Since an HD map is critical to autonomous driving, it must be created with high precision and be up-to-date. To achieve this, it usually needs sophisticated engineering procedures to analyze data from multiple modalities by exploiting both machine learning techniques and human supervision. A standard map creation module includes two core components: *3D point cloud stitching* and *semantic feature extraction*; see Figure 4. 3D point cloud stitching merges real-time LiDAR sweeps collected from multiple vehicles across times into a point-cloud map; and semantic feature extraction extracts semantic features, such as lane geometries and traffic lights, from the point-cloud map. See a video illustration of the industrial-level HD maps through the link[5] and additional illustrations in the supplementary material.

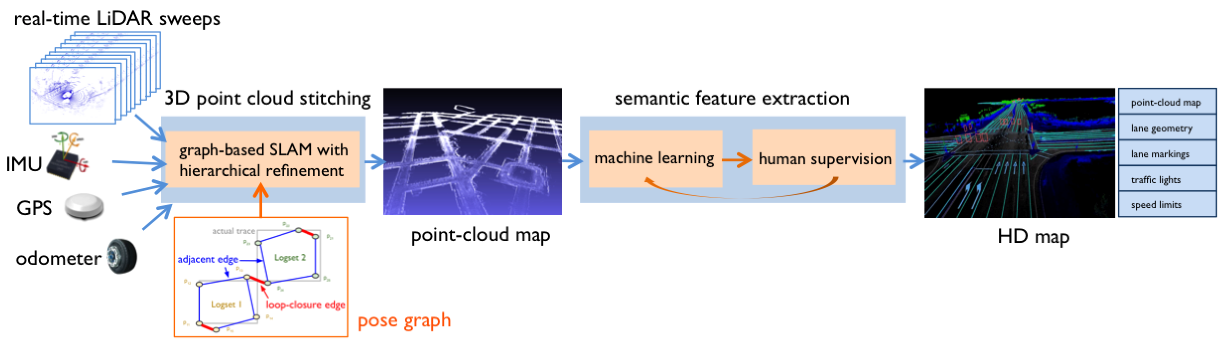


Fig. 4: A standard HD map creation system includes two core components: 3D point cloud stitching and semantic feature extraction. 3D point cloud stitching usually adopts graph-based SLAM with hierarchical refinement; and semantic feature extraction contains iterative procedures of machine learning and human supervision. A key component in graph-based SLAM is a pose graph, modeling the relations among LiDAR poses. The nodes are LiDAR poses and edges reflecting the misalignment level between two LiDAR poses. The final outputs include a point-cloud map, which is a dense 3D point cloud, as well as a traffic-rule-related semantic feature map, containing the positions of landmarkers, traffic signs and traffic lights.

B. 3D point cloud stitching

The goal of 3D point cloud stitching is to create a high-precision point-cloud map from the sensor data collected by a fleet of vehicles across time. Since a point-cloud map dominates the precision of all the map priors, centimeter-level precision is required for any local portion of the point-cloud map. To promptly create and update city-scale HD maps, the process of 3D point cloud stitching must be highly robust and efficient.

One fundamental problem of 3D point cloud stitching is to estimate the 6-degree-of-freedom (DOF) pose of each LiDAR sweep; also called LiDAR pose. We consider the map frame as the standardized global frame, and the LiDAR frame as the ego frame of an autonomous vehicle at the time stamp when the corresponding real-time LiDAR sweep is collected. A LiDAR pose is then a transformation between the map frame and the LiDAR frame. It includes 3D translation and 3D rotation. Note that the 6-DOF pose can be represented as a 4×4 homogeneous transformation matrix. With the LiDAR poses, all the LiDAR sweeps can be synchronized to the standardized global frame and integrated to form a dense 3D point cloud. To estimate LiDAR poses, a common technique is simultaneous localization and mapping (SLAM). Let $\mathcal{S}_i$ and $\mathcal{S}_j$ be the ith and jth real-time LiDAR sweeps, respectively. SLAM works as

$$\operatorname{argmin}_p \left[\sum_{p_i} \sum_{p_j} h_{\mathcal{S}_i, \mathcal{S}_j}(p_i, p_j) + g(p_i) \right], \tag{6}$$

where p_i is the 6-DOF LiDAR pose associated to the ith real-time LiDAR sweep, $h_{\mathcal{S}_i, \mathcal{S}_j}(p_i, p_j)$ indicates the negative log likelihood of the measurement on the misalignment between $\mathcal{S}_i$ and $\mathcal{S}_j$, and $g(\cdot)$ indicates the negative

[5]https://vimeo.com/303412092

log likelihood of the difference between the predicted LiDAR position in the map frame and the direct measurement of GPS [15]. A typical choice of $h_{\mathcal{S}_i,\mathcal{S}_j}(p_i, p_j)$ is the objective function of the iterative closest point (ICP) algorithm. We thus minimize the objective function of the ICP algorithm and assign the optimized value to $h_{\mathcal{S}_i,\mathcal{S}_j}(p_i, p_j)$.

SLAM is a big research field in robotics communities and there exists extensive research that aims to solve the optimization problem (6). For example, the filter-based SLAM solves the optimization problem (6) in an approximated and online fashion. It employs Bayes filtering to predict and optimize the map and LiDAR poses iteratively based on the online sensor measurements. On the other hand, the graph-based SLAM optimizes all the LiDAR poses together by using all sensor measurements across time. It constructs a pose graph that models the relations among LiDAR poses, In the pose graph, the ith node is the ith LiDAR pose, p_i; and the (i, j)th edge is the cost of misalignment between the ith and jth LiDAR poses, $h_{\mathcal{S}_i,\mathcal{S}_j}(p_i, p_j)$; see the pose graph in Figure 4. Intuitively, each edge weight is either the total point-to-point distance or the total point-to-plane distance between two LiDAR sweeps. Solving (6) is thus equivalent to minimizing the total sum of the edge weights of a pose graph.

