박 사 학 위 논 문
Ph.D. Dissertation

점군 데이터의 공간 상관성을 활용한 실시간 정밀 점유 지도 생성

Real-time Dense Occupancy Mapping
using Spatial Correlation of Point Clouds

2022

권 용 선 (權 容 善 Kwon, Youngsun)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

점군 데이터의 공간 상관성을 활용한 실시간 정밀 점유 지도 생성

2022

권 용 선

한 국 과 학 기 술 원

전산학부

점군 데이터의 공간 상관성을 활용한 실시간 정밀 점유 지도 생성

권 용 선

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2021년 12월 6일

심사위원장 윤 성 의 (인)

심 사 위 원 명 현 (인)

심 사 위 원 성 민 혁 (인)

심 사 위 원 조 성 호 (인)

심 사 위 원 최 성 희 (인)

Real-time Dense Occupancy Mapping using Spatial Correlation of Point Clouds

Youngsun Kwon

Advisor: Sung-Eui Yoon

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
December 6, 2021

Approved by

Sung-Eui Yoon
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS 권용선. 점군 데이터의 공간 상관성을 활용한 실시간 정밀 점유 지도 생성. 전산학부. 2022년. 61+iv 쪽. 지도교수: 윤성의. (영문 논문)
Youngsun Kwon. Real-time Dense Occupancy Mapping using Spatial Correlation of Point Clouds. School of Computing. 2022. 61+iv pages. Advisor: Sung-Eui Yoon. (Text in English)

초 록

자율 로봇 및 자동화 시스템에서 변화하는 주변 환경을 이해하는 것이 필수요소이다. 이러한 주변 환경의 기하 정보는 깊이 센서나 라이다를 이용함으로써 점군 데이터 형태로 얻을 수 있다. 점군 데이터를 활용하여 주변 환경을 표현하는 점유 지도는 경로 생성이나 충돌 회피 등 로보틱스 응용 분야의 기반 기술로서 지난 수십 년 동안 여러 방면으로 연구되었다. 그러나, 한 시점에서 획득한 점군 데이터는 주변 공간의 부분적인 기하 정보만을 제공하기 때문에, 동적인 환경에서 실시간으로 정밀하게 점유 정보를 표현하는 기술은 여전히 도전적인 문제로 남아있다.

본 논문은 점군 데이터의 공간 상관성을 활용하여 실시간으로 점유 표현을 업데이트하는 정밀 점유 지도 생성 기술을 다룬다. 구체적으로, 변화하는 환경에서 **실시간으로** 점유 표현을 업데이트하기 위해서, 1) 점유 지도 업데이트 방식의 기하 분석을 통해 알고리즘을 가속하는 기법을 제안한다. 또한, 부분적인 공간 관측 정보로부터 **정밀한** 점유 지도 표현을 위해 2) 점유 관측값의 상관관계를 활용하는 회기 분석 및 3) 점군 데이터의 공간 상관성을 사전 지식으로 학습하는 딥러닝 기반의 점유 지도 생성 방법을 제안한다.

핵 심 낱 말 점유 지도, 공간 상관성, 정밀 표현, 실시간 업데이트, 점군 데이터

Abstract

Understanding a changing environment is one of the essential elements in autonomous robots and systems. A depth sensor or a LiDAR provides the geometric information of the surroundings as a point cloud. As a fundamental technique for robotics applications such as motion planning and collision avoidance, occupancy mapping techniques from sensor data have been studied for decades. However, updating robust representation with real-time processing speed remains a challenging problem of occupancy mapping in a dynamic environment, since the point cloud data of a single scan contains partial geometry observations of the environment.

This dissertation studies the occupancy mapping techniques that update their dense occupancy representations in real-time, exploiting spatial correlation of point cloud data.

Specifically, we propose a method for **real-time** occupancy updates in a dynamic environment; 1) an acceleration algorithm exploiting the geometric update patterns of an occupancy map. Furthermore, we propose two approaches to update the **dense** occupancy representation of the environment given sparse sensor data; 2) regression method using correlation among occupancy observations, and 3) deep-learning network embedding prior knowledge about the spatial correlation of measuring sensor data.

