Regular Article

LiDAR Road-Atlas: An Efficient Map Representation for General 3D Urban Environment

Banghe Wu[1], Chengzhong Xu[2] and Hui Kong[3]

[1]School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, Jiangsu, China

[2]The State Key Laboratory of Internet of Things for Smart City (SKL-IOTSC), Department of Computer Science, University of Macau, Macau

[3]The State Key Laboratory of Internet of Things for Smart City (SKL-IOTSC), Department of Electromechanical Engineering (EME), University of Macau, Macau

Abstract: In this work, we propose the LiDAR Road-Atlas, a compact and efficient 3D map representation, for autonomous robot or vehicle navigation in a general urban environment. The LiDAR Road-Atlas can be generated by an online mapping framework which incrementally merges local 2D occupancy grid maps (2D-OGMs). Specifically, the contributions of our method are threefold. First, we solve the challenging problem of creating local 2D-OGMs in nonstructured urban scenes based on a real-time delimitation of traversable and curb regions in a LiDAR point cloud. Second, we achieve accurate 3D mapping in multiple-layer urban road scenarios by a probabilistic fusion scheme. Third, we achieve a very efficient 3D map representation of a general environment thanks to the automatic local-OGM-induced traversable-region labeling and a sparse probabilistic local point-cloud encoding. Given the LiDAR Road-Atlas, one can achieve accurate vehicle localization, path planning, and some other tasks. Our map representation is insensitive to dynamic objects which can be filtered out in the resulting map based on a probabilistic fusion. Empirically, we compare our map representation with a couple of popular map representations in robotics society, and our map representation is more favorable in terms of efficiency, scalability, and compactness. Additionally, we also evaluate localization performance given the LiDAR Road-Atlas representations on two public datasets. With a 16-channel LiDAR sensor, our method achieves an average global localization error of 0.26 m (translation) and 1.07° (rotation) on the Apollo dataset, and 0.89 m (translation) and 1.29° (rotation) on the MulRan dataset, respectively, at 10 Hz, which validates its promising performance. The code for this work is open-sourced at https://github.com/IMRL/Lidar-road-atlas.

Keywords: mapping, localization

Received: 15 April 2022; revised: 16 February 2023; accepted: 26 February 2023; published: 24 March 2023.

Correspondence: Hui Kong, The State Key Laboratory of Internet of Things for Smart City (SKL-IOTSC), Department of Electromechanical Engineering (EME), University of Macau, Macau., Email: huikong@um.edu.mo

DOI: https://doi.org/10.55417/fr.2023014

1. Introduction

Nowadays, autonomous online 3D mapping plays a key role in robot exploration of unknown environments, and can also act as the "teaching" role in most "teach-and-repeat" paradigms for robot navigation or autonomous driving applications. As exploration goes on, the map size (either runtime memory usage or hard-drive storage) can scale up quickly. Likewise, 3D mapping of large general urban environments is also important for commercialized autonomous vehicles, where the resulting 3D high-definition (HD) map can be reused for vehicle localization and path planning. When applied at large scale, the map's storage size is one key factor that has to be considered as well. Therefore, a parsimonious map representation is highly desirable. In this paper, we propose the LiDAR Road-Atlas, a compact and efficient map representation of a general 3D urban environment, for autonomous robot or vehicle navigation. It will be generated by an online mapping framework based on incrementally merging 2D local occupancy grid maps.

In the robotics society, one classical and efficient way to represent the environment is the 2D occupancy grid map (2D-OGM) (Elfes, 1987), where the cells in the grid are labeled as drivable, unknown, or occupied. A 2D occupancy map usually suffices for robot navigation in a structured outdoor environment. For navigation in nonstructured outdoor environments, where the ground is not purely flat (Figure 1) or where overhanging objects (e.g., tree branches, tunnels, low roofs, etc.) are very common, a 2D representation of the environment is insufficient because 2D-OGM cannot fully represent the actual environment. In such outdoor scenarios, 2D-OGM cannot be accurately created in general because it is not easy to reliably find the boundaries between known (traversable region) and unknown (occluded region) areas due to the existence of low obstacle regions. Using an underrepresented map can be totally erroneous in path planning, let alone vehicle localization. Therefore, a more suitable alternative for outdoor navigation would be a map representation where the environment could be well described. Over the years, a few more expressive 3D map representation methods than 2D-OGM have been proposed for autonomous vehicle navigation (Funk et al., 2021; Hornung et al., 2013; Pfaff et al., 2007; Souza et al., 2013; Souza and Gonçalves, 2016; Steinbrücker et al., 2014; Triebel et al., 2006; Yue et al., 2016), and these methods can be broadly categorized as elevation maps or full 3D point-cloud maps. The map's scale in robotics society is not necessarily very large considering that robots only operate in an area of relatively limited size.

Figure 1. Examples of nonflat urban infrastructures including underpass, overpass, multiple-layer garage, steep road scenes, etc.

In the autonomous driving society, unlike the map representations in the robotics field, HD maps can be extremely large, which can be up to a city scale. Therefore, highly efficient and compact map representations are desirable for autonomous vehicle navigation. The recent road-based mapping methods for autonomous driving in urban environments (Herb et al., 2019; Jeong et al., 2017; Lu et al., 2017; Qin et al., 2020, 2021; Ranganathan et al., 2013; Rehder and Albrecht, 2015; Schreiber et al., 2013) belong to this line of research, where varieties of semantic 2D and 3D features are segmented from road region. The 2D elements are basically on the road surface, including lane markers and curb lines, zebra crossings, all kinds of ground arrow signs, etc. The 3D structures are mostly regular vertical objects, such as traffic lights, traffic signs, light poles, etc. Generally, deep-learning methods are used to segment these semantic image regions. Then, high-precision GPS/IMU and multiray LiDAR sensors are combined to build a prior urban map. Thereafter, the locations of the extracted semantic features in images are transformed into the world coordinates and anchored to the created prior map. Given the created HD map, one can apply it to accurately localize a vehicle with a single camera (aided by a low-cost GPS receiver and wheel odometers for coarse localization in general), where the image from the camera is segmented by a deep-learning model to find the road-scene semantic features, and thereafter a fine localization is achieved by feature association and PnP (perspective-n-point) algorithms.

By comparing the above two types of HD maps in the robotics domain and autonomous-driving industries, respectively, we find that neither of them can well meet the requirement for creating HD maps of general urban environments, where semantic road markers or regular man-made road-scene infrastructures (e.g., traffic light poles) might be absent, and there might also exist multiple-level road surfaces (e.g., underpass and multiple overpass). In the robotics field, the elevation-based methods cannot fully represent complex scenes and the full 3D point-cloud-based maps generally consume too much memory or storage space. In autonomous driving, the existing road-based mapping methods heavily rely on semantic road features. Although these features are salient and common in highway road scenes, they are not necessarily present universally in general urban environments, e.g., various residential communities, university campuses, industrial parks, etc. Thus, one has to collect new images of semantic road-scene features which are specific to each individual scene before learning a new semantic segmentation deep-learning model, which is a heavy workload. In addition, these methods cannot guarantee to build maps in an online mode, and some of them also need expensive high-precision GPS/IMU sensors.

Given the problems of the above map representations, it is necessary to think of an efficient map which is parsimonious in memory consumption and hard-drive storage, and highly descriptive for a complex environment as well. Summing up all the above thoughts, the main contributions of our mapping work can be as follows:

1. We propose an efficient online mapping method which can be applied to general 3D urban environments to create an HD map for autonomous vehicle navigation. The result of mapping is a parsimonious map representation by embedding sparse local point-cloud encodings and local-OGM-induced traversable-region labeling.
2. We can handle the challenging problem of creating local 2D-OGMs in nonstructured urban scenes based on a real-time delimitation of traversable and curb regions in a LiDAR point cloud.
3. We can achieve accurate 3D mapping in multiple-layer urban road scenarios and solve the map update problem of the existing multiple-level elevation maps caused by dynamic objects present in the mapping process by a probabilistic fusion scheme.

We have compared the proposed map representation with the popular OctoMap map representation (Hornung et al., 2013). Empirically, the resulting map by the proposed map representation is much smaller than that of the OctoMap representation. Based on the runtime memory coverage, our map representation can afford the online large-scale map creation while the OctoMap cannot. We have also conducted extensive experiments on online map updating and map-based localization, by which we have shown the accuracy and efficiency of the proposed map representation.

In contrast to the existing mapping methods, our method belongs to the more broad category of traversable-region-based mapping algorithms, where the traversable region is not limited to structured urban roads where the man-made semantic road elements (road markings and signs, traffic lights, etc.) are available. Actually, our method can be applied to more general natural scenes instead of just urban environments. Our method only uses a multiple-channel LiDAR sensor (VLP-16) to generate the LiDAR Road-Atlas. Note that we do not use cameras in this work although we occasionally show images in the paper, and the purpose is to tell what the evaluation scenarios look like. Because our mapping is an online process, it can also be easily extended to some robotic exploratory tasks in an unknown environment.

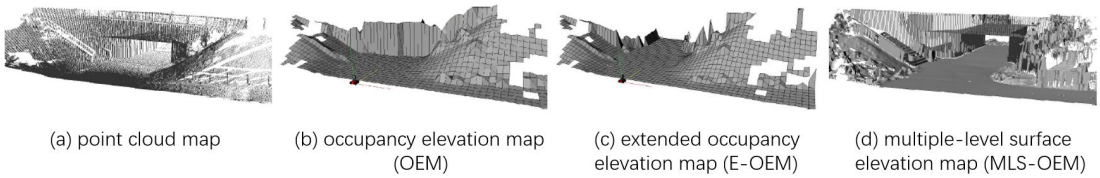
2. Related works

2.1. Representation of Traversable Region

In representing traversable regions, some research (Herb et al., 2019; Jeong et al., 2017; Qin et al., 2020, 2021; Rehder and Albrecht, 2015) focused on the process of building road maps. Rehder and Albrecht (2015) detected lanes in images and exploited odometry information to generate local grid maps, and the pose between local grid maps is further optimized and stitched. Jeong et al. (2017) first differentiated road markings, and more informative class information can be used to avoid ambiguity on matching. Pose graph optimization based on loop closure is performed to eliminate drift for global pose consistency. Qin et al. (2020) utilized road markers to build a semantic map for underground parking lots with application to autonomous parking based on only cameras. Herb et al. (2019) proposed a crowdsourced solution to build a semantic map. However, it was difficult to apply because its computational complexity is generally high in matching intersession features. Very recently, Qin et al. (2021) proposed a lightweight semantic map representation for assisting camera-based vehicle localization in a highway structured environment. In utilizing the created road HD map, Schreiber et al. (2013) proposed to detect curbs and lanes, and then match the structure of these elements with an HD map. Ranganathan et al. (2013) utilized detected corners on road markers for localization. Lu et al. (2017) formulated a nonlinear optimization to match road markers with the map. The vehicle odometry and epipolar geometry were also taken into consideration to estimate the six-degree-of-freedom (6-DoF) camera pose. Morales et al. (2009) proposed an autonomous system for robot navigation in outdoor cluttered pedestrian walkways. As a part of this system, a module of driving path detection is given for safe navigation of the robot. Different from our method, this method applied a cascaded height filter to find a traversable region in each individual scan, and then detected curb edge points to construct a map for navigation. The navigation map is still two-dimensional because the planned waypoints lie in a planar map.