For a city-scale map creation, the SLAM solution must satisfy the following requirements.

High local and global precision. *Local precision* indicates that the LiDAR poses in a local region are accurate with respect to one another; and *global precision* indicates that all the LiDAR poses in the entire HD map are accurate with respect to the standardized global frame. For the SLAM solution, centimeter/micro-radian level local precision must be achieved because autonomy software modules require the highly accurate local surroundings from the HD map; and the centimeter-level global precision is useful to accelerate the HD map update process especially for the city-scale application;

High robustness. The SLAM solution requires to handle the noisy sensor measurements collected by multiple vehicles driving in complicated scenes and complex driving conditions in the real world; and

High efficiency. The SLAM solution requires to handle the optimization of over 100 millions of LiDAR poses.

To achieve high precision and robustness, the graph-based SLAM is a better option than the filter-based SLAM because the global optimization formalism makes the graph-based SLAM inherently more accurate; however, it is still challenging to solve the city-scale graph-based SLAM problem with high efficiency and robustness. There are two main reasons. First, the scale of the problem is enormous. It is expensive to solve the optimization problem (6) in a brute-force way because the core step of the optimization algorithm is to solve a series of equation associated with an n-by-n matrix, where n is the total number of LiDAR poses. For a city-scale map, n could be more than 100 millions, causing big issues for both computational efficiency and numerical stability of the optimization algorithm. Second, evaluating edge weights in a pose graph usually suffers from low precision because sensor data is collected in complex driving conditions. For example, the calculation of the misalignment between consecutive LiDAR sweeps will likely be compromised by the moving objects.

To effectively solve this problem, the graph-based SLAM with the hierarchical refinement formalism can be adopted [16]. The functionality of hierarchical refinement formalism is to provide a good initialization for the global optimization, making the optimization both fast and accurate. The hierarchical refinement formalism distinguishes two types of edges in a pose graph; that is, adjacent edges and loop-closure edges. Adjacent edges model the relations between two LiDAR poses whose corresponding LiDAR sweeps are consecutively collected from the same logset; and loop-closure edges model the relations between two LiDAR poses whose corresponding LiDAR sweeps are collected around the same location from different logsets (different vehicles or across time). To handle these two types of edges, the hierarchical refinement formalism includes two steps: (1) optimizing adjacent edges, including a chain of LiDAR poses from a single logset; and (2) optimizing loop-closure edges, including LiDAR poses across logsets; see Figure 4. In the first step, rather than relying simply on aligning LiDAR sweeps, sensor measurements from multiple modalities, including inertial measurement unit (IMU), global positioning system (GPS), odometer, camera and LiDAR, can be fused together to calculate the adjacent edges. Because consecutive LiDAR sweeps have similar LiDAR poses, this step is usually easy and provides extremely high precision. In the second step, the loop-closure edges are calculated by aligning LiDAR sweeps through the ICP algorithm. After these two steps, we then perform the global optimization (6).

Since most edges in a pose graph are adjacent edges, which can be highly optimized through the first step, the hierarchical refinement formalism provides a good initialization for the global optimization. Therefore, the computational cost for optimizing the entire pose graph can be significantly reduced and the robustness of the global optimization can be greatly improved by the hierarchical refinement formalism.

C. *Semantic feature extraction*

The goal of semantic feature extraction is to extract traffic-rule-related semantic features, such as lane geometries, lane connectivities, traffic signs and traffic lights, from the point-cloud map. This component requires both high precision and recall. For example, missing a single traffic light prior in a city-scale HD map can potentially cause serious issues to the perception and motion planning modules, which can severely jeopardize the safety of autonomous driving.

The semantic feature extraction component usually contains two iterative steps. The first step uses machine learning techniques to automatically extract features; and the second step introduces human supervision and quality assurance process to ensure the high precision and recall of the semantic features.

To automatically extract features, standard machine learning techniques are based on convolutional neural networks. The inputs are usually the combination of the LiDAR ground images and the camera images associated with the corresponding real-time LiDAR sweep. A LiDAR ground image renders the BEV-based representation of the point-cloud map obtained in 3D point cloud stitching, where the values of each pixel are the ground height and laser reflectivity of each LiDAR point. The outputs are usually the semantic segmentation of either the LiDAR ground images or the camera images. The networks follow from standard image segmentation architectures.

After obtaining the output, the pixel-wise semantic labels are projected back to the point-cloud map. By fitting the projected 3D points into 3D splines or 3D polygons, the traffic-rule-related semantic feature map can then be obtained. Note that the human-editing outcomes also serve as an important source of training data for automatic feature extraction algorithms, where these two steps therefore form a positive feedback loop to improve the precision and efficiency of HD map production.

D. *Real-world challenges*

There still exist several challenges for the HD map creation.

Point-cloud map with centimeter-level global precision. Global precision can greatly benefit the updating of a city-scale point-cloud map. The changes of the urban appearance usually take place locally. Ideally the map update should focus on the targeted portion of the pose graph; however, a point-cloud map with high local precision but without high global precision cannot freely access the targeted portion from a global aspect and guarantee its precision. In comparison, given a point-cloud map with high global precision, one can focus on updating the targeted portion of the pose graph, thus significantly reducing the scale of computation; however, it is challenging to enforce the global precision to the graph-based SLAM. This is because the global optimization formalism of graph-based SLAM tends to distribute the error of each edge uniformly in the graph. Therefore, even if the GPS observations are accurate, the corresponding LiDAR poses can be misaligned after global optimization. Enforcing centimeter-level global precision of a point-cloud map can be especially challenging in the places where the GPS signal is unavailable, such as in building canyon, tunnel and underground garage.

Automatic semantic feature extraction. Although there exists extensive research on the semantic segmentation based on 3D point clouds and camera images, it is still challenging to automatically extract the lane connectivities in intersections and traffic lights that indicate lane control relations. This is due to limited training labels and complex traffic conditions. Currently, the solution to extracting the complex semantic features such as traffic light to lane control information still relies largely on human supervision, which is both expensive and time-consuming.