Keywords Occupancy map, spatial correlation, real-time update, dense representation, point clouds

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Many robotic applications, such as motion planning and collision avoidance, use various sensor data for understanding their environments. Well-known sensors (e.g., depth sensors, laser scanners, and LiDARs) are often exploited as a means to estimate the surroundings, and these produce a noisy point cloud that represents a partial geometry of environment. The point clouds, which the sensors have collected over time, can serve as a map representation themselves for the environment. Nonetheless, the point clouds can have an excessive number of points especially for large-scale scenes, and more severely they can have inherent sensor noise. Due to these issues, many prior approaches [1–4] convert point clouds to other representations (e.g., meshes) in order to process them in a simple and efficient way.

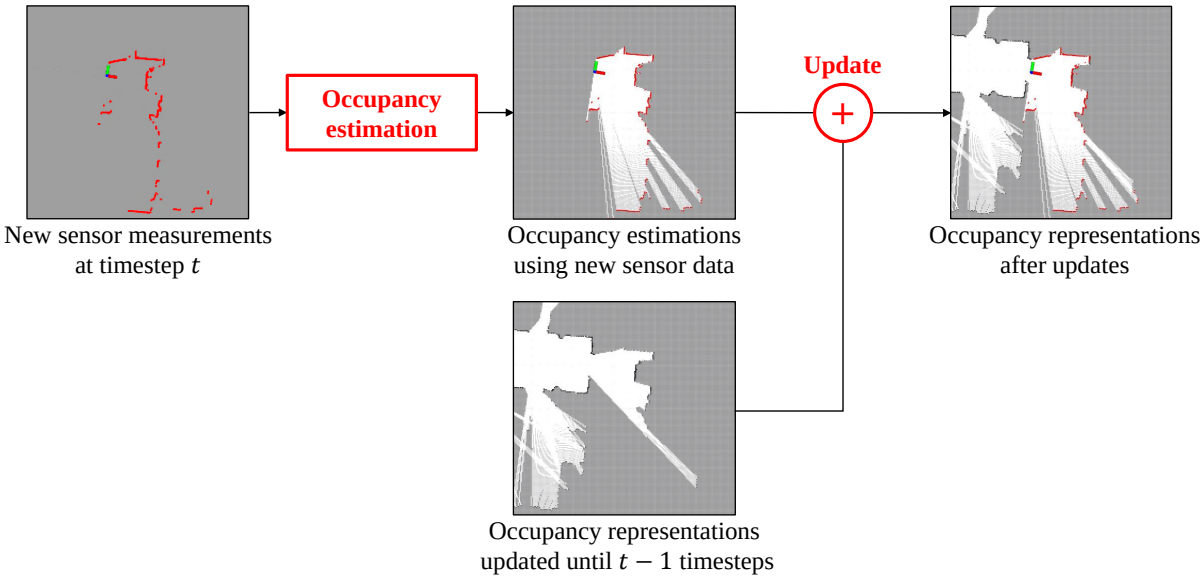
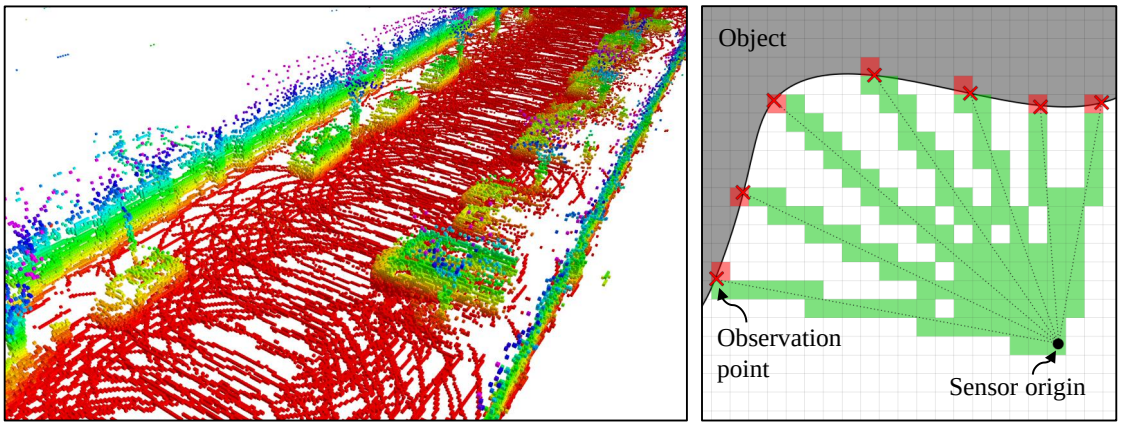