2.2. Representation of 3D Space Occupancy

The occupancy elevation map (OEM) is also a frequently used map representation method to deal with the problem of underrepresenting the 3D environment in 2D-OGM (Bares et al., 1989; Kweon et al., 1989; Souza and Gonçalves, 2016; Souza et al., 2013; Triebel et al., 2006), as shown in Figure 2. In OEM, the height of the surface is stored in the corresponding cell of the discretized



(a) point cloud map (b) occupancy elevation map (OEM) (c) extended occupancy elevation map (E-OEM) (d) multiple-level surface elevation map (MLS-OEM)

Figure 2. Four types of 3D occupancy grid map representations (courtesy of 33). Refer to our Introduction for more details on these methods.

grid. While the OEM provides an efficient map representation, it lacks the ability to fully represent vertical structures or even multiple levels of horizontal planes (e.g., underpass/overpass bridges in Figure 2(b)). The extended occupancy elevation map (E-OEM)(Pfaff and Burgard, 2005) can alleviate the problem of OEM to some extent, and allows the handling of overhanging objects (e.g., tree branches) or representing the underpass. But it cannot fully represent the environments where there is more than one level of horizontal surface (e.g., the overpass bridges in Figure 2(c) or multiple-level parking garages). To address this issue, Triebel et al. propose multilevel surface occupancy elevation map (MLS-OEM) data structures (Triebel et al., 2006), where a list of surfaces is stored in each cell of the discrete grid (e.g., Figure 2(d)). Additionally, they use intervals to represent vertical structures. The MLS-OEM roughly provides a 3D representation without the complexity of a full 3D point-cloud map, enabling a mobile robot to model complex environments. However, the MLS-OEM representation is designed for mapping static environments, and it is not capable of handling the influence of dynamic objects during the mapping process. In contrast, our method can not only represent the multiple-level surface elevation but it can also deal with the influence of dynamic objects in the mapping process.

Another widely used map representation is the OctoMap (Hornung et al., 2013), where each point can be stored and indexed in an Octree data structure. The indexing of each point is fast due to the tree structure. The OctoMap map is able to model almost any kind of environment without prior assumptions on it, and it models occupied areas as well as free space. Unknown areas of the environment can also be implicitly encoded in the map. Although OctoMap has high representative capability, its memory usage and hard-drive storage are also quite high. The normal distribution transform traversability mapping (NDT-TM) representation (Ahtiainen et al., 2017) for outdoor environments is able to classify sparsely vegetated areas as traversable without compromising accuracy on other terrain types. The NDT-TM representation is proposed by exploiting 3D LIDAR sensor data to incrementally expand normal distributions transform occupancy maps (NDT-OMs) (Saarinen et al., 2013), which is a 3D spatial model based on a regular grid that concurrently estimates both the occupancy and the shape distribution in each cell. In addition to geometrical information, the NDT-OM representation is augmented with statistical data of the permeability and reflectivity of each cell of the NDT-OM map. Using these additional features, a support-vector machine classifier is trained to discriminate between traversable and nondrivable areas of the NDT-TM maps. In spite of advantages over 2D grid mapping, the construction of full 3D occupancy grid models (Ahtiainen et al., 2017; Hornung et al., 2013; Saarinen et al., 2013) typically has excessively high demands for runtime memory and hard-drive storage during its online application to a mobile robot in large-scale outdoor scenarios or when fine map resolution is necessary.

The remainder of the paper is organized as follows. We introduce our method in Section III, and the implementation details and experimental results are given in Section IV. We discuss our method in terms of its scalable, extendable, and applicable properties in Section V. The conclusions are drawn in the last section. Four videos which demonstrate the creation of the LiDAR Road-Atlas for four different scenes are available at https://youtu.be/9wXiZzGau3w, https://youtu.be/wjkf8snJzRA, https://youtu.be/4tfKbb47Ajc, and https://youtu.be/UMZLE8CMGtk.

3. Our Method

Our work aims at building a 3D global sparse occupancy grid map with labeled traversable regions and static obstacle (e.g., trees and buildings) codings embedded inside. To do that, we estimate the LiDAR's 6D pose based on a LiDAR-based simultaneous localization and mapping (SLAM) or odometry method. At each LiDAR's keyframe location, we build a local occupancy grid map and align it to the estimated 6D pose (Figure 8). A stitching operation is given to fuse two overlapping local occupancy grid maps when necessary. We consider not only common outdoor scenes but also the scenarios where it is necessary to build multiple layers of a local occupancy grid map, e.g., the crossovers and multiple-layer parking lots shown in Figure 1.

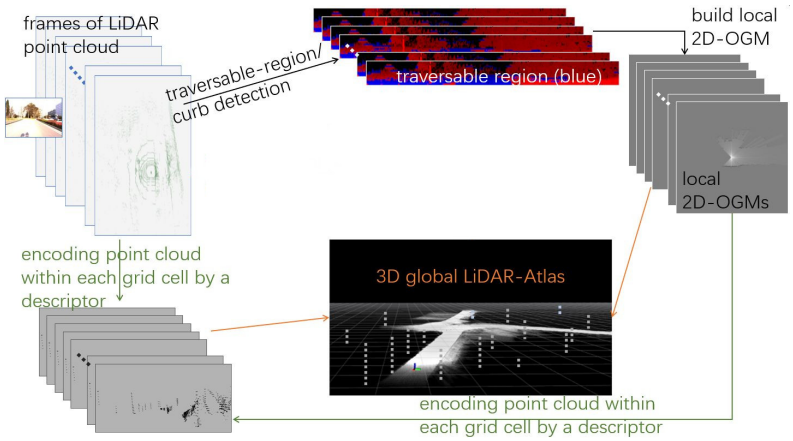


Figure 3. Overview of the process of generating 3D global LiDAR-Atlas. Basically, it consists of three major components to extract intermediate results that are aggregated to form the 3D LiDAR-Atlas, i.e., traversable-region (curb) detection for building local 2D-OGM, encoding of the point cloud's geometric information within each cell of the local 2D-OGM.

Our method takes in point-cloud data from a 16-channel LiDAR sensor and estimates the relative pose between two consecutive frames. An overview of our method is shown in Figure 3, where the point cloud is processed and automatically labeled, with each point being classified as traversable regions or obstacles. Thereafter, the processed point cloud is used to construct a local grid map. Meanwhile, for each frame of the point cloud, spare descriptors of local geometric structures are extracted and embedded into the global 3D LiDAR Road-Atlas along with the labeled traversable region in each local grid map.

3.1. Detecting Traversable Area and Obstacles in Each LiDAR Point-Cloud Keyframe

Accurate detection of traversable regions is crucial for path planning and route following in the "repeat" stage of the "teach-and-repeat" navigation paradigm. Especially when localization in the map is not robust enough, a reliable traversable road extraction method can ensure the robot stays and runs in the path. Traversable region labeling is strongly relevant to detection of obstacles, ranging from nontrivial ones such as vehicles, walls, overhanging tree branches, and overpass bridges to trivial ones such as curbs or ditches.

Our algorithm is designed to automatically detect a traversable region during the map-building process. The online detection is also valuable to planning paths for robotics exploration tasks. In general, one usually uses the Random Sample Consensus (RANSAC) algorithm (Fischler and Bolles., 1981) to find a dominant plane of the scene. However, there usually exist multiple planes of similar size in a scene. When applying a single RANSAC, small obstacles (e.g., curbs or pits on road) are usually overlooked if a large tolerance threshold is adopted. As a result, it is neither reasonable nor robust to apply a single RANSAC model to detect traversable regions.

Although a single RANSAC does not work well in a whole point-cloud frame, it can do a good job locally. Therefore, one could apply multiple RANSAC models sequentially to the whole point cloud, where a partial traversable region is detected and removed from the point cloud by a different RANSAC model in an iterative way. Although this multiple-RANSAC way could generally get better results than a global RANSAC model, it is not trivial to determine the exact number of iterations necessary for a good segmentation.

We propose a new multiple-RANSAC-based method to detect traversable and obstacle regions from a frame of point cloud. Specifically, we divide the point cloud P into overlapping local sectors A_i, and perform multiple RANSAC models on these sectors, respectively, to separate traversable areas from obstacle regions. Note that the smallest obstacles by our method can range from curbs

Algorithm 1. Detecting traversable area and obstacles.

Input: whole point cloud P, local point cloud A_i
Output: traversable-region G, obstacles O

```
foreach A_i do
    G_local = RANSAC(A_i)                    ▷ N is the normal vector of the plane
    if size(G_local) ≥ size(A_i)/2 && check_normal(N) then
        G = G ∪ G_local
    end
end
O = P − G
```

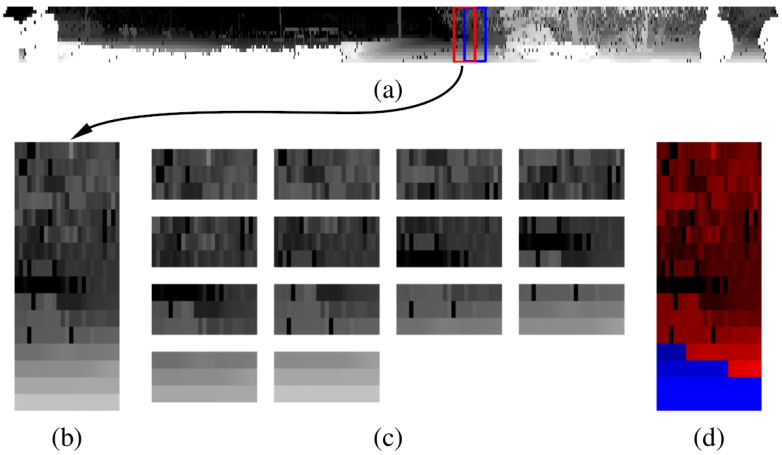


Figure 4. Splitting a whole LiDAR frame into many overlapping regions. (a) One VLP-16 LiDAR image (16x1800) is shown. Note that we have adjusted the scale of the LiDAR image for a better view. The LiDAR image is split into many overlapping local regions horizontally and vertically, where two neighboring horizontal overlapping sectors (red and blue boxes) have a size of 16x50 and a step size of 25. Likewise, let us forget about the disproportionate ratio of the box's dimension. (b) The enlarged red box. (c) All overlapping local areas extracted from (b) have a step size of 1, and each local area has a size of 3x50. (d) The detected traversable region (blue) and obstacle area (red) based on the proposed method.

to small ditches or stones on the road. Figure 4 illustrates the way of splitting a whole LiDAR point-cloud frame into many overlapping regions. Based on the splitting, the multiple-RANSAC traversable region and obstacle detection method is given in Algorithm 1.

Since RANSAC cannot distinguish between the ground and the wall, we define *check_normal* to check the normal vector of the plane obtained by RANSAC to remove too steep planes. The return value of *check_normal* is only true when the angle between the normal vector and the z axis is less than the threshold. The threshold depends on the vehicle. We set it to 0.4 rad in our experiments.