IV. 3D Point Cloud Processing for Localization

A. *Overview of localization module*

As introduced in Section I-B, the localization module finds ego position of an autonomous vehicle relative to the reference position in the HD map. It consumes the real-time measurements from multiple sensors, including LiDAR, IMU, GPS, odometer, cameras, as well as the HD map; see Figure 5. Because of the 3D representation of an HD map, the ego position of an autonomous vehicle is a 6DOF pose (translation and rotation), which is a rigid transformation between the map frame and the LiDAR frame. The importance of the localization module to autonomous driving is that it bridges the HD map to the other modules in an autonomy system. For example, by projecting the HD map priors, such as the lane geometries to the LiDAR frame, the autonomous vehicle gains the knowledge of which lane itself drives on and which lanes the detected traffics are on. See a video illustration of the real-time localization through the link[6] and additional illustrations in the supplementary material.

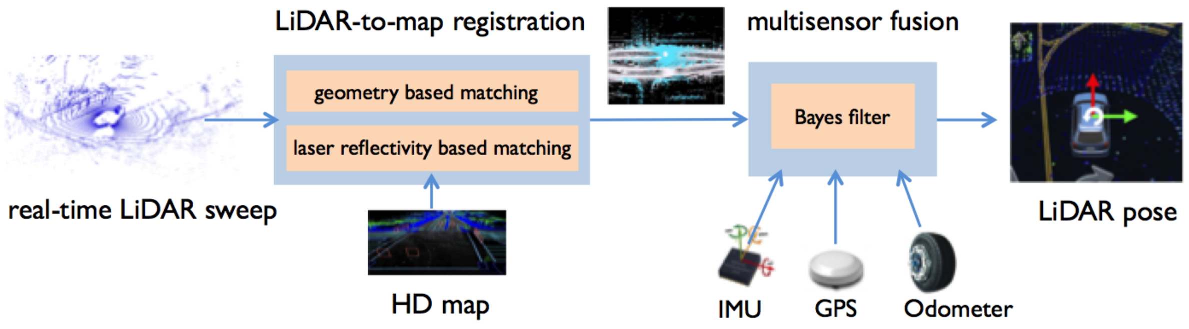


Fig. 5: A standard map-based localization system includes two core components: LiDAR-to-map registration and multisensor fusion. LiDAR-to-map registration uses geometry based matching and laser reflectivity based matching to achieve high precision and recall; and multisensor fusion adopts Bayes filters to merge multiple modalities.

To enable the full autonomous driving, high precision and robustness are the most critical criteria for the performance of localization module. High precision indicates the error of translation should be at the centimeter level and the error of rotation angle should be at the micro-radian level. It allows the traffic detected from 1 kilometer away to be associated to the correct lanes in HD map, and the lane-change intentions of the closer traffic can be predicted by measuring the distance between its wheels to the lane boundaries, which can significantly benefit motion planning and prediction modules; and robustness indicates that the localization module is expected to work in all driving conditions with the changes of illumination, weather, traffic and the condition of roads. Note that although the commercial-grade GPS/IMU unit with real-time kinematics mode has accurate position measurement in open areas, it is not robust enough for autonomous driving because it suffers from the low precision issue in the city due to the multi-path effects.

To achieve these aforementioned criteria, the map-based localization with multi-sensor fusion is the standard approach. As discussed in previous sections, an HD map could be created beforehand and significantly ease the localization. On the contrary, the SLAM-based solution cannot satisfy these criteria.

B. Map-based localization

The basic idea of the map-based localization is to estimate the LIDAR pose by matching a LiDAR sweep to the point-cloud map in an HD map by leveraging the measurements from IMU, GPS, cameras to make pose estimation robust. A map-based localization system usually consists of two components; see Figure 5. The first component is the LiDAR-to-map registration, which computes the LiDAR pose by registering LiDAR sweep to a point-cloud map; The second component is the multisensor fusion, which estimates the final pose from IMU, odometer, GPS, as well as the estimation from the LiDAR-to-map registration.

LiDAR-to-map registration. The LiDAR-to-map registration component is to directly estimate the LiDAR pose by matching the LiDAR sweep to the the point-cloud map. Let $\mathcal{S}$, $\mathcal{S}^{(\mathrm{map})}$ be a real-time LiDAR sweep and the point-cloud map, respectively. The problem of LiDAR-to-map registration can be formulated as

$$\operatorname{argmin}_p \left[\sum_{\mathbf{x}_i \in \mathcal{S}} g\Big(f_p(\mathbf{x}_i), {\mathcal{S}^{(\mathrm{map})}}_{i^*} \Big) \right], \tag{7}$$

where p is the LiDAR pose, $\mathbf{x}_i$ is the ith 3D point in the LiDAR sweep and ${\mathcal{S}^{(\mathrm{map})}}_{i^*}$ is the 3D point in the point-cloud map that is associated with the ith 3D point in the LiDAR sweep. The associated index i^* is usually chosen from the closest point in the Euclidean distance. The function $f_p : R^3 \rightarrow R^3$ is the function that transforms a 3D point $\mathbf{x}_i$ in the LiDAR frame into the map frame based on the LiDAR pose p; and the function $g(\cdot)$ indicates

[6]https://vimeo.com/327949958

a loss function measuring the misalignment between the points from the LiDAR sweep and the HD map. Usually, $g(\cdot)$ takes the forms of the point-to-point, point-to-line, or point-to-plane distance between the associated points in the LiDAR sweep and the point-cloud map.

To solve (7) and achieve high precision and recall, there exist two major approaches.