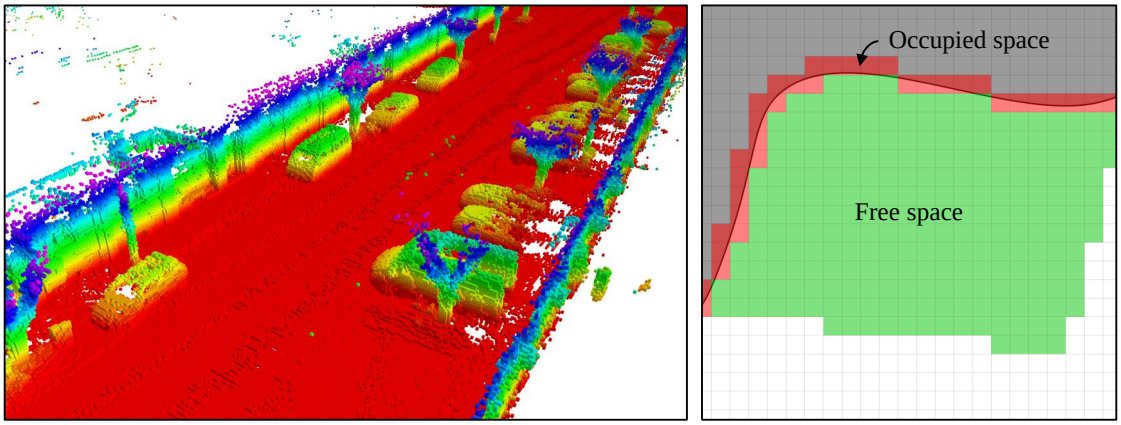
Figure 1.1: On-the-fly occupancy mapping. This figure shows the process of on-the-fly occupancy mapping when a sensor captures new measurements. In a single scan, various models estimate the occupancy states of the sensing area, and then update the occupancy representations of the maps. In this figure, the white to black color represents occupancy probability from the free to occupied states.

As one of the notable examples in the robotics literature, occupancy maps have been widely used as a functional approximation of the changing environment over time. Many robotic systems operate with various dynamic objects such as walking humans or driving cars, and thus require a geometry representation of the dynamic environment for their tasks. A mapping method uses occupancy information to model the environment's geometry in these systems. As shown in Fig. 1.1, an occupancy map updates its occupancy representation when a sensor captures new partial geometry of the environment.

As one of the most well-known approaches, the occupancy grids partition an environment into multiple cells representing the occupancy probabilities. While simple uniform grid maps [5, 6] are proposed early and succeed in various robotic applications, it has certain limitations. Its main drawback is that it requires a tremendous size of memory when we handle large-scale environments or the map has a high resolution for the accurate representation. Tree-based representations such as a quadtree map in 2D and an octree map in 3D have been studied to overcome the problem [7–10]. The octree map divides a 3D



(a) Sparse occupancy representation



(b) Dense occupancy representation

Figure 1.2: Occupancy maps at different density levels. Left captures of each sub-figure show the regions that a map represents as occupied space. The color in the capture represents the relative height of the structure. (a) An occupancy map can have holes representing no occupancy information due to sparse sensor observations. (b) To solve the problem, we exploit a spatial correlation of point clouds in this dissertation. Our approach can predict occupancy states of unobserved regions, resulting in dense occupancy representation of an environment.

space into 8 sub-spaces having the same volume and represents a space with a cell having an occupancy state. When all the children cells have the same states, this map results in a compact representation than the grid map. The volumetric structure enables to improve the performance of downstream applications, i.e., bounding volume of collision detection. Furthermore, this simple representation allows updating occupancy states in real-time via a ray-casting algorithm on grid. However, like Fig. 1.2-(a), sparse point clouds can lead to problematic holes in the resulting maps, where some cells do not contain any occupancy observations.