Generally, there are two possibilities in fitting a plane in a split local region. One is that the whole local region can be perfectly fitted by a plane model, e.g., the left dark rectangle shown in Figures 5(a) and 5(b). The other is that part of the split local area can be fitted by a plane model, e.g., the red rectangle and the right dark rectangle shown in Figure 5(b). For such local regions, the plane is detected as the part which covers more than 50% of the whole local region. In Figure 5(a), we show an alternative local-region splitting scheme where there is no overlap between two neighboring local regions, for which the segmentation of small obstacles is worse than that in the splitting case with overlap. Therefore, the local-region splitting method with overlap is our choice. It deserves pointing out that the splitting with overlap is crucial for reliably detecting small objects such as curbs, which can guarantee creating an accurate local occupancy grid map, as shown in Figure 6.

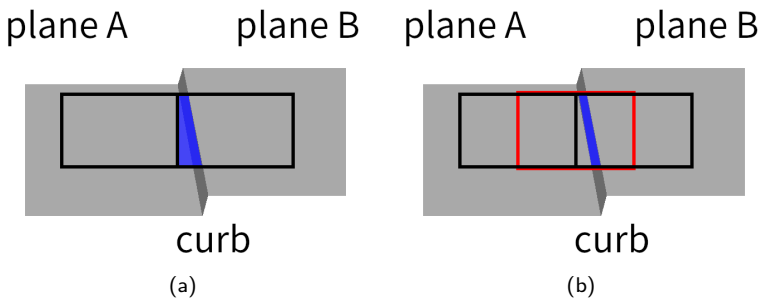


Figure 5. A demonstration of the role of our "splitting with overlap" scheme on segmentation of a curb-like region. (a) Two split regions without "overlap" between them. (b) Three split regions with "overlap" between two neighboring local regions. Based on Algorithm 1, the total traversable region in a point-cloud frame is composed of many smaller traversable planes which are detected in the local regions. The rest corresponds to the obstacle area. In (a), part of the road plane (blue region) is detected as the curb obstacle. In (b), we can deal with this issue.

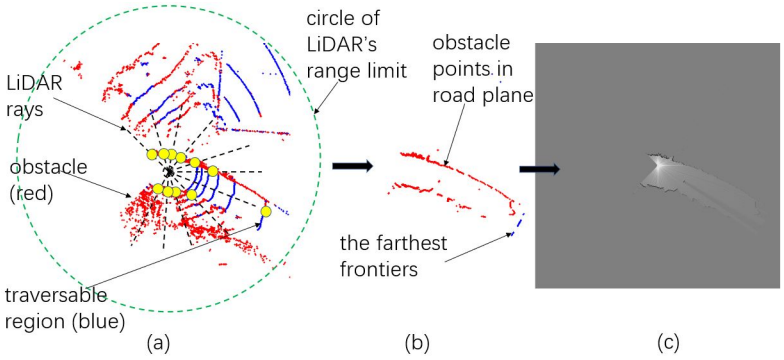


Figure 6. (a) Projecting the automatically labeled 3D points (based on the method proposed in the last section) onto the 2D ground plane, and applying ray tracing to find the points corresponding to small obstacles (curb points). (b) The obtained obstacle points (red) corresponding to curbs and farthest frontier points (blue). (c) The obtained 2D occupancy grid map by using the points in (b) based on the method in Hess et al. (2016) and Thrun et al. (2005).

In our implementation, we split a whole point-cloud frame into local areas using the LiDAR image representation (Chen et al., 2017). The raw output data of a multiple-channel LiDAR are actually based on a spherical coordinate system, including the yaw angle θ of each laser beam and measurement distance d. The pitch angle ϕ of each laser beam can be loaded from the calibration file. Actually, there are a couple of ways to divide a point cloud to meet the overlap condition. However, the raw point-cloud representation is sparse, and it is not straightforward to directly perform neighborhood calculation and extract the neighborhood that meets the overlap condition. That is why we choose the LiDAR image representation in traversable region detection.

3.2. Creating 2D Local Occupancy Grid Map from Each LiDAR Point-Cloud Keyframe

Conventionally, the 2D-OGM is obtained by a 2D LiDAR sensor. It can produce a clear boundary representing dynamic or static obstacles in indoor environments or 2D structured outdoor environments, and thus a 2D-OGM can well represent the environment. In such an environment, vehicle localization and path planning can be fulfilled with ease in a 2D-OGM generally. In a more general 3D urban environment, 2D-OGM cannot be reliably created if obstacles are not well labeled. In our work, we can deal with this issue by the traversable and curb-region labeling procedure as described in the last section. In the meantime, we can also take the advantage of 2D-OGM over its 3D counterpart, i.e., efficiency and small memory usage.

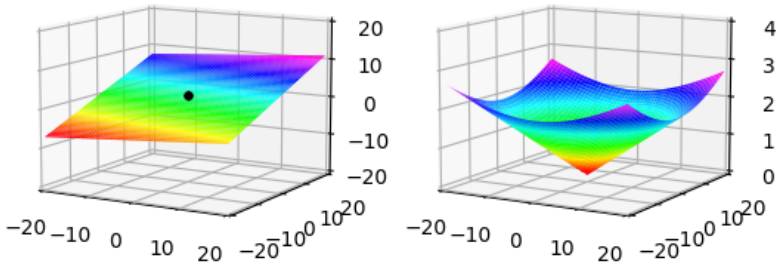


Figure 7. An example of the distribution of the height of a local grid. The left image shows the mean values μ at each cell location, and the black point in the center corresponds to the location of the robot. The right image shows the standard deviation σ at each cell location, where $\sigma = \tan 5^\circ \times d + 0.1$.

Specifically, we assume that the local ground area around the vehicle can be approximated as a 2D plane. After removing points with significant height, each frame of the point cloud is projected vertically onto this 2D plane. Figure 6(a) shows such a projection. However, the resulting points are too dirty to apply the 2D occupancy grid mapping algorithm (Hess et al., 2016; Thrun et al., 2005) to construct a local occupancy grid effectively. In addition, directly using the projected points as the result of 2D-OGM will be very sparse when the resolution is small and cannot be filled effectively, which is unfavorable for the subsequent application of the map (such as active exploration). Therefore, we hope to convert all the projected 2D points to obtain a virtual scan which is more convenient for generating a 2D-OGM before applying the method in Hess et al. (2016) and Thrun et al. (2005). To do that, we adopt the ray-tracing technique to find the closest obstacle point along each ray direction, e.g., the yellow points in Figure 6(a). In addition, if there is no obstacle point along a certain ray, the farthest point, which has been labeled as traversable (e.g., the blue points in Figure 6(b)), is used to speed up the filling of the map to differentiate known area from unknown one. Figure 6 shows a created 2D-OGM. In our implementation, if the resolution of our 2D-OGM is set to 0.1 m per cell width and the size of the local region that LiDAR can sense is 40 m $\times$ 40 m, the size of our 2D-OGM is 400 pixels by 400 pixels.

Due to measurement noise, it can be assumed that the altitude of each local grid of the segmented traversable region in each LiDAR frame is subject to a Gaussian distribution. Specifically, the mean of each Gaussian distribution is the height of the local grid that is represented in the global coordinate system. In general, the standard deviation of each Gaussian distribution increases with the distance, d, to the LiDAR sensor. For efficiency purposes, we simply set the standard deviation as a combination of two parts, namely $\sigma = \alpha_d + \beta$. The α_d is a variable standard deviation which becomes larger with the increased distance d, and β is a constant used to ensure that the standard deviation of the cell under the robot is nonzero. Figure 7 shows an example of the distribution.

3.3. Building 3D Global Occupancy Grid Map

With a LiDAR-based odometry method, we can obtain the pose trajectory of the vehicle. For each keyframe pose along the trajectory, we estimate a local 2D-OGM and make it aligned with the pose represented in the global coordinate system. To construct a 3D global occupancy grid, our solution is to merge all local occupancy grid maps by stitching the neighboring ones so that the transition between two local 2D-OGMs is as smooth as possible, as shown in Figure 8.

To make our explanation easier to understand, let us use Figure 9 as a conceptual example although the real scenarios should be different from it. As shown in Figure 9, the scene includes an underpass and an overpass. When a robot carrying a LiDAR sensor moves along the dark line in Figure 9(a), correspondingly, the whole route can be split into five segments (red→dark→green→dark→blue, shown in Figure 9(b)) based on the visibility of the cell m to the LiDAR sensor. For example, the red, green, and blue segments are the first, third, and fifth one, respectively, where the cell m is visible to the LiDAR sensor. The two dark segments represent the

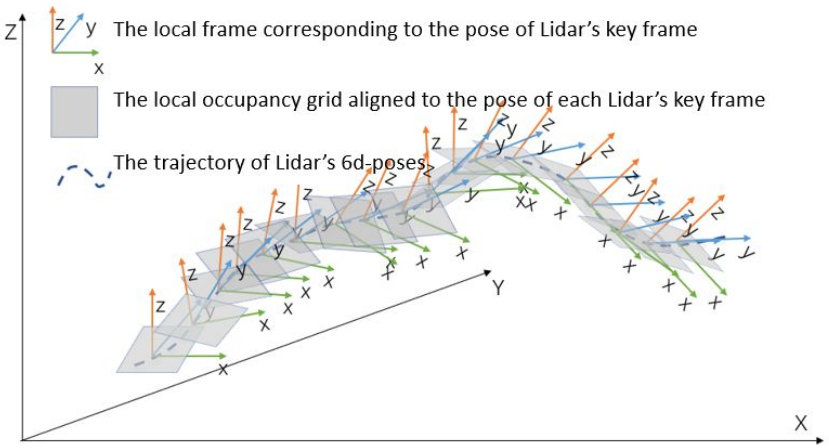
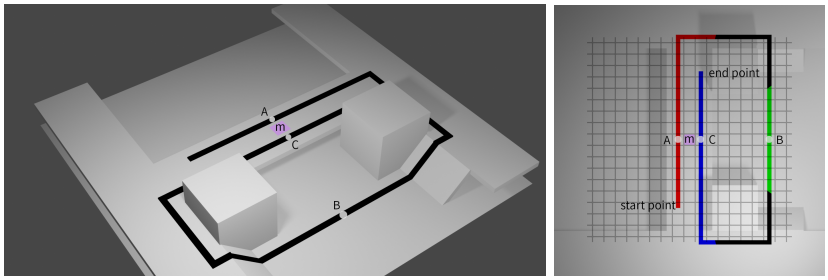


Figure 8. Alignment of local occupancy grid maps to the estimated 6D pose at each keyframe location of the pose trajectory.



(a) (b)

Figure 9. A conceptual illustration of fusing multiple observations made at a multiple-level local cell. (a) A typical scene and vehicle's trajectory, where m is a multilayer cell in the global map. (b) Top view. Each little grid is a cell of the global map. According to the *OLR* criterion, the representative observations of the red and blue segments (made at A and C) are redundant and are fused into the same layer. The representative observation of the green trajectory (made at B) is categorized into another layer.

two courses where the cell m is not visible to the LiDAR sensor, and will not affect the result at cell m, so these parts do not need to be discussed in the fusion process. Note that there are two levels at the cell m location, and the top level of the cell m is visible only to the red and blue segments, and the bottom level of m is visible only to the green segment. To efficiently generate the 3D-OGM, only the representative observations of the red, green, and blue segments are utilized during fusion. Let us take the representative observations of the red segment as an example. When the robot moves from the starting point to the end of the red segment, the distance between the robot and the cell m first decreases and then increases, with the minimal distance achieved at location A. Generally, the variance of measuring the ground cell m at location A is the smallest. We set the observation made to the cell m at location A denoted as the representative one of the red segment. Likewise, the observations made to the cell m at locations B and C are the representative ones of the green and blue segments, respectively.