- *Geometry based matching.* This approach calculates the high precision 6DOF pose by matching the LiDAR sweep to the point-cloud map based on the ICP algorithm [17]. This approach usually works well in heavy traffic and challenging weather conditions, such as snow, because a point-cloud map contains abundant geometry priors for LiDAR sweeps to match with; however, in geometry-degenerated scenes, such as tunnel, bridge, and highway, the ICP calculation could diverge because of the loss of geometric patterns, hence causing bad precision; and
- *Laser reflectivity based matching.* This approach calculates the pose by matching a LiDAR sweep to a point-cloud map based on laser reflectivity signals. The matching can be done in either the dense 2D image matching method or the feature extraction based ICP matching method. For the first method, the laser reflectivity readings of the LiDAR sweep and the point-cloud map are first converted into grey-scale 2D images, following the BEV-based representation (4), and then the pose is calculated by image matching techniques. Note that this method only calculates the x, y, yaw components of the pose. To obtain the 6-DOF pose, the z, roll, pitch components are estimated based on the terrain information in the HD map. For the second method, the region of interest objects, such as lane markers, and poles, are firstly extracted from the LiDAR sweep based on the Laser reflectivity readings [18]. The ICP algorithm can then be used to calculate the LiDAR pose by matching the region of interest objects between a real-time LiDAR sweeps and the priors in the HD map. This approach usually outperforms the geometry based matching in the scenarios of highway and bridge, because those scenarios lack geometry features but have rich laser reflectivity textures on the ground (e.g. dashed lane markers). This approach does not work well in the challenging weather conditions such as heavy rain and snow where the laser reflectivity of the ground will change significantly.

To achieve the best performance, both of these two strategies can simultaneously be used to estimate LiDAR poses; however, LiDAR-to-map registration alone cannot guarantee the 100% precision and recall for the pose estimation over the time. To give an extreme example, if LiDAR is totally occluded by trucks driving side-by-side or front-and-back, the LiDAR-to-map registration component would fail. To handle extreme cases and make the localization module robust, the multisensor fusion component is required.

Multisensor fusion. The multisensor fusion component is to estimate a robust and confident pose from measurements of multiple sensors, including IMU, GPS, odometer, cameras, as well as the poses estimated by the LiDAR-to-map registration module. The standard approach of multisensor fusion is to employ a Bayes-filter formalism, such as Kalman filter, extended Kalman filter, or particle filter. Bayes filters consider an iterative approach to predict and correct the LiDAR pose and other states based on the vehicle motion dynamics and the multisensor readings. In autonomous driving, the states tracked and estimated by Bayes filters usually include motion related states such as pose, velocity, acceleration, etc., and sensor related states such as IMU bias etc.

Bayes filters work in two iterative steps: prediction and correction. In the prediction step, during the gaps between sensor readings, the Bayes filter predicts the states based on the vehicle motion dynamics and the assumed sensor model. For example, by taking the constant acceleration approximation as the vehicle motion dynamics during a short period of time, the evolution of pose, velocity, and acceleration can be predicted by Newton's laws. The IMU bias states can be predicted by assuming that it behaves as white noise.

In the correction step, when receiving a sensor reading or a pose measurement, the Bayes filter corrects the states based on the corresponding observation models. For examples, when an IMU reading is received, the states of acceleration, angular velocities, and the IMU bias are corrected. When a pose measurement is received, the pose state is corrected. Note that the states require the correction because the prediction step is not prefect and there are accumulated errors over time.

C. Real-world challenges

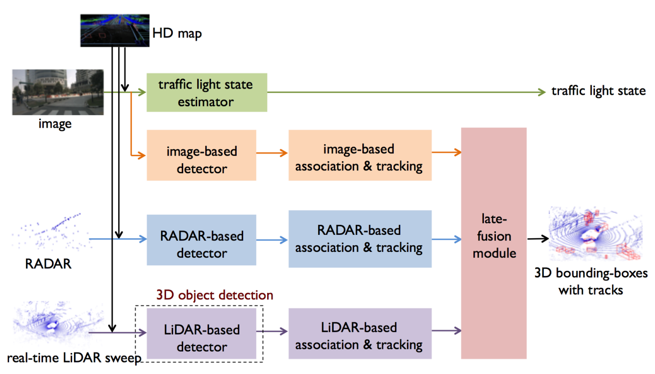
The real-world challenges of the localization module is to work in extreme scenes. For example, when an autonomous vehicle drives through a straight tunnel without dashed lane marker, there are few geometric and texture features, causing the failure of the LiDAR-to-map registration; when an autonomous vehicle is surrounded by large trucks, LiDAR could be totally blocked, also causing the failure of the LiDAR-to-map registration. When the failure of the LiDAR-to-map registration lasts for several minutes, the LiDAR pose estimated by the multisenor fusion component will drift significantly and the localization module will lose the precision.

V. 3D Point Cloud Processing for Perception

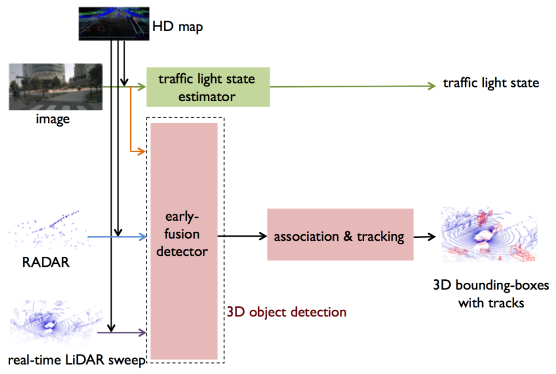
A. *Overview of perception module*

As introduced in Section I-B, the perception module is the visual system of an autonomous vehicle that enables the perception of the surrounding 3D environment. The input of the perception module usually includes the measurements from cameras, LiDAR, RADAR and ultrasound, as well as the ego-motion pose output from the localization module and the priors from the HD map. The outputs of the perception module are typically traffic light states and objects' 3D bounding boxes with tracks.