To overcome this issue, machine-learning based maps have been studied to represent the environment's occupancy as a continuous function, resulting in dense occupancy reconstruction from noisy, sparse sensor data. The maps adopt well-studied inference models to learn an implicit correlation of partial occupancy observations. For example, Gaussian process-based methods [11–17] can provide occupancy estimations with predictive variances, but the heavy time complexity of the regressor remains. The prior methods [18–25] employ a logistic classifier model to overcome the well-known drawback of GP

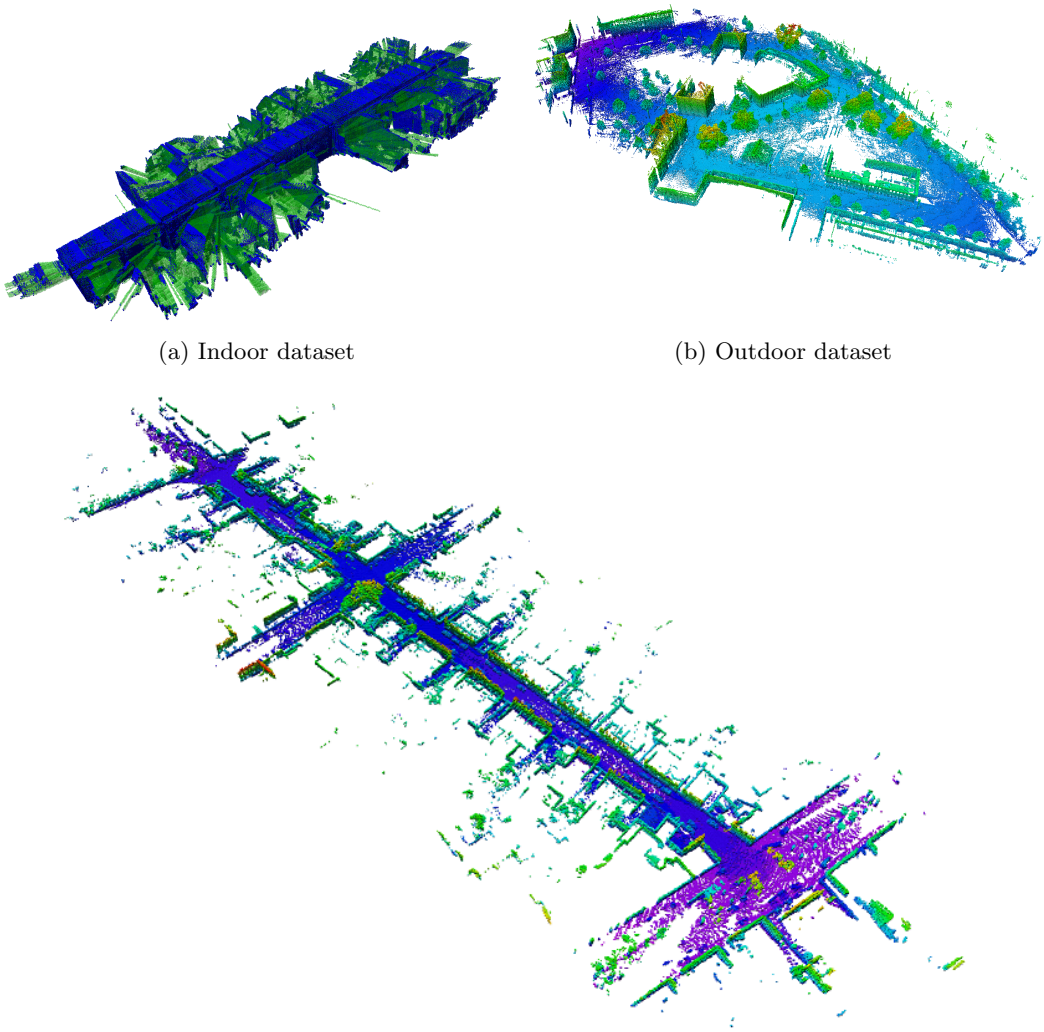
models. The functional occupancy estimators of these maps enable predicting an occupancy state in an unmeasured region, and thus can reconstruct a dense representation of the environment. Nonetheless, the reconstruction process requires a high computational overhead in downstream on-the-fly applications.

Some works [26–29] have been studied to combine the strengths of two approaches; volumetric structure of occupancy grid and occupancy prediction of the learning-based map. These methods train each local occupancy estimator with new sparse sensor measurements, and then incrementally update the estimations to grid cells. The estimators such as the Gaussian process or kernel regression allow the dense representations of occupancy maps despite sparse sensor data. Furthermore, the volumetric approximation can improve performances of robotics applications, while preserving fast map updates.