We first give some notations before the detailed description. A certain cell in the real world (the global map) can be denoted by c_i, and there might be multiple-level planes at c_i. The distance between the moving robot and c_i over time is calculated as $d_{c_i}(t)$, where t is a time variable. The observations made to c_i by a robot over time in a segment s_j (e.g., the red one in Figure 9) are denoted as $\mathbf{m}_{c_i}^{s_j}(t), t \in [\text{the period when the robot was running in } s_j]$, where the representative observation is represented by $\mathbf{m}_{c_i}^{s_j}(t^*)$. That means the robot is closest to c_i in the segment s_j at time t^*.

As illustrated in Figure 9, if assuming the whole course includes N segments, $s_j, j \in [1, N]$, where the cell c_i is totally visible to the LiDAR sensor, we thus have N representative observations made to c_i, $\mathbf{m}_{c_i}^{s_j}(t^*), j \in [1, N]$. Without losing generality, each of the representative observations $\mathbf{m}_{c_i}^{s_j}(t^*), j \in [1, N]$, can initially be assumed to be made on each individual ground plane of different altitude (if assuming that there might exist more than one level of ground planes at the location of cell m). From a probabilistic perspective, $\mathbf{m}_{c_i}^{s_j}(t^*), j \in [1, N]$, is subject to a Gaussian mixture model (GMM) in theory. Similarly, this applies to the representative observations made to the other cells.

Before we estimate the GMM representation of the altitude at each cell c_i, we would first apply a merging step to remove the redundant ground planes. For example, the representative observations made at locations A and C to the cell m in Figure 9 should be merged because they are at the same altitude. In contrast, the representative observations made at A and B, or C and B, respectively, are not at the same altitude and should not be merged.

According to the method in Sun and Wang (2011), we use the *overlap rate* (OLR) to evaluate whether these independent observations should be merged or not. To do this, the set of 2D local occupancy grids that correspond to the representative observations $\mathbf{m}_{c_i}^{s_j}(t^*), j \in [1, N]$ are used, represented by $\mathbf{ogm}_{c_i}^{s_j}(t^*), j \in [1, N]$. Note that the altitude of each of these occupancy grids can also be represented by a Gaussian mixture model in the global coordinate system (refer to Section 3.2). Thus, we first sort them based on the means of their altitudes.

The lowest one with the smallest mean value of altitude, e.g., $\mathbf{ogm}_{c_i}^{s_q}(t^*)$, is initially assigned the label "1". The second lowest, e.g., $\mathbf{ogm}_{c_i}^{s_k}(t^*)$, is compared with $\mathbf{ogm}_{c_i}^{s_q}(t^*)$ based on altitude difference, and is assigned a label by checking whether the altitude difference is larger than a preset threshold. Generally, given the label of $\mathbf{ogm}_{c_i}^{s_q}(t^*)$, $f_{c_i}^{s_q}$, the label of its adjacent neighbor, $f_{c_i}^{s_k}$, can be updated based on (1),

$$f_{c_i}^{s_k} = \begin{cases} f_{c_i}^{s_q}, & \text{if } OLR(\mathbf{ogm}_{c_i}^{s_q}(t^*), \mathbf{ogm}_{c_i}^{s_k}(t^*)) > \epsilon \\ f_{c_i}^{s_q} + 1, & \text{else} \end{cases} \tag{1}$$

where $OLR(\cdot, \cdot) = p_{saddle}/p_{sub\_max}$. The p_{saddle} is the saddle-point location of the joint distribution of the altitudes of $\mathbf{ogm}_{c_i}^{s_q}(t^*)$ and $\mathbf{ogm}_{c_i}^{s_k}(t^*)$, and p_{sub_max} is the location of the smaller peak of the two modes. The ϵ is a threshold and can be 0.6 as described in Sun and Wang (2011).

In the end, we get Z independent labels which means there are Z layers of ground surface at c_i. Then each observation through the whole course can be assigned one of the Z labels based on (2):

$$f_{c_i}(t) = \begin{cases} f_{c_i}^{s_1}, & t < t_1^* \\ f_{c_i}^{s_N}, & t \geq t_N^* \\ f_{c_i}^{s_j}, & t_j^* \leq t < t_{j+1}^*, |\mu_j^* - \mu_t| \leq |\mu_{j+1}^* - \mu_t| \\ f_{c_i}^{s_{j+1}}, & t_j^* \leq t < t_{j+1}^*, |\mu_j^* - \mu_t| > |\mu_{j+1}^* - \mu_t| \end{cases} \tag{2}$$

where μ_t is the altitude of the observation made at time t, and the variables with $*$ are associated with the representative observations. For example, the t^* is the time instance at which a representative observation is made, and μ^* is the mean altitude value of a certain representative occupancy grid.

Finally, we will get Z sequences that can be used for fusion. According to the general elevation map fusion method (Triebel et al., 2006), the Z sequences are fused sequentially to generate the final result. Note that because our framework is an online mapping process, it is not necessary to wait until all observations thought the whole course are made. The update in (3) is an online process as well in real implementation:

$$\begin{aligned} \mu_{f,0:n_f} &= \frac{\sigma_{f,n_f}^2 \mu_{f,0:n_f-1} + \sigma_{f,0:n_f-1}^2 \mu_{f,n_f}}{\sigma_{f,0:n_f-1}^2 + \sigma_{f,n_f}^2} \\ \sigma_{f,0:n_f}^2 &= \frac{\sigma_{f,0:n_f-1}^2 \sigma_{f,n_f}^2}{\sigma_{f,0:n_f-1}^2 + \sigma_{f,n_f}^2} \\ \mu_{f,k} &\in \{\mu_t | f_{c_i}(t) = f\},\ \sigma_{f,k} \in \{\sigma_t | f_{c_i}(t) = f\} \end{aligned} \tag{3}$$

where $\mu_{f,0:n_f}$ and $\sigma_{f,0:n_f}$ represent the fusion result from the first observation to the last observation that has been assigned the label f, and n_f is the number of observations labeled as f.

The whole merging algorithm is based on the observations for the current cell and the corresponding pose. Therefore, when the poses of local maps change, it is only necessary to update the area in the global map which is covered by the local map whose pose changes.

3.4. Embedding Geometric Information for Vehicle Localization

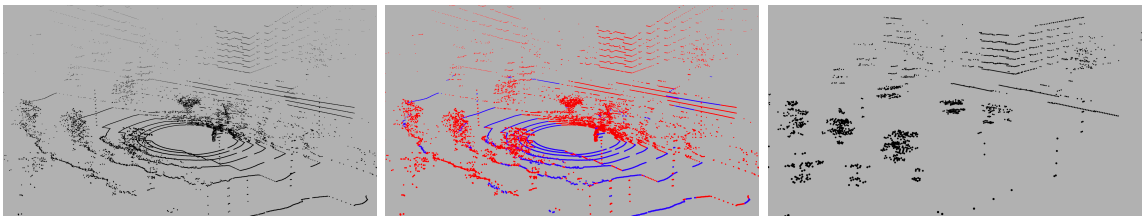
The above steps can be used to create a 3D road map or traversable-region map, which is mainly for vehicle path planning purposes. Sometimes it is also useful to keep the vehicle running in the track when vehicle localization is not accurate or it totally gets lost. But for accurate vehicle localization, the above created map cannot be reliable generally although it should be applicable to 2D structured scenes. When the vehicle is running in a nonstructured 3D scene, we have to incorporate 3D vertical geometric structures into the created road map. Inserting vertical structures into the road map is in analogy to growing trees along a road which stretches away into the distance.

To do that, vertical structures should be detected in each LiDAR point-cloud frame and then inserted into the generated road map. Based on our scheme presented in Section 3.1, we initially get the point-cloud region pc_{obst} which corresponds to obstacles (e.g., trees, bushes, light poles, or buildings). However, dynamic objects (moving pedestrians or vehicles) can also be detected as obstacles because this scheme cannot distinguish dynamic objects from static obstacles. Dynamic objects are not expected to be reserved as vertical structures in the final LiDAR Road-Atlas because they can affect adversely localization performance when the LiDAR Road-Atlas is used for navigation. In addition, the static obstacle points are somewhat redundant for localization and should be sparsified for efficiency purposes.

To deal with these issues on vertical geometric structures, we propose a probabilistic way to achieve the aim of reducing redundant 3D points and removing dynamic objects as much as possible. Specifically, an efficient 3D probabilistic grid map (3D-PGM) in x-y-z dimensions (z representing the vertical dimension) is created to reserve sparse vertical static structures, where the resolution of the x-y plane is set to be 0.1 m per cell width (the same as the one adopted in creating the 2D-OGM in Section 3.2). Vehicles actually drive on the ground, and there is no significant change in height for dynamic objects in vertical dimension locally. So we set a sparser resolution in the vertical dimension. The lowest grid in the 3D-PGM is slightly higher than the ground so as to effectively ignore the noisy obstacle points that may exist near the ground.

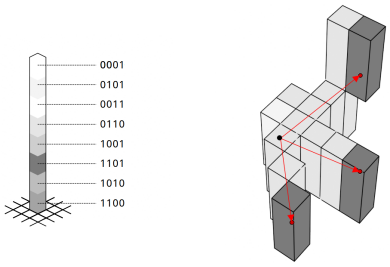
To remove dynamic obstacles through the 3D probabilistic grid map, we find that it is not necessary to use high-precision numbers to represent probability values. Specifically, for each 3D probability cell, we can use fewer bits (4 bits) to store a probability value to reduce memory cost during runtime, shown in Figure 11(a). The encoding process is a simple linear mapping of probability values to the range 1 to 15. Exclusion of 0 is to ensure that there are an odd number of states, ensuring that 0.5 (unknown state) can be represented exactly. So 1 to 7 means non-obstacle, 8 means unknown, and 9 to 15 means obstacle. In contrast, 2-bit representation cannot express uncertainty under the condition of an odd state (e.g., the part greater than 0.5 or less than 0.5 contains multiple possible values), and 3-bit representation cannot fill all bytes when the number of segments is not a multiple of 8. The 4-bit representation is the most saving and appropriate option. Then, we encode the occupancy probability for all segments at the same 2D position based on the 4-bit representation. In implementation, the probability update is done by mapping back to the range of 0 to 1 from the 4-bit representation based on an efficient look-up table. Considering the result of this encoding as a descriptor, the representation is ultimately consistent with a 2D map.