As discussed in Section I-B, multiple sensing modalities are used to ensure the robustness of the perception module. Depending on the mechanism to fuse those modalities, a perception module can be categorized into late fusion and early fusion. Late fusion fuses modalities in a semantic space, which usually happens in the final step; and early fusion fuses modalities in a feature space, which usually happens in an early or intermediate step. Figure 6 (a) shows a standard framework of a late-fusion-based perception module. To obtain objects' 3D bounding boxes with tracks, a late-fusion-based perception module uses an individual pipeline to handle each sensor input. Each pipeline includes the detection component and the association and tracking component. The detection component finds bounding boxes and the association and tracking component tracks bounding boxes across frames to assign a unique identity for each individual object. A late-fusion module unifies the bounding box information from multiple pipelines and outputs a final 3D bounding-boxes with tracks. In comparison, Figure 6 (b) shows an early-fusion-based perception module. It uses an early-fusion detector to take the outputs from all the sensing modalities and produce all the 3D bounding boxes. It then uses an association and tracking component to associate 3D bounding boxes across frames and assign an identity for each object. To estimate traffic light states, a traffic light state estimator extracts the traffic light regions from images according to the position priors in an HD map and then it uses machine learning techniques to analyze the image and identify the traffic light state.


HD map
image
traffic light state estimator
traffic light state
image-based detector
image-based association & tracking
RADAR
RADAR-based detector
RADAR-based association & tracking
late-fusion module
3D bounding-boxes with tracks
3D object detection
real-time LiDAR sweep
LiDAR-based detector
LiDAR-based association & tracking

(a) Late-fusion-based perception module.



(b) Early-fusion-based perception module.

Fig. 6: A perception module takes multiple sensing modalities and outputs traffic light states and objects' 3D bounding boxes with tracks. Depending on the mechanism to fuse modalities, a perception module is categorized into late fusion, which fuses in a semantic space, or early fusion, which fuses in a feature space.

The late-fusion-based approach is much more mature while the early-fusion-based approach is believed to have a bigger potential [8]. The industry has adopted the late-fusion-based approach for decades because this approach modularizes the tasks and makes each sensor pipeline easy to implement, debug and manage. The early-fusion-based approach carries the spirit of end-to-end learning and enables the mutual promotion of multiple sensing modalities in a high-dimensional feature space; however, there are still significant challenges in this research direction and many companies still use the late-fusion-based approach.

A robust perception module usually includes multiple intermediate components, such as lane detection, 2D object detection, 3D object detection, semantic segmentation and object tracking, to achieve the final goal. Among those components, 3D object detection is particularly interesting and challenging because it needs to handle real-time LiDAR sweeps and can directly produce the 3D bounding boxes for all objects in the scene. This task has drawn much attention recently when combined with the power of deep learning [8]. We next focus on 3D object detection.

B. 3D object detection

The task of 3D object detection is to detect and localize objects in the 3D space with the representation of bounding boxes based on one or multiple sensor measurements. 3D object detection usually outputs 3D bounding boxes of objects, which are the inputs for the component of object association and tracking. Based on the usage of sensor measurements, we can categorize 3D object detection into LiDAR-based detection (see Figure V(a)) and fusion-based detection (see Figure V(b)). Qualitative performances are illustrated in the supplementary material.

LiDAR-based detection. Let $\mathcal{S}$ be a real-time LiDAR sweep. A LiDAR-based detector aims to find all the objects in the sweep; that is,

$$\{\mathbf{o}_i\}_{i=1}^{O} = h(\mathcal{S}), \tag{8}$$

where $\mathbf{o}_i = [\mathbf{y}_i, \mathbf{b}_i]$ is the ith object in the 3D scene with $\mathbf{y}_i$ the object's category, such as vehicle, bikes and pedestrian, and $\mathbf{b}_i$ the corners of bounding box. Now the detection function $h(\cdot)$ is typically implemented with deep-neural-network-based architectures.

The main difference between 2D object detection and 3D object detection is the input representation. Different from a 2D image, a real-time LiDAR sweep could be represented in various ways, leading to corresponding operations in subsequent components. For example, PointRCNN [19] adopts the raw-point-based representation (1) and then uses PointNet with multi-scale sampling and grouping to learn point-wise features; 3D FCN [20] adopts the 3D-voxelization-based representation (2) and uses 3D convolutions to learn voxel-wise features; PIXOR [21] adopts the BEV-based representation (4) and then uses 2D convolutions to learn pixel-wise features; and LaserNet [6] adopt the range-view-based representation (3) and then use 2D convolutions to learn pixel-wise features. Some other methods consider hybrid representations. VoxelNet [22] proposes a voxel-feature-encoding (VFE) layer that combines the advantages of both the raw-point-based representation and the 3D-voxelization-based representation. VFE first groups 3D points according to the 3D voxel they reside in, then uses PointNet to learn point-wise features in each 3D voxel, and finally aggregates point-wise features to obtain voxel-wise feature for each 3D voxel. The benefit of VFE is to convert raw 3D points to the 3D voxelization-based representation and simultaneously learn 3D geometric features in each 3D voxel.

Similarly to 2D objection detection, there are usually two paradigms of 3D object detection: single-stage detection and two-stage detection. The single-stage detection directly estimates bounding boxes, while the two-stage detection first proposes coarse regions that may include objects and then estimates bounding boxes. The single-stage detection directly follows (8). To implement the detection function $h(\cdot)$, a deep-neural-network architecture usually includes two components: a backbone, which extracts deep spatial features, and a header, which outputs the estimations. For a backbone, all these methods use 2D/3D convolutional neural networks with multiscale, pyramidal hierarchical structure. One off-the-shelf backbone structure is feature pyramid networks [23]. A header is usually a multitasking network that handles both category classification and bounding box regression. It is usually small and efficient. Some off-the-shelf header structures are single shot detector [24] and other small convolutional neural networks. The two-stage detection implements the detection function $h(\cdot)$ in two stages; that is,

$$\{r_i\}_{i=1}^{R} = h_1(\mathcal{S}), \tag{9a}$$

$$\{\mathbf{o}_i\}_{i=1}^{O} = h_2(\mathcal{S}, \{r_i\}_{i=1}^{R}), \tag{9b}$$

where r_i is a set of parameters that describes the ith proposed region in the 3D space[7]. The proposal-generation stage (9a) proposes several 3D regions that may include objects inside; and the bounding-box-estimation stage (9b) extracts 3D points from those proposed regions and estimates the precise object positions.