This dissertation aims to design an occupancy map that **densely** represents the environment's occupancy states and updates its representation **in real-time**, exploiting a spatial correlation of point clouds. Chapter 2 proposes an accelerating algorithm based on geometric update patterns of sensing data for real-time occupancy mapping. Furthermore, Chapter 3 proposes a robust occupancy estimator using non-uniform distributions of sparse occupancy observations in order to achieve dense occupancy representation. In Chapter 4, we also introduce a novel deep-learning network, which learns a spatial correlation of sensing an environment's geometry as prior knowledge. The proposed network improves the density of point clouds and thus results in dense occupancy representation.

Chapter 2. Super Rays and Culling Region for Real-Time Updates on Grid-based Occupancy Maps

2.1 Introduction



(a) Indoor dataset

(b) Outdoor dataset

(c) KITTI dataset

Figure 2.1: Occupancy mapping results. These figures visualize the map representations for three public datasets. (a) Blue and green cubes represent occupied and free spaces, respectively. (b), (c) We use heat colors to represent relative heights for visualizing the occupied cells of the maps.

Real-time robotic applications require to fast process a point cloud acquired over time for reacting to a dynamic environment where obstacles appear suddenly. Grid-based maps are appropriate for these applications thanks to the high update speed. Unfortunately, constructing such occupancy maps out of point clouds can still take a high computation overhead on sensors with high-frequency. We observed a low update speed when using the large numbers of points captured by depth sensors or reconstructed by a deep-learning network (Chap. 4).

In this chapter, we present novel, efficient map update methods based on *super rays* as well as *culling region*, while achieving high performance without compromising the representation accuracy of grid-based maps. Specifically, we propose to use super rays of points as our main update method for maps, where the ray is a common model for extracting geometric information from point clouds; for example, the space between a sensor origin and a point of point cloud has non-collision. A super ray is a representative ray for a set of rays and is constructed in a way that updating the maps with those super rays traverses the same set of cells with original rays. To generate such super rays given input points, we propose to use a mapping line for updating 2D maps (Sec. 2.3.1 and Sec. 2.3.2), and generalize it to 3D maps using the 2D approach (Sec. 2.3.3). Furthermore, we propose a culling region that uses occupancy correlation between scans and reduces redundant computations by stopping unnecessary traversals of rays in advance (Sec. 2.3.5).

To demonstrate the benefits and robustness of our methods, we test our methods and prior works with a variety of cases. We first test the update performance with two public datasets (Fig. 2.1-(a) and (b)) for the grid-based maps such as uniform 3D grid map and octree map. We found that our method combined with super rays and culling region is robust enough to show the performance improvement, 6.3 and 1.8 times improvement across a diverse set of configurations, compared to prior works in the indoor and outdoor scenes, respectively (Sec. 2.4.1). Furthermore, we provide the update speed and the representation accuracy of various mapping algorithms using a public KITTI dataset [30], for discussing the practical benefits of real-time updates. In this test, we found that our combined method can give positive effects to such navigation in practice by showing the best frequency of updates (Sec. 2.4.2).

2.2 Overview

2.2.1 Updates on Grid-based Occupancy Maps

A point cloud consists of points captured by a depth sensor or laser range finder. When the sensor reports a point, it implies that the space between the sensor origin and the point is empty. As a result, we associate a ray with the point starting from the sensor origin. Thus, the problem can be transformed into map traversal along the ray from the sensor origin toward the reported endpoint.

Such a ray provides two kinds of state information about space under the study: occupied and free states. The endpoint of the ray has the occupied state since the sensor reports an object on that particular point. On the other hand, space that the ray passes through has the free state. This information is critical for various applications such as motion planning. Therefore, it is essential to construct an occupancy map accommodating this information acquired from sensors. Unfortunately, data captured by sensors are plagued by various levels of noise. To consider such noise, map representations commonly use an occupancy probability, instead of simple boolean occupancy states of occupied or free.

Grid-based maps such as uniform grid-map or octree-map partition an environment into grid cells representing the occupancy probabilities, and update them to maintain the recent occupancy states of the environment through sensor measurements. Such maps use a ray-casting algorithm to find cells where rays modeled by point clouds traverse on a grid from the sensor origin to the endpoints.

A Ray-casting algorithm such as 3DDDA [31] computes adjacent cells on the uniform grid where a ray traverses from the start to the end cells containing the sensor origin and endpoint, respectively. The cells traversed by rays are updated to have free states, and the end cells are updated to have occupied states.