Once we have the economical representation of probability values, given a LiDAR frame, the point cloud pc_{obst} (e.g., the red region in the Figure 10(b)) corresponding to the vertical structures (including dynamic objects) can be represented with the 3D probability grid map. This procedure is similar to the one used for constructing a general 2D occupancy grid map, where a miss probability p_{miss} is applied to all segments passed by the ray from the LiDAR to the obstacle point, and a hit



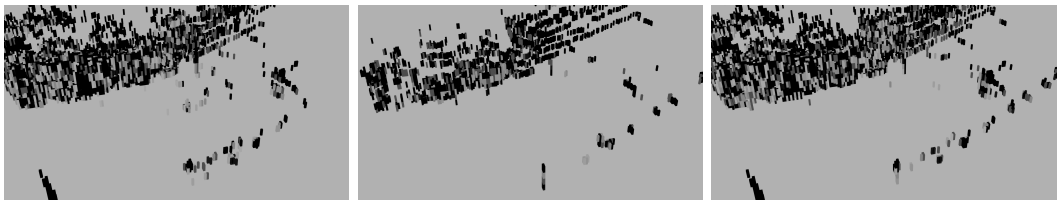
(a) (b) (c)

Figure 10. (a) A frame of raw point cloud data. (b) The result after applying the traversable region detection method introduced in Section 3.1, where the red points belong to obstacles and the blue points belong to the ground. (c) The reconstructed point cloud based on the encoded descriptor of each cell as described in Section 3.4.



(a) (b)

Figure 11. Grey-scale level shows the occupancy probability of a segment. The darker the grey-scale level, the larger the probability of being occupied. (a) An illustration of probabilistic coding, where multiple segments corresponding to the same 2D location are shown. In this example, we encode the probability with 4 bits and label it on the right. (b) The process of converting an obstacle point cloud to a 3D probability grid map for a LiDAR frame. The dark dot is the position of the LiDAR sensor and the three red dots represent three obstacle-grid locations.

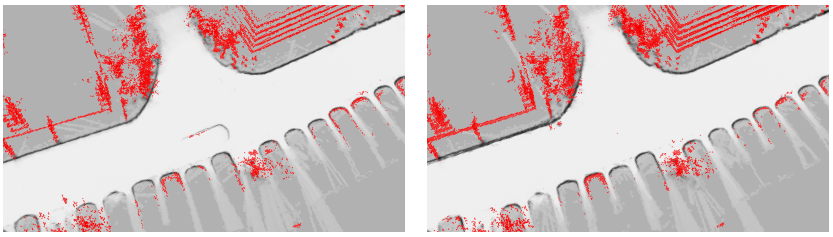


(a) (b) (c)

Figure 12. An illustration of the process of fusing two neighboring probabilistic grid maps. The darker the intensity, the higher the probability of being an obstacle (vertical structure). Segments corresponding to traversable road region are not drawn. (a) The global map before adding a new frame. (b) A new frame. (c) After adding the new frame. It can be seen that the probabilities of vertical structures (including dynamic obstacles) are updated and the dynamic obstacles (in the central part of the image) are removed.

probability p_{hit} is applied to the segment where an obstacle point is located. For ease of calculation, the sensor model we use takes fixed values for both p_{miss} and p_{hit}. The process of adding obstacle points within a single LiDAR frame is shown in Figure 11(b).

Reversely, we can decode the probability of each segment from the descriptors of each 2D cell, and based on these probabilities we can approximately reconstruct the obstacle point cloud with the obstacle segments. The reconstruction result of one frame is shown in Figure 10(c). Over time, we can construct a global map from each consecutive local frame by applying a probabilistic mixture model to fuse two 3D probabilistic grid maps. Figure 12 shows an update of the probabilistic grid map by adding a new frame (each keyframe as a local map) to the existing global map. We also show one example of filtering out dynamic objects with the probabilistic mixture in Figure 13.



(a) Before the car leaves. (b) After the car leaves.

Figure 13. Illustrating dynamic object filtering by probabilistic fusion: A car is leaving the parking lot.

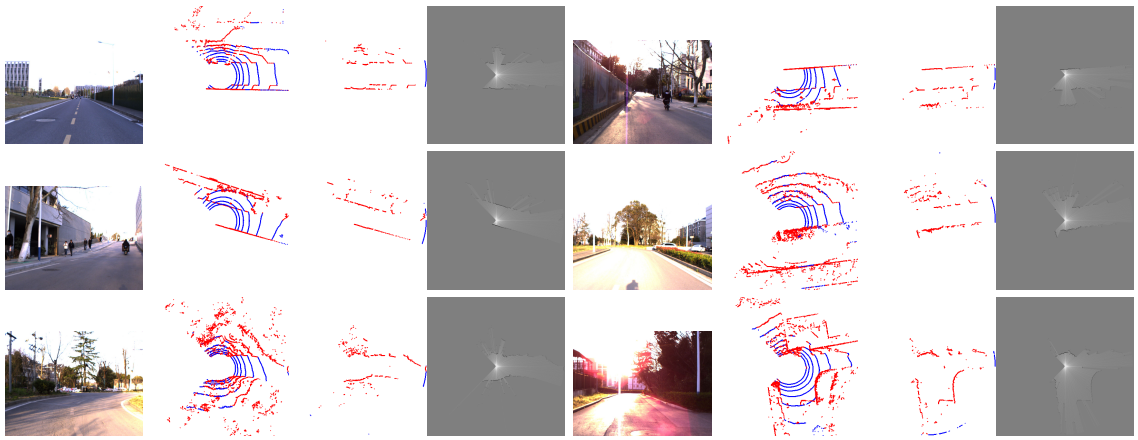


Figure 14. Examples of creating local 2D occupancy grid maps (2D-OGMs) in a real environment test, with the columns corresponding to the images captured by a camera installed on the robot, the frames of point cloud by VLP-16, the detected curb regions, and the created 2D-OGMs based on the curbs, respectively.

To apply the map to robot localization, one can reconstruct a point cloud from the 4-bit descriptors of a LiDAR frame, and utilize iterative closest point (ICP)-like registration method for vehicle localization in the map. In our localization implementation, we first encode the frame to be localized, then decode the point cloud for ICP, and then do ICP matching with the point cloud decoded from the global map. The purpose of encoding and then decoding the LiDAR frame to be localized is to make it consistent with the sparsity of the global point cloud.

4. Implementation Details and Experimental Results

We adopt the VLP-16 (16-channel) LiDAR as the only sensor to implement the proposed method, although we believe more advanced multichannel LiDAR sensors (e.g., 32 or 64 channels) should work equivalently or even can achieve better results.

4.1. Traversable- and Curb-Region Detection Accuracy

In Figure 14, we show some examples of creating local 2D occupancy grid maps (2D-OGMs) in a real environment test. In general, our method is a road-based or traversable-region-based map representation. Therefore, traversable- and curb-region detection performance is first evaluated on the KITTI-360 benchmark dataset (Xie et al., 2016). The reason we choose the KITTI-360 dataset is that this benchmark dataset provides ground truth semantic segmentation of street scenes. Curb detection is crucial for autonomous driving in urban scenes. Actually, in most urban areas, the lowest positive obstacles, whether for autonomous cars or for robots, are probably curbs, where positive obstacles are defined as the objects which stick out above the ground. The other positive

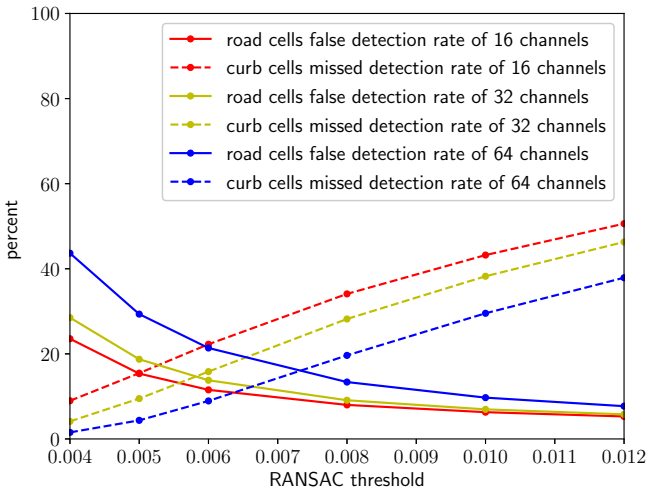


Figure 15. Performance of traversable- and curb-region detection using 16-channel, 32-channel, and 64-channel LiDAR point-cloud data with different thresholds in RANSAC.

obstacles such as vehicles or pedestrians are relatively easier to be detected. In addition, we also need curb-region detection results to build the 2D local grid map (Figure 6).

Therefore, both traversable- and curb-region detection performance is evaluated to ensure the accuracy of the created road map. Although our framework is implemented with a VLP-16 LiDAR sensor, it should also work with the other multichannel LiDAR sensors. To evaluate with LiDAR sensors of different numbers of channels, we tried our method on the original point-cloud data of the KITTI-360 benchmark and on the subsampled versions, respectively. Specifically, we sample one channel from every four channels of the KITTI-360 LIDAR data to simulate a 16-channel LiDAR sensor. We sample one channel from every two channels of the KITTI-360 LIDAR data to simulate a 32-channel LiDAR sensor.

The KITTI-360 benchmark provides multiframe 3D semantic labels. We divide the point cloud into cells at the resolution of 0.1 m $\times$ 0.1 m per cell, and label these cells as road, curb, or irrelevant cells based on semantic information, where the irrelevant cells mostly contain nontrivial obstacles such as vehicles or trees. These labels are automatically generated based on the existing semantic labels of KITTI-360. Then our curb and traversable detection results represented in the 2D cells are compared with the ground truth.

For the curb region, we evaluate the N_{missed} only, which is calculated as the number of curb cells that have been labeled as traversable. Additionally, we deem it correct if a curb cell is labeled as an irrelevant cell. For the traversable region, we evaluate the N_{false} only, which is the number of road cells that are mislabeled as obstacle. Given the total number of road cells, N_{road}, and total number of curb cells, N_{curb}, we define the missed detection rate of curb cells $rate_{missed} = \frac{N_{missed}}{N_{curb}}$ and false detection rate of road cells $rate_{false} = \frac{N_{false}}{N_{road}}$. Based on the different threshold in RANSAC, we finally get the performance curves as shown in Figure 15. From the figure, we observe that the missed-detection rate deceases with more channels of LiDAR beams. Counterintuitively, we have noticed that the false detection rate of traversable-region cells increases when using the LiDAR sensor of more channels. The main reason is that the LiDAR sensor with more channels has more points, so there are more false road points generated in general, and these points will fall into more cells. As a result, more cells are incorrectly marked as curb.