For example, PointRCNN is a recent work that follows the two-stage detection include PointRCNN [19]. In the proposal-generation stage, PointRCNN uses PointNet++ as the backbone and proposes the bin-based localization to propose regions. The bin-based localization first finds the bin associated with the center location of an object

[7]There are multiple approaches to parameterizing a 3D region [19].

and then regresses the residual. In the bounding-box-estimation stage, PointRCNN use canonical transformation to align 3D points in each proposed region and PointNet to estimate the parameters of 3D bounding boxes.

To summarize, the input representation plays a crucial role in the LiDAR-based detection. The raw-point-based representation provides complete point information, but lacks the spatial prior. PointNet has become a standard method to handle this issue and extract features in the raw-point-based representation. The 3D voxelization-based representation and the BEV-based representation are simple and straightforward, but result in a lot of empty voxels and pixels. Feature pyramid networks with sparse convolutions can help address this issue. The range-view-based representation is more compact because the data is represented in the native frame of the sensor, leading to e.ficient processing, and it naturally models the occlusion. But objects at various ranges would have significantly different scales in the range-view-based representation, it usually requires more training data to achieve high performance. VFE introduces hybrid representations that take advantages of both the raw-point-based representations and the 3D voxelization-based representation. The one-stage detection tends to be faster and simpler, and naturally enjoys a high recall, while the two-stage detection tends to achieve higher precision [25].

Fusion-based detection. A real-time LiDAR sweep provides a high-quality 3D representation of a scene; however, the measurements are generally sparse and only return instantaneous locations, making it difficult for LiDAR-based detection approaches to estimate objects' velocities and detect small objects, such as pedestrians, at range. On the other hand, RADAR directly provides motion information and 2D images provides dense measurements. It is possible to naively merge detections from multiple modalities to improve overall robustness, but the benefit of this approach is limited. Following the end-to-end fashion in deep neural networks, early fusion is believed to be a key technique to significantly improve the detection performance; however, it remains an unresolved problem to design an effective early-fusion mechanism. The main challenges are: (1) measurements from each modality come from different measurement spaces. For example, 3D points are sparsely scattered in a continuous 3D space, while images contain dense measurements supported on a 2D lattice; (2) measurements from each modalty are not perfectly synchronized. LiDAR, camera and RADAR capture the scene at their own sampling frequencies; and (3) sensing modalities have unique characteristics. The low-level processing of the sensor data depends on the individual sensor modality, but the high-level fusion needs to consider the characteristics across multiple modalities.

Some existing early-fusion-based detection systems include MV3D [8], F-PointNet [26], PointFusion [27], ContinuousConvolution [28] and LaserNet++ [29]. Each of these works has shown that adding image data can improve detection performance, especially when LiDAR data is sparse; however, the benefit is not substantial and there is no consensus on a system prototype or a basic operation. This makes the industry hard to overturn the previous late-fusion-based approaches.

To summarize, it remains an open problem to design an early-fusion-based detection system. Most designs are based on concatenation of intermediate features from both images and 3D point clouds, allowing the networks to figure out how to merge them. So far, there has been no specific design to handle the unsynchronization issue of multiple sensors, which might be implicitly handled by learning from large-scale training data.

Datasets. High-quality datasets are required to train any of the referenced machine learning models. KITTI [30] is the most commonly used autonomous-driving dataset, which was released in 2012 and has been updated several times since then. Most 3D object detection algorithms are validated on KITTI; however, KITTI is a relatively small dataset and does not provide detailed map information. Several autonomous-driving companies have recently released their datasets, such as nuScenes[8], Argoverse[9], Lyft Level 5 AV dataset[10] and the Waymo open dataset[11].

Evaluation metrics. To evaluate the detection performance, standard evaluation metrics in academia are the precision-recall (PR) curve and average precision (AP); however, there is no standard platform to evaluate the running speed of each model. On the other hand, industry considers more detailed evaluation metrics to check the detection performances. For example, practitioners would check the performances at various ranges, shapes, sizes, appearances, and occlusion levels to get more signals. They would also check the influences on the subsequent modules, such as object tracking, future trajectory prediction, and motion planning to obtain the system-level metrics.

[8]https://www.nuscenes.org/

[9]https://www.argoverse.org/

[10]https://level5.lyft.com/dataset/

[11]https://waymo.com/open

C. Real-world challenges

With the growth of deep learning, the perception module has achieved tremendous improvements. Some practitioners no longer consider it as the technical bottleneck of autonomous driving; however, the perception module is still far from perfect. Here are a series of challenges in the perception module.

High cost. A self-driving vehicle is usually equipped with one or more LiDARs and computing devices, such as GPUs and other specialized processors, which are expensive. The high cost makes it formidable to maintain a scaled fleet of autonomous vehicles. It remains an open problem to exploit information from real-time LiDAR sweeps using low-cost computation;

Tradeoffs between effectiveness and efficiency. A self-driving vehicle should react to its surroundings in real-time. It would be meaningless to pursue a high-precision perception module when it introduces too much latency; however, researchers tend to focus much more on the effectiveness than the efficiency of an algorithm;

Training data deluge. A modern perception module heavily depends on machine learning techniques, which usually need as much training data as possible; however, it takes a lot of time and computational resources to handle large-scale training data. It remains a yet to be resolved problem to effectively choose a representative subset of training data from the entire dataset, which would significantly accelerate the product development;

Long-tail issues. There are countless traffic conditions where large-scale training data cannot cover all the possibilities. It remains an unresolved problem to find and handle corner cases, especially detecting objects that never appear in the training data;