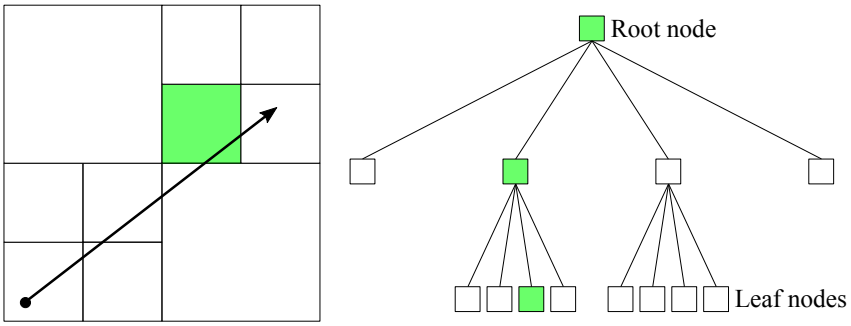


Figure 2.2: Updating a cell's occupancy in a tree-based map. The ray traverses the cell colored by green in the left figure. To maintain the tree structure of the quadtree, we update the occupancy probabilities of all nodes from the leaf to the root, colored by green in the right figure.

On the assumption that cells of a map are independent of occupancy states, the likelihood of occupancy, $P(x|z_{1:t})$, represents the occupied state of a cell, x, given sensor measurements, $z_{1:t}$, from the initial time step 1 to the current time step t, and can be modeled by the Bayes rule and Markov assumption [32] as follows:

$$\frac{P(x|z_{1:t})}{1-P(x|z_{1:t})} = \frac{P(x|z_{1:t-1})}{1-P(x|z_{1:t-1})}\frac{P(x|z_t)}{1-P(x|z_t)}\frac{1-P(x)}{P(x)}. \tag{2.1}$$

For the fast update to the map, a well-known approach using the log-odd notation $\mathrm{L}(x) = \log\left[\frac{P(x)}{1-P(x)}\right]$ transforms the prior equation into:

$$L(x|z_{1:t}) = L(x|z_{1:t-1}) + L(x|z_t) - L(x). \tag{2.2}$$

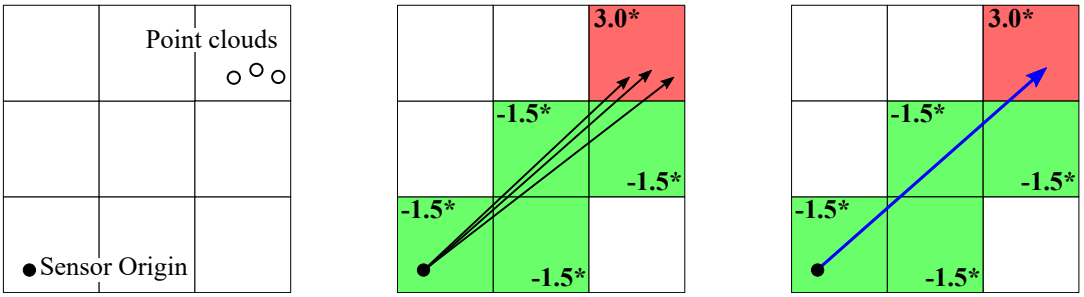
Based on this equation, the OctoMap framework [9] uses a prior probability $P(x) = 0.5$ representing the unknown state and the simple inverse sensor model on the log-odd notation $L(x|z_t)$ defined as the following:

$$L(x|z_t) = \begin{cases} l_{occ}, & \text{if the endpoint of a ray is in the cell,} \\ l_{free}, & \text{if a ray passes through the cell.} \end{cases}$$

As a result, the simple form of Eq. 2.2 results in the efficient update of occupancy probability at a cell by using the fast addition operation.