4.2. The 3D LiDAR Road-Atlas Results

To evaluate the whole 3D mapping process, we have evaluated our approach in three real-world urban scenes and two virtual urban scenes (garage and campus). The three real-world urban scenes include

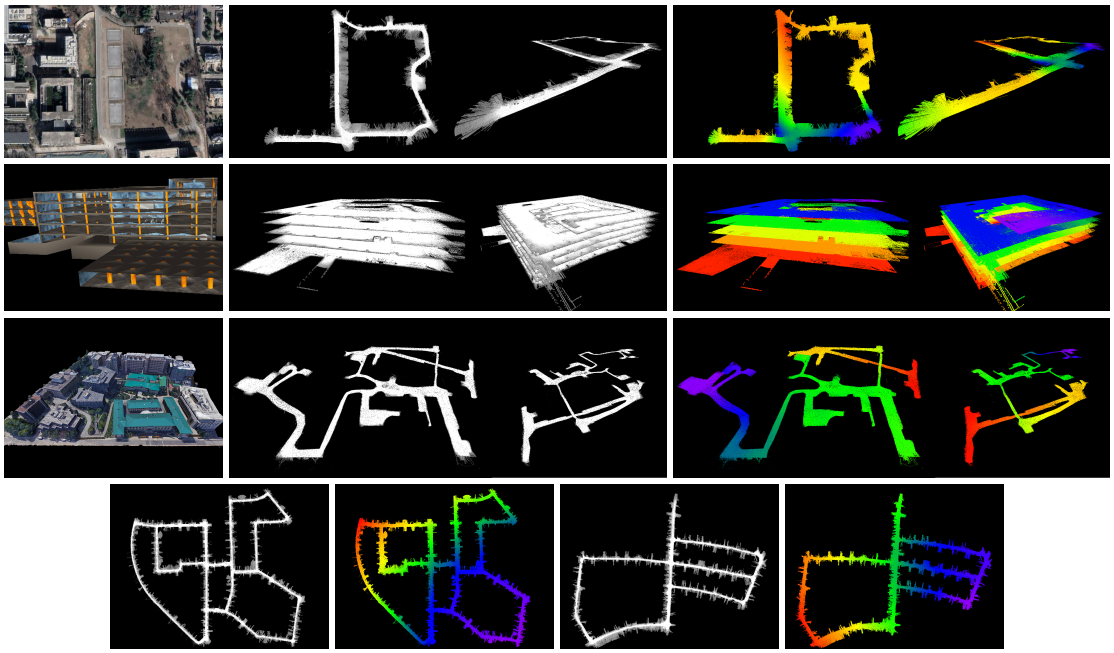


Figure 16. Global 3D LiDAR-Atlas in five scenes. **Top three rows**: The first column shows a picture of the whole scene except the two KITTI odometry scenes, for which we do not have the global pictures. The second column shows two views of the created LiDAR road atlas for each scene. To perceive the altitude variation at each location of the created 3D road map, we have encoded the altitude of each cell in gradually changing colors, where red represents the lowest in altitude, blue represents the highest in altitude, and green and yellow in the middle. The third column shows this effect. **The last row** shows correspondingly the LiDAR-Atlas results for the two KITTI odometry scenes.

our own campus scene and two scenes from the KITTI Odometry datasets (Geiger et al., 2012). The two virtual urban scenes were parts of the Autonomous Exploration Development Environment (Cao et al., 2022) built by a CMU robotics team. All evaluations are based on a 16-channel LiDAR sensor.

In the three real-world urban scenes, we use the LeGO-LOAM approach (Shan and Englot, 2018) to estimate a relative pose between two consecutive LiDAR frames, and use the LiDAR-Iris method (Wang et al., 2020) to provide a stable loop-closure detection. Part of the reason for choosing such a set of algorithms is that it can provide good poses in the height axis. In the two virtual scenes, we use the ground truth pose estimation provided by the datasets. Regarding the selection of keyframes, in the three real-world scenes, we directly use the keyframes of SLAM as our keyframes (because only the keyframes of SLAM can get more accurate poses); in the virtual scene, we use 1 Hz frequency to extract key frames. In Figures 16 and 17, we show the obtained 3D LiDAR Road-Atlas without embedded vertical structures and the 3D LiDAR Road-Atlas with embedded vertical structures, respectively.

Note that although the tested real-world scenes contain non-flat road scenarios, there are no multiple-layer road infrastructures, such as underpass and overpass. In contrast, the simulated garage and campus scenes provided by the autonomous exploration development environment (Cao et al., 2022) contain such multiple-layer road or traversable-region scenes. The obtained LiDAR Road-Atlas maps from the two simulation environments have shown that our method is effective in dealing with such scenes. In implementation of traversable and curb detection in the simulation environment, due to the large horizontal resolution of the LiDAR in the simulation environment (350 points per circle), we adjusted the width of LiDAR imagery to 350, and the width of the local window for detecting traversable areas and obstacles to 9 (also about 10°).

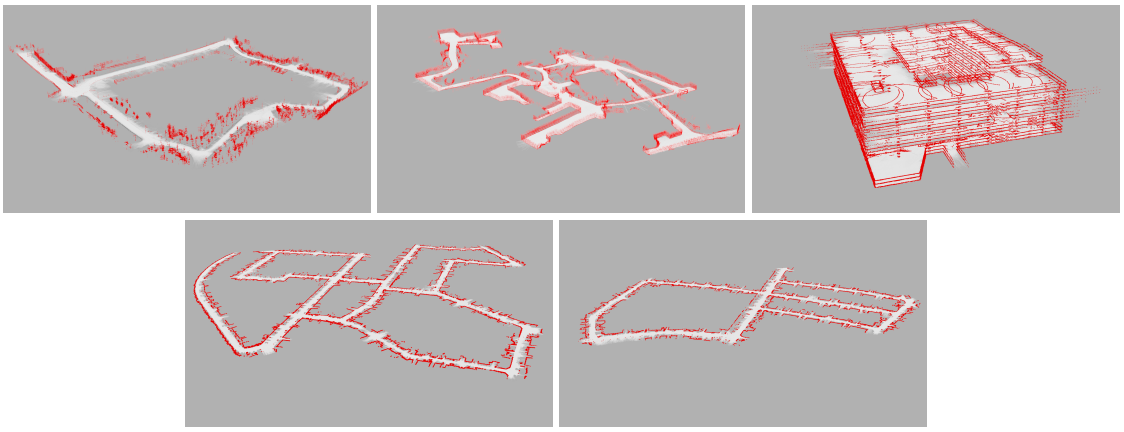


Figure 17. Embedded static-obstacle points for vehicle localization results in five scenes: our campus, v-campus, v-garage, KITTI-00, and KITTI-05. Embedded static-obstacle points for vehicle localization are in red and traversable regions for vehicle navigation are in white.

Table 1. Runtime memory and hard-drive (H-D) storage usage of our map.

		our campus	**v-campus**	**v-garage**	**KITTI00**	**KITTI05**
OctoMap radius=30 m	runtime(GB*)	4.25	3.53	2.43	NA*	NA
	H-D(MB*)	371.71	295.53	222.46		
OctoMap radius=20 m	runtime(GB)	3.55	3.09	2.36	NA	NA
	H-D(MB)	306.96	253.78	214.66		
our method with radius=30 m, 32 bits obstacle descriptor*	runtime(GB)	8.69	5.71	6.44	10.32	6.09
	H-D(MB)	49.77	52.51	41.50	178.18	120.59
our method with radius=20 m, 32 bits obstacle descriptor	runtime(GB)	3.95	2.02	3.28	4.69	2.79
	H-D(MB)	39.57	47.96	40.93	148.27	96.37
our method with radius=30 m, 8 bits obstacle descriptor*	runtime(GB)	6.06	4.34	4.84	7.10	4.51
	H-D(MB)	41.06	43.27	34.44	146.97	99.46
our method with radius=20 m, 8 bits obstacle descriptor	runtime(GB)	3.05	1.51	2.70	3.81	2.19
	H-D(MB)	32.64	39.56	33.83	122.30	79.49
LeGO-LOAM*	H-D(MB)	20.08			72.78	45.67

*Due to immense memory usage and very long update time, OctoMap cannot complete the mapping of the KITTI dataset.
*The file size of LeGO-LOAM refers to the sum of corner point cloud and surface point cloud.
*GB, gigabytes. MB, megabytes.
*The number of bits for obstacle (vertical-structure) descriptor here is equal to the product of the number of segments and the number of bits per segment.

4.3. Memory Usage and Map Size

The total memory consumption of our method includes two major parts, i.e., the cost for the created 3D road map (e.g., the results shown in Figure 16) and the space required for saving the encoded vertical structure of each cell. As in the occupancy elevation map (Souza and Gonçalves, 2016), our 3D road map also takes the advantage of the probabilistic fusion when merging the 2D local occupancy grid maps. Generally, it can save a lot of memory compared to the other 3D occupancy grid maps, e.g., the OctoMap method (Hornung et al., 2013). To make a comparison with the OctoMap method in terms of memory and hard-disk usage, we have recorded the file size and memory usage by our method and the OctoMap method for different scenes, shown in Table 1. The input data of the two methods are exactly the same, and the cell or grid resolution equals 0.1 m.

In the table, the "runtime" represents the total runtime memory cost till the end of the whole online mapping process. The "H-D" represents the size of the file after the map is saved as a file in the hard drive. In our implementation, the runtime memory is dependent on several major factors, i.e., the scanning range of the LiDAR sensor, the cell resolution (representing the actual size in 3D world per cell), the number of vertical segments which are used for encoding the vertical structure within each cell (determining the height limit of the point cloud within each cell), the number of bits per segment, and the number of collected keyframes during traveling unit distance (e.g., 1 m in our work).

To make a fair comparison with the OctoMap representation, the cell resolution and the number of collected keyframes across each whole dataset are set to the same for both our method and the OctoMap algorithm (Hornung et al., 2013). Therefore, the rest parameters that can be adjusted for determining the "runtime" and "H-D" values of our method are the scanning range of the LiDAR sensor and the number of bits used to encode the vertical structure within each cell.

In terms of the scan range of the LiDAR sensor, it can affect the "runtime" memory of our method significantly. In our 3D road map, we temporarily store the local 2D occupancy grid maps as the intermediate results in order to allow for updating poses of frames (such as the pose optimization when loop closure happens). In general, the size of a local 2D-OGM depends mostly on the LiDAR's sensing range and more memory consumption is necessary as the sensing range becomes larger. Normally this value can be set as the range limit of a LiDAR senor. For example, the VLP-16's scan-range limit can be up to 100 m. In this case, the size of a local 2D-OGM is 2000×2000 given the cell resolution of 0.1 m per cell, which means that there are about 4×10^6 cells in each local 2D-OGM.

Empirically, the point cloud 60 m away is generally quite sparse, meaning that it is not necessary to use the whole frame of the point cloud for building a map. Considering these factors, it is preferable to reduce the size of the local 2D-OGM by only utilizing the point cloud within a certain small range. For example, we set the radii of the LiDAR's scan range to be 20 m and 30 m, respectively, in our experiments. Additionally, we can further reduce the memory cost on the descriptor of encoding the vertical structure of each cell from 4 bytes to 1 byte only. Accordingly, the runtime memory usage of our method with different settings of mapping parameters is shown in Table 1.