Research conversion. In academia, research tends to design algorithms based on clean and small-scale datasets. It turns out that many effective algorithms work well for those clean and small-scale datasets, but are ineffective on noisy and large-scale datasets. Meanwhile, some algorithms that work well on large-scale datasets do not work well on small-scale datasets [6]. These discrepencies can reduce the usefulness of research results when applied to real-world problems. Industry should consider providing representative datasets and perhaps even a computational evaluation platform that allows people to compare various methods at full industrial scale; and

Evaluation metrics. Objects in a scene have various levels of interactions with an autonomous vehicle. Incorrect estimations of some objects would lead to much bigger consequences than that of other objects; however, the PR curve and AP give uniform weights to all the samples. Additionally, the PR curve and AP do not clearly reflect corner cases, which have only a small sample size; Thus, improving the PR curve and AP do not necessarily lead to a better behavior of an autonomous vehicle. It is often more important to slice the test data and look at the performance over subsets of high-impact cases in addition to overall AP. A standardized simulator could also be developed to provide some system-level metrics.

VI. Summary and Open Issues

The field of autonomous driving is experiencing rapid growth. Many techniques have become relatively mature; however, an ultimate solution for autonomous driving has yet to be determined. At the current stage, LiDAR is an indispensable sensor for building a reliable autonomous vehicle, and advanced techniques for 3D point cloud processing and learning are critical building blocks for autonomous driving. In this article, we surveyed recent developments in the area of 3D point cloud processing and learning and presented their applications to autonomous driving. We described how 3D point cloud processing and learning makes a difference in three important modules in autonomous driving: map creation, localization and perception.

With the rapid development of 3D point cloud processing and learning, the overall performances of the map creation, localization and perception modules in an autonomous system have been significantly improved; however, quite a few challenges remain ahead. Here we briefly mention a few important open issues from a big picture perspective[12].

How should we make processing and learning algorithms scalable and efficient? Now we are still in the developing phase and autonomous vehicles are tested in a limited number of canonical routes or over a small area. In the near future, autonomous vehicles might be tested in a city/country scale, which needs a city/country-scale HD map. This requires scalable algorithms to create and update HD maps. Now an autonomous vehicle is usually

[12]Adversarial attack is also a potential issue; however, it is not one of the most critical technique challenges at the current stage because the current techniques are far away from the performance level where adversarial attack could be a major concern. Meanwhile, we need to consider adversarial attack, so that we can avoid optimizing a solution that may have a non-considered major issue at the end.

equipped with a 64-line LiDAR, which still produces relatively sparse point clouds. In the near future, LiDAR might have many more lines and produce much denser point clouds. This requires more efficient algorithms to achieve LiDAR-to-map localization and 3D object detection in the real-time;

How should we make processing and learning algorithms robust enough to handle corner cases? We can collect large amounts of real-world sensor data and generate large amounts of simulated sensor data, but we need to deliberately select the most representative data to improve the generality of the algorithms. At the same time, one has to face the fact that all learning algorithms depend on training data, which can never cover all the possibilities. To address this issue, one key research area is to improve the uncertainty estimation of an algorithm, because this allows a system to react conservatively when the learned components are not confident. This requires reasoning both about the known uncertainty from the training data and also the more challenging uncertainty from cases that are not covered by the training data;

How should we develop processing and learning algorithms with a faster iteration speed? We want more data and more complicated algorithms to achieve better performance for autonomous driving; meanwhile, we want efficient and practical algorithms to accelerate product development, which is also critical. Practitioners in industry should collaborate closely with researchers in academia to increase the research conversion rate; and

How should we evaluate processing and learning algorithms? Currently most processing and learning algorithms are evaluated on specific model-level metrics to meet the criteria of the corresponding tasks; however, these model-level metrics often do not fully correlate with system-level metrics that reflect the overall behavior. Along these same lines, the research community often focuses on improving the average performance, but there needs to be an increased focus on improving the rare long-tail cases that are really critical for a real-world system.

References

[1] A. Taeihagh and H. Si Min Lim, "Governing autonomous vehicles: emerging responses for safety, liability, privacy, cybersecurity, and industry risks," *Transport Reviews*, vol. 39, no. 1, pp. 103–128, Jan. 2019.

[2] National Research Council, "Technology development for army unmanned ground vehicles," 2002.

[3] C. Badue, R. Guidolini, R. Vivacqua Carneiro, P. Azevedo, V. Brito Cardoso, A. Forechi, L. Ferreira Reis Jesus, R. Ferreira Berriel, T. Meireles Paixo, F. Mutz, T. Oliveira-Santos, and A. Ferreira De Souza, "Self-driving cars: A survey," arXiv:1901.04407 [cs.RO], Jan. 2019.

[4] M. Bansal, A. Krizhevsky, and A. S. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," *CoRR*, vol. abs/1812.03079, 2018.

[5] C. Urmson, J. Anhalt, D. Bagnell, C. R. Baker, R. Bittner, M. N. Clark, J. M. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. M. Peterson, B. Pilnick, R. Rajkumar, P. E. Rybski, B. Salesky, Y-W. Seo, S. Singh, J. M. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," in *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base, Victorville, California, USA*, 2009, pp. 1–59.

[6] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, "Lasernet: An efficient probabilistic 3D object detector for autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, 2019.

[7] C. Ruizhongtai Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, 2017, pp. 77–85.

[8] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, 2017, pp. 6526–6534.

[9] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[10] X-F. Hana, J. S. Jin, J. Xie, M-J. Wang, and W. Jiang, "A comprehensive review of 3D point cloud descriptors," *arXiv preprint arXiv:1802.02297*, 2018.

[11] J. Peng and C.-C. Jay Kuo, "Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition," *ACM Trans. Graph. Proceedings of ACM SIGGRAPH*, vol. 24, no. 3, pp. 609–616, Jul. 2005.