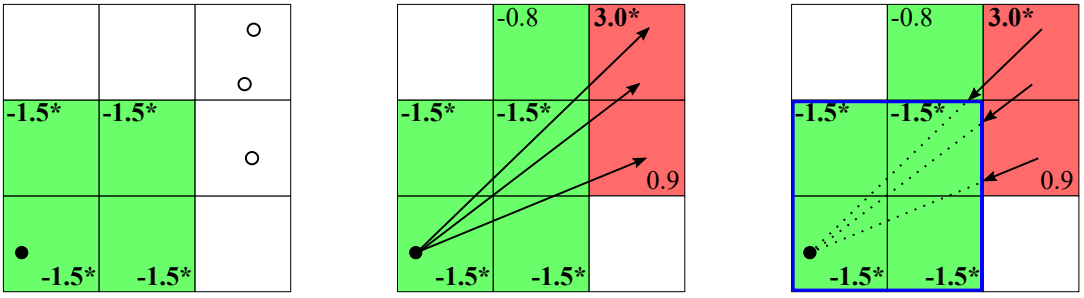
When a cell has an occupancy probability that has been accumulated over long time steps, a new input data that conflicts with the current state of the cell cannot change the state immediately. This over-confidence problem can frequently occur in dynamic environments. Yguel et al. [33] solve the problem by using a clamping policy that limits the occupancy probability of a cell based on the minimum and maximum state bounds: l_{min} and l_{max} for free and occupied states, respectively. The state of a cell limited by either one of those two bounds is considered to be fully free or fully occupied with a high occupancy probability. Fig. 2.3 and Fig. 2.4 show illustrations of updating the grid map in 2D given point clouds.

Hierarchical representations. A uniform grid map consists of cells having the same size determined by a user-defined resolution. We find a set of traversed cells of rays using a ray-casting algorithm, and update their occupancy probabilities using the update rule (Eq. 2.2). Using the uniform grid with the ray-casting updating method is intuitive and straightforward to represent occupancy states of an environment, but can require a vast size of memory, especially when we have a high resolution. To



(a) Before a new update (b) State-of-the-art (c) Super Ray

Figure 2.3: An overview of our super ray when we have the new measurements as shown in (a). (b) and (c) represent occupancy probabilities of cells after updating the 2D grid map with different methods. The green and red cells have free and occupied states, respectively. The bold numbers with * notation in cells indicate that those cells are classified into fully occupied or fully free state. In (b), the state-of-the-art method updates the same set of cells for three different rays, which causes redundant computation on overlapped traversals on the cells. The blue ray in (c) is a super ray computed out of those three rays in (b). The super ray updates the map with a single traversal on the cells. In this figure, we use $l_{occ} = 1.7$, $l_{free} = -0.8$, $l_{max} = 3.0$, and $l_{min} = -1.5$.



(a) Before a new update (b) State-of-the-art (c) Culling Region

Figure 2.4: An overview of our culling region, given the new measurements as shown in (a). In (b), the prior method causes redundant computation on traversals on the cells having fully-free states. The blue box in (c) is a culling region that prevents the three rays to traverse the fully-free cells for updates. In this figure, we use the same setting with Fig. 2.3.

overcome the problem, a quadtree map in 2D or an octree map in 3D are proposed to have the tree data structure for providing various resolutions, resulting in a more compact representation. Such maps merge 4 or 8 sub-divided children nodes that have the same occupancy probabilities into one parent cell representing a space at a coarser level (Fig. 2.2). The properties of the tree structure can be used efficiently for collision detection or motion planning, although it generates the hierarchical update from the leaf to all the parent cells unlike the grid map. The recent OctoMap [9] framework avoids the duplicated updates made from updating the tree structure. The work is done by batching leaf cells that all rays traverse on the uniform grid with the maximum resolution, before updating the maps. The batching-based method updates tree-based maps in a single time using the batched cells, resulting in the performance improvement.

2.2.2 Motivations

Grid-based occupancy maps have been widely used for various applications. We, however, found that updating these maps can take a huge amount of computation time compared to the frequency of sensor measurement. Furthermore, we have identified that the original update method for occupancy maps has redundant computations, because of the discrete nature of grid-based representations. For example, Fig. 2.3-(b) shows three different rays traverse the same set of cells in the 2D grid, while these rays have different endpoints. When we update the map with these rays one-by-one, duplicate computations are made on traversal and updating through the exact same set of cells, resulting in lower performance.

Additionally, certain traversals do not contribute at all to cells whose occupancy probabilities are out of range of the min and max bound values due to the clamping policy, as shown in Fig. 2.4-(b). We define the traversal as an unnecessary traversal. These problems frequently occur because the original update method does not consider the discrete nature and occupancy states of map representations.