When only using a point cloud within a scan range of 20 m for building a 2D-OGM and using 1-byte descriptors, the runtime memory usage in all environments is smaller than that of OctoMap except for the virtual garage environment. We can also observe a significant reduction of runtime memory by using 1-byte descriptors in contrast to using 4-byte descriptors, where we observe a reduction of about 23%, 25%, and 17% on runtime memory cost in our campus, the v-campus, and v-garage scenes, respectively. It deserves pointing out that the garage environment is a multilayer parking-lot environment (as shown in Figure 16) and the trajectory is more concentrated within a small space. For such kinds of scenarios, the memory usage of OctoMap is smaller. In contrast, our method needs to build the 3D road atlas by tracing a quite long trajectory within the same area repeatedly. Thus, the intermediate results (the local 2D-OGM) can occupy lots of memory, which results in a larger runtime memory consumption than the OctoMap. When using the point cloud within the scan range of 30 m for building the 2D-OGM, the runtime memory usage of our method is generally more than that of the OctoMap method. This is still caused by the memory coverage of the intermediate results which are prepared for pose correction during the loop-closure process. But we also notice that the OctoMap method will crash when applied to very large scenes, such as KITTI00 and KITTI05 odometry benchmark datasets, due to the immense memory cost and extremely long map update time. In contrast, our method can still be applicable to such kinds of large scenes.

In terms of the hard-drive storage space, it can be seen from Table 1 that our method saves a lot more storage space than the OctoMap, whether in the simulated environments or in real scenes. We also observe that the hard-drive storage of the map by our method does not change much when using either 4-byte or 1-byte descriptors for representing vertical structure geometry within each map cell. For all the results shown in Table 1, a 16-channel LiDAR (or simulated 16-channel LiDAR) sensor

Table 2. Time consumption for building local map from point cloud data, tested on our campus dataset.

Resolution (m)	Range (m)	RANSAC detection time consumption (s)	Building a local map from detection results time consumption (s)
0.1	20	0.0137	0.0325
0.2	20	0.0139	0.0189
0.1	40	0.0130	0.0595
0.2	40	0.0140	0.0303

is used, where the point cloud within a radius of 30 m is used for building a local 2D-OGM with only 1-byte descriptor for encoding the vertical structure of each cell. Because it is the completed map that will be used in the "repeat" stage of the "teach and repeat" paradigm, considering the small hard-drive storage space of our method, our map representation is more intriguing than the existing 3D LiDAR maps, such as the OctoMap, when applied to a large-scale environment.

We also compare our method with the LeGO-LOAM approach (Shan and Englot, 2018) in terms of the hard-drive storage size of the resulting map file. In the LeGO-LOAM method, only points with high curvature or points within large planar regions (corresponding to walls or road) are reserved in the resulting map. On three compared datasets, the "H-D" values of the LeGO-LOAM method are smaller than ours, but the point-cloud map obtained by the LeGO-LOAM method alone cannot be used for navigation because a traversable region is not provided in the map. In contrast, the LiDAR Road-Atlas obtained by our method is a complete map which is favorable for navigation purposes alone.

4.4. Online Map Update

Our map representation can be applied to an online SLAM process, for which it is important to update the map instantly. Thus, we tested the time consumption on map update. There is no need to worry about the time to build a local map from the point cloud: the time consumption of this part is significantly smaller than the LiDAR data interval, and the specific results are listed in Table 2. In the implementation, the map is updated once per second. When there is no pose update, the update will be skipped. The frequency of the LeGO-LOAM back-end optimization for pose update is less than 1 Hz. The code of the map update part is paralleled in eight threads. We record the time of map update for three scenes and get the results as shown in Figure 18.

Specifically, for the three scenes used for evaluating the map-update time cost, the first scene is part of our campus (the first row of Figure 16), to which our full online mapping procedure is applied, including front-end pose estimation, loop-closure detection, back-end optimization, traversable-region (curb) detection, 2D-OGM generation, and multiple-layer road-map fusion. The second and third scenes are the virtual campus and garage scenarios, where poses between consecutive LiDAR frames are known beforehand. Thus, we do not need the front-end pose estimation, loop-closure detection, and pose optimization steps.

We can observe that most map updates can be finished within 1 s when point-cloud data within the radius of 20 m are used for creating the local 2D-OGM, shown as the bottom row of Figure 18. In contrast, it generally takes more time to update the map when using point-cloud data within the radius of 30 m, shown as the top row of Figure 18, which makes much sense in that a larger local 2D-OGM needs more time to be incorporated into the global map.

The first column of Figure 18 shows the results of running SLAM online and updating the map in our campus environment. It can be seen that there are two obvious peaks near the 141st and 181st updates. This is due to the fact that more local map poses are updated after finding two loop closures. The other results shown in Figure 18 correspond to the map update in the two virtual scenes, where relative poses between two LiDAR frames are given in the simulation environment and thus no significant time-cost variations are present. The last column shows the results of the simulated garage scene. It can be clearly seen that the update time increases mostly linearly as the

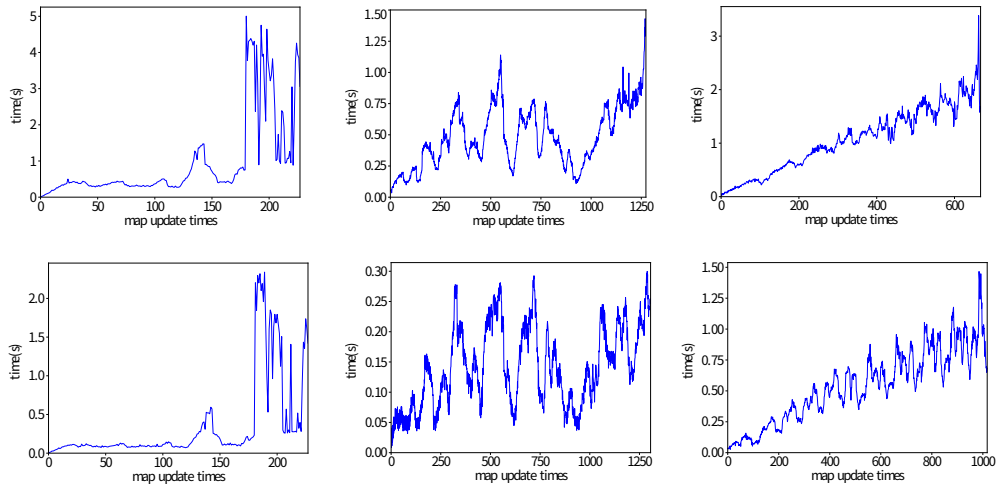
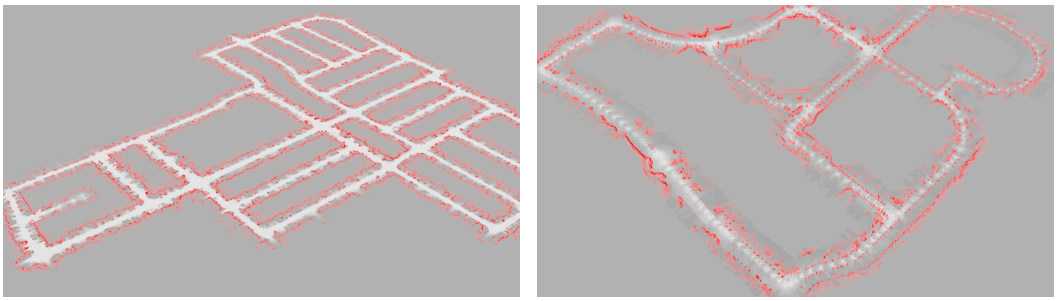


Figure 18. Time consumption for map update in three scenes. The **top row** corresponds to the map update results when using point-cloud data within the radius of 30 m for mapping, and the **bottom row** corresponds to the results of using point-cloud data within the radius of 20 m for mapping. The **left column** shows the results of our campus scene, where there are a short reverse loop closure in the middle and a long loop closure in the last part of the course. Therefore, we can observe an obvious increase of time cost in the middle, and a sharp increase in time cost in the last part. The **middle column** shows the map update results of the virtual campus scene, where there is no loop closure in the whole course. But there exist fusions of multiple-layer road in the middle; therefore, occasionally we can see an increase in time consumption. The **right column** shows the map update results of the virtual garage scene, where there is no loop closure detected. Due to the repeated fusion of multiple-layer driving ways in the garage, we can observe an increase of time cost globally.

number of map updates increases. This is because the garage itself covers a relatively small area, but it has more floor layers. The number of local maps covered in the same 2D-OGM cell continues to increase, and the time used to update a cell is proportional to the number of local maps that need to be fused.

4.5. Vehicle Localization in the Map

Once our map is ready, we first test the performance of vehicle localization in it given the LiDAR point-cloud data. Our test of localization is performed based on the ICP method by directly matching the current LiDAR point cloud with the obstacle point cloud which is decoded from our map representation. Since ICP is sensitive to the initial value, we use the localization result of the previous frame as the initial value of the new frame. At the same time, in order to accurately describe the localization error, we use the ground truth pose in mapping to exclude the pose error of the map itself caused by SLAM in the results. The way of our localization experiment is similar to that in Monte Carlo localization (MCL) methods. The Range-MCL (Chen et al., 2021) is a method with better experimental results in the class of Monte Carlo localization methods. It provides test results on multiple public datasets including Apollo (Lu et al., 2019) and MulRan (Kim et al., 2020) in the paper, so we compare our results with the Range-MCL. We test on Apollo and MulRan datasets, which were collected when vehicles ran more than once in the same real scenes. Therefore, both datasets are suitable for testing localization performance given the map representation. We take the same trajectories as used in Range-MCL for ease of comparison. For the Apollo dataset, since the beginning of the test trajectory is not included in the map, it is difficult to locate it effectively. We start the test trajectory on this dataset from the 200th frame. For mapping data, since dense continuous frames are not necessary for mapping, we take one frame every 20 frames. Testing is done on every frame. For both mapping and test data, we do not directly use the original 64-channel data,



(a) Apollo (b) MulRan

Figure 19. The created LiDAR Atlases from the Apollo and MulRan datasets based on our method. They are used for localization when a vehicle is running a second time in the maps.

Table 3. Comparison between our method and the Range-MCL (Chen et al., 2021) in terms of localization error and time cost.

Dataset	Method	Translation RMSE (m)	Rotation RMSE (deg)	Average time (s)
Apollo	Our method	0.26	1.07	0.1011
	Range-MCL	0.57	3.40	
MulRan	Our method	0.89	1.29	0.1058
	Range-MCL	0.83	3.14	

but simulate 16-channel LiDAR data by taking one channel from every four channels to validate the performance of localization using a 16-channel LiDAR and our LiDAR Road-Atlas representation. The maps constructed from the mapping data of both datasets are shown in Figure 19.

For the test results, we obtain the translation and rotation errors, and the average time cost for each frame. In Table 3, we compare the results with those given in Range-MCL. In this test, our method adopts the following parameters: resolution 0.2 m, radius 40 m (because some scenarios are relatively open and a small radius can result in inaccurate positioning). The obstacle point (static vertical structures) takes the range of -1 to 7 m with respect to the sensor in the vertical dimension, which is enough to cover most effective vertical structures. The number of segments in the vertical dimension is 8, and each segment uses 4 bits to represent the probability. This means that the length of the descriptor for encoding vertical structures in each cell is 32 bits.

To demonstrate the effect of different parameters on localization accuracy, Table 4 shows the effect of various numbers of segments in splitting the vertical dimension and Table 5 shows the effect of different LiDAR range radius and map resolutions. We observe that more segments can slightly improve the accuracy of localization, but at the same time, the time cost will increase due to the increase in the number of points.