[12] S. Chen, D. Tian, C. Feng, A. Vetro, and J. Kovačević, "Fast resampling of three-dimensional point clouds via graphs," *IEEE Trans. Signal Processing*, vol. 66, no. 3, pp. 666–681, 2018.

[13] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, November 2019.

[14] G. Li, M. Müller, A. K. Thabet, and B. Ghanem, "DeepGCNs: Can GCNs go as deep as CNNs?," in *ICCV*, Seoul, South Korea, Oct. 2019.

[15] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intell. Transport. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, 2010.

[16] D. Droeschel and S. Behnke, "Efficient continuous-time SLAM for 3D lidar-based online mapping," in *2018 IEEE International Conference on Robotics and Automation, ICRA, 2018, Brisbane, Australia, May 21-25, 2018*, 2018, pp. 1–9.

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

[18] A. Y. Hata and D. F. Wolf, "Road marking detection using LIDAR reflective intensity data and its application to vehicle localization," in *17th International IEEE Conference on Intelligent Transportation Systems, ITSC 2014, Qingdao, China, October 8-11, 2014*, 2014, pp. 584–589.

[19] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, Long Beach, CA, June 2019.

[20] B. Li, "3D fully convolutional network for vehicle detection in point cloud," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, 2017, pp. 1513–1518.

[21] B. Yang, W. Luo, and R. Urtasun, "PIXOR: real-time 3D object detection from point clouds," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, 2018, pp. 7652–7660.

[22] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, Salt Lake City, UT, USA, June 2018, pp. 4490–4499.

[23] T-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, 2017, pp. 936–944.

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C-Y. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, 2016, pp. 21–37.

[25] T-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 2999–3007.

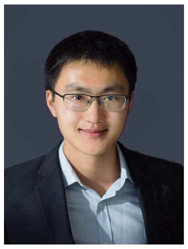
[26] C. Ruizhongtai Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum pointnets for 3D object detection from RGB-D data," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, 2018, pp. 918–927.

[27] D. Xu, D. Anguelov, and A. Jain, "PointFusion: Deep sensor fusion for 3D bounding box estimation," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.* 2018, pp. 244–253, IEEE Computer Society.

[28] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep continuous fusion for multi-sensor 3D object detection," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, 2018, pp. 663–678.

[29] G. P. Meyer, J. Charland, D. Hegde, A. Laddha, and C. Vallespi-Gonzalez, "Sensor fusion for joint 3D object detection and semantic segmentation," *CoRR*, vol. abs/1904.11466, 2019.

[30] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recogn.*, Providence, RI, June 2012, pp. 3354–3361.



Siheng Chen is a research scientist at Mitsubishi Electric Research Laboratories (MERL). Before that, he was an autonomy engineer at Uber Advanced Technologies Group, working on the perception and prediction systems of self-driving cars. Before joining Uber, he was a postdoctoral research associate at Carnegie Mellon University. Chen received the doctorate in Electrical and Computer Engineering from Carnegie Mellon University in 2016, where he also received two masters degrees in Electrical and Computer Engineering and Machine Learning, respectively. He received his bachelor's degree in Electronics Engineering in 2011 from Beijing Institute of Technology, China. Chen was the recipient of the 2018 IEEE Signal Processing Society Young Author Best Paper Award. His coauthored paper received the Best Student Paper Award at IEEE GlobalSIP 2018. His research interests include graph signal processing, graph neural networks and 3D computer vision.

Baoan Liu is the founder and chief technology officer of Precivision Technologies, Inc. (acquired by DeepMap Inc. in 2019), where he leads the technologies and products development of high-definition mapping and high-precision vehicle localization solutions for autonomous driving. Dr. Liu received the doctorate in Mechanical Engineering from Carnegie Mellon University in 2016. He received his bachelor's degrees in Electrical and Computer Engineering and Applied Physics in 2011 from Shanghai Jiao Tong University, China.

Chen Feng earned his Bachelor's degree in geospatial engineering from Wuhan University in China. Then he went to the University of Michigan at Ann Arbor and earned a master's degree in electrical engineering and a Ph.D. in civil engineering in 2015, where he studied robotic vision and learning and attempted to apply them in civil engineering. After graduation, he became a research scientist in the computer vision group at the Mitsubishi Electric Research Labs (MERL), focusing on visual SLAM and deep learning. In 2018 August, he became an assistant professor jointly in the Department of Mechanical and Aerospace Engineering and the Department of Civil and Urban Engineering in New York University Tandon School of Engineering, where his lab AI4CE (A-I-force) aims to advance robotic vision and machine learning through multidisciplinary use-inspired research that originates from civil/mechanical engineering domains. More information can be found at https://ai4ce.github.io/.

Carlos Vallespi-Gonzalez is a Senior Staff Engineer and Technical Lead Manager at Uber Advanced Technologies Group, leading the research and development of perception algorithms deployed in the Uber self-driving cars. Previously, he worked over 10 years at the National Robotics Engineering Center in the automation of agricultural machinery, including the development and deployment of fully autonomous tractors in orange orchards in Florida. He graduated with honors from La Salle School of Engineering with a degree in Software Engineering and received a M.Sc. in Robotics from Carnegie Mellon. He has authored and co-authored over 30 patents as well as several publications in major Computer Vision conferences. His research interests are in the fields of Machine Learning and Computer Vision.

Carl Wellington is a Director of Engineering at Uber Advanced Technologies Group. He was one of the founding members of Uber ATG when it was started in 2015 and currently leads the Perception and Prediction teams of Uber's self-driving effort. Before that he spent 10 years as an applied researcher at Carnegie Mellon University's National Robotics Engineering Center (NREC), developing perception systems for autonomous agricultural and military vehicles that ranged in size from small utility carts to large tractors and multi-ton unmanned ground vehicles. His work at NREC included several large scale field trials and multiple computer vision systems that have since become commercial products. Dr. Wellington received his doctorate in Robotics from Carnegie Mellon University in 2005 and his research interests are in machine learning for robotic perception.