In order to ensure that the occupied space does not increase significantly, our experiments simultaneously change the resolution and range while keeping the number of cells in the local map unchanged. Increasing the cell side length will reduce the number of points and increasing the range will bring a lesser increase in the number of points. It can be seen that the implementation shows that as these two increase, the time consumed decreases.

4.6. Path Planning in the Map

Since our map belongs to a traversable- or road-based 3D map representation, the traversable or road region has been explicitly represented in the map. Therefore, another function that our map can provide is path-planning capability. Given a start location and a destination in the map, a global path-planning algorithm can be applied to find a route for vehicle navigation given accurate

Table 4. Effects of various number of vertical segments on localization.

Dataset	Segments	Translation RMSE (m)	Rotation RMSE (deg)	Average time (s)
Apollo	2	0.288	1.106	0.0792
	4	0.281	1.110	0.0867
	8	0.265	1.073	0.1011
	16	0.257	1.002	0.1305

Table 5. Effects of different LiDAR range radius and map resolutions on localization.

Dataset	(resolution, range)	Translation RMSE (m)	Rotation RMSE (deg)	Average time (s)
Apollo	(0.2, 40)	0.265	1.073	0.1011
	(0.3, 60)	0.512	0.976	0.0908
	(0.4, 80)	0.364	0.886	0.0817
	(0.2, 60)	0.267	0.963	0.1506
	(0.4, 40)	0.370	1.078	0.0807
MulRan	(0.2, 40)	0.892	1.286	0.1058
	(0.3, 60)	1.126	1.184	0.0946
	(0.4, 80)	0.875	1.159	0.0844
	(0.2, 60)	0.867	1.214	0.1330
	(0.4, 40)	0.905	1.250	0.0699

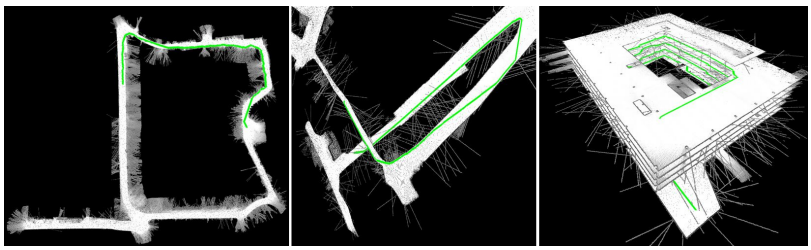


Figure 20. Three examples of path planning based on A* algorithm in the created 3D maps by our method.

localization. Figure 20 shows three planned paths in three road maps created by our methods, respectively, where the A* algorithm is applied to find the optimal path in each road map. It can be observed that the planned routes actually extend at different layers of the road map of the virtual campus and virtual garage scenes.

5. Discussion

The proposed map representation is built in an online mode by a LiDAR-based full SLAM framework. Combined with an exploratory planning algorithm, the proposed method can be used for autonomous robotic exploration in unknown environments. Certainly, the proposed method can also be used in an offline mode where the collected data are processed in a workstation to obtain the environment maps. In all, the created map representation can be scalable, extendable, and applicable in various ways.

Scalable. Our map representation can be readily scaled up whenever necessary to expand the current map size. Since the hard-drive storage size of our map representation is small and compact enough, one preferable way to expand the map scale is to utilize the existing map that has been saved to hard drive, and one only need to load the map file into memory, and to relocalize the

current position of vehicle in the existing map, and do the same online mapping as this work. This way is better than continuously building a large map online in terms of the runtime memory usage.

Extendable. Although we only use a single 16-channel LiDAR sensor to build the map, one can use a LiDAR sensor with more channels, such as 32- or 64-channel LiDAR sensors. One advantage of our map representation when only using multiple-channel LiDAR is that the runtime memory consumption and hard-drive storage should approximately be invariant to the number of LiDAR channels. Additionally, one can add heterogeneous information to the map by incorporating the other types of sensor data, such as visual features from cameras of different spectra. To do that, one can estimate the external parameters between the LiDAR sensor and camera before extracting visual features along with corresponding point-cloud descriptors. Generally, richer information can be obtained from a camera and stored in the map. Thus it can result in more accurate vehicle localization if vehicles are equipped with cameras as well.

Applicable. Our map representation can also be used for detecting dynamic objects given the map, and is also useful for local motion planning. Because our map's hard-drive storage is small, it can be applied to distributed multiple-robot online mapping, and robots can share already saved maps with each other. Additionally, our map representation is compact in terms of the cell resolution, where robots can use a coarse-to-fine manner to exchange intermediate map results under bandwidth limitation whenever necessary.

6. Conclusion

We propose a compact and efficient 3D map representation for autonomous robot or vehicle navigation, called the LiDAR Road-Atlas. It can be generated by an online mapping framework based on incrementally merging 2D local occupancy grid maps, where automatically labeled traversable regions and 3D landmarks are encoded and embedded into the LiDAR Road-Atlas. Given the LiDAR Road-Atlas, one can achieve accurate vehicle localization in a coarse-to-fine manner, path planning in the labeled traversable regions, and some other tasks. We compare our map representation with a couple of popular map representation methods in robotics and autonomous driving societies, and our map representation is more favorable in terms of efficiency, scalability, and compactness.

Acknowledgments

This work is supported by the Science and Technology Development Fund of Macau SAR (File No. AGJ-2021-0046, SKL-IOTSC(UM)-2021-2023, and 0015/2019/AKP), Guangdong-Hong Kong-Macao Joint Laboratory of Human-Machine Intelligence-Synergy Systems (No. 2019B121205007), and the startup project of Macau University (SRG2021-00022-IOTSC).

ORCID

Chengzhong Xu https://orcid.org/0000-0001-9480-0356
Hui Kong https://orcid.org/0000-0002-5303-0276

References

Ahtiainen, J., Stoyanov, T., and Saarinen, J. (2017). Normal distributions transform traversability maps: Lidar-only approach for traversability mapping in outdoor environments. *Journal of Field Robotics*, 34(3):600–621.

Bares, J., Hebert, M., Kanade, T., Krotkov, E., Mitchell, T., Simmons, R., and Whittaker, W. (1989). Ambler: An autonomous rover for planetary exploration. *Computer*, 22(6):18–26.

Cao, C., Zhu, H., Yang, F., Xia, Y., Choset, H., Oh, J., and Zhang, J. (2022). Autonomous exploration development environment and the planning algorithms. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8921–8928. IEEE.

Chen, L., Yang, J., and Kong, H. (2017). Lidar-histogram for fast road and obstacle detection. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1343–1348. IEEE.

Chen, X., Vizzo, I., Läbe, T., Behley, J., and Stachniss, C. (2021). Range image-based lidar localization for autonomous vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5802–5808. IEEE.

Elfes, A. (1987). Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265.

Fischler, M. A., and Bolles., R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. In *Communications of the ACM*, pages 381–395. ACM.

Funk, N., Tarrio, J., Papatheodorou, S., Popović, M., Alcantarilla, P. F., and Leutenegger, S. (2021). Multi-resolution 3D mapping with explicit free space representation for fast and accurate mobile robot motion planning. *IEEE Robotics and Automation Letters*, 6(2):3553–3560.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361. IEEE.

Herb, M., Weiherer, T., Navab, N., and Tombari, F. (2019). Crowd-sourced semantic edge mapping for autonomous vehicles. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7047–7053. IEEE.

Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2D lidar SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206.

Jeong, J., Cho, Y., and Kim, A. (2017). Road-SLAM: Road marking based SLAM with lane-level accuracy. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1736–1473. IEEE.

Kim, G., Park, Y. S., Cho, Y., Jeong, J., and Kim, A. (2020). MulRan: Multimodal range dataset for urban place recognition. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6246–6253. IEEE.

Kweon, I.-S., Hebert, M., Krotkov, E., and Kanade, T. (1989). Terrain mapping for a roving planetary explorer. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 997–1002. IEEE.

Lu, W., Zhou, Y., Wan, G., Hou, S., and Song, S. (2019). L^3-Net: Towards learning based lidar localization for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6389–6398.

Lu, Y., Huang, J., Chen, Y.-T., and Heisele, B. (2017). Monocular localization in urban environments using road markings. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 468–474. IEEE.

Morales, Y., Carballo, A., Takeuchi, E., Aburadani, A., and Tsubouchi, T. (2009). Autonomous robot navigation in outdoor cluttered pedestrian walkways. *Journal of Field Robotics*, 26(8):609–635.

Pfaff, P., and Burgard, W. (2005). An efficient extension of elevation maps for outdoor terrain mapping. In *Proceedings of the International Conference on Field and Service Robotics (FSR)*, pages 165–176.

Pfaff, P., Triebel, R., and Burgard, W. (2007). An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 26(2):217–230.

Qin, T., Chen, T., Chen, Y., and Su, Q. (2020). AVP-SLAM: Semantic visual mapping and localization for autonomous vehicles in the parking lot. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5939–5945. IEEE.

Qin, T., Zheng, Y., Chen, T., Chen, Y., and Su, Q. (2021). RoadMap: A light-weight semantic map for visual localization towards autonomous driving. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Ranganathan, A., Ilstrup, D., and Wu, T. (2013). Light-weight localization for vehicles using road markings. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 921–927. IEEE.

Rehder, E., and Albrecht, A. (2015). Submap-based SLAM for road markings. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1393–1398. IEEE.

Saarinen, J. P., Andreasson, H., Stoyanov, T., and Lilienthal, A. J. (2013). 3D normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments. *International Journal of Robotics Research*, 32(14):1627–1644.

Schreiber, M., Knöppel, C., and Franke, U. (2013). LaneLoc: Lane marking based localization using highly accurate maps. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 449–454. IEEE.

Shan, T., and Englot, B. (2018). LeGO-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE.

Souza, A., and Gonçalves, L. M. (2016). Occupancy-elevation grid: An alternative approach for robotic mapping and navigation. *Robotica*, 34(11):2592–2609.

Souza, A., Maia, R. S., Aroca, R. V., and Gonçalves, L. M. (2013). Probabilistic robotic grid mapping based on occupancy and elevation information. In *2013 16th International Conference on Advanced Robotics (ICAR)*, pages 1–6. IEEE.

Steinbrücker, F., Sturm, J., and Cremers, D. (2014). Volumetric 3D mapping in real-time on a CPU. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2021–2028. IEEE.

Sun, H., and Wang, S. (2011). Measuring the component overlapping in the Gaussian mixture model. *Data Mining and Knowledge Discovery*, 23(3):479–502.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. MIT Press.

Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2276–2282. IEEE.

Wang, Y., Sun, Z., Xu, C.-Z., Sarma, S. E., Yang, J., and Kong, H. (2020). LiDAR Iris for loop-closure detection. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5769–5775. IEEE.

Xie, J., Kiefel, M., Sun, M.-T., and Geiger, A. (2016). Semantic instance annotation of street scenes by 3D to 2D label transfer. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3688–3697. IEEE.

Yue, Y., Wang, D., Senarathne, P., and Moratuwage, D. (2016). A hybrid probabilistic and point set registration approach for fusion of 3D occupancy grid maps. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 001975–001980. IEEE.

How to cite this article: Wu, B., Xu, C., & Kong, H. (2023). LiDAR Road-Atlas: An efficient map representation for general 3D urban environment. *Field Robotics, 3*, 435–459.