2.2.3 Overview of Our Approaches

To overcome the problems aforementioned above, we first propose an update method utilizing super rays for occupancy maps. We define a super ray as a representative ray to rays associated with given points (Fig. 2.3). The super ray is constructed in a way that traversing those rays for updating the map requires to access the same set of cells in the map. We then update the map by traversing those cells with the super ray only a single time, while considering the number of points associated with the super ray, thus removing redundant computation and achieving higher performance.

On top of super rays, we propose a culling region-based method by culling out traversal on cells having already saturated occupancy probabilities to fully-free states. We define a culling region as a set of cells where traversals from those cells to the sensor origin are unnecessary (Fig. 2.4). If rays encounter the culling region while traversing from their endpoints to the sensor origin, our method stops the remaining traversals from a cell entering the culling region to the sensor origin. As a result, we can achieve a higher performance thanks to removing unnecessary traversals.

Our two approaches are orthogonal and easily applied to the 3DDDA and the batch based methods on grid-based occupancy maps. Ours improve the performance of updates compared to the prior work, and these two methods complement each other as shown in the result section (Sec. 2.4). Overall, our method combined both with super rays and culling region shows the best performance on average.

2.3 Updates using Super Rays and Culling Region

In this section, we explain our approaches in detail. We first propose a mapping line and explain how to use it for generating super rays starting from a single, seed frustum containing all the points of a cell in the map (Sec. 2.3.1). We then identify which points in a cell have the same set of traversed cells for updating the map based on the mapping line (Sec. 2.3.2). To extend the concept of a 2D case into a 3D case, we introduce a mapping plane conceptually used for generating super rays in the 3D case and then explain how to solve the 3D problem efficiently using mapping lines of the 2D case (Sec. 2.3.3). We then explain how to update cells that each super ray passes without compromising the representation accuracy of the map (Sec. 2.3.4). In the end, we also propose to use a culling region for improving performance by considering occupancy states of the map (Sec. 2.3.5).

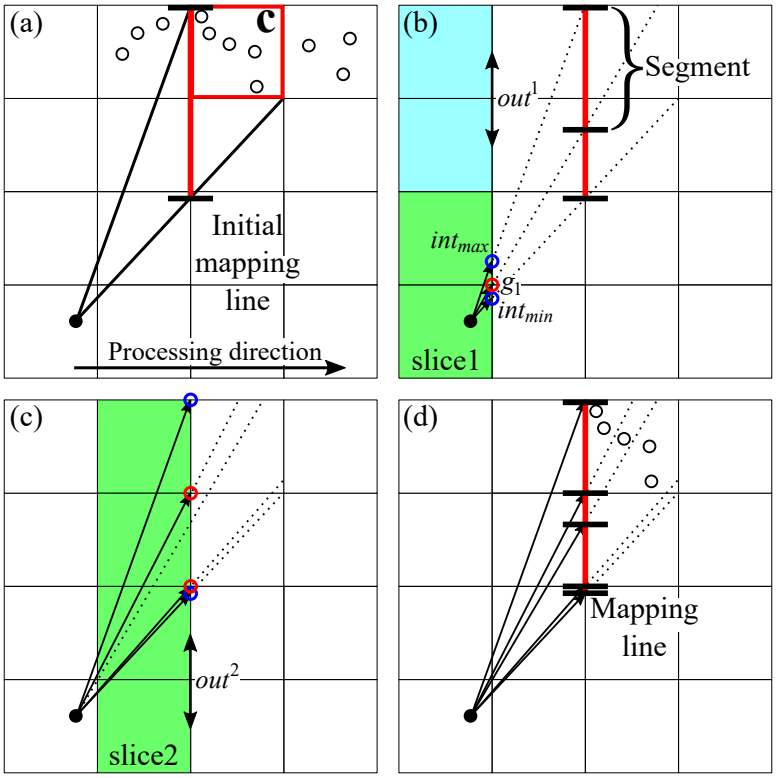


Figure 2.5: An example of generating a mapping line for a cell c. The red grid point g_1 in (b) divides the seed frustum into two sub-frustums, and its projected point generates two segments on the mapping line. In (c), two grid points in out^2 in the slice 2 also generate two more segments in the mapping line shown in (d).

2.3.1 Generating a Mapping Line

In general, point clouds are defined in the sensor coordinate system, while occupancy maps model them in the world coordinate. Based on the assumption that we know the position and orientation for the sensor in the world coordinate, we transform point clouds from the sensor coordinate to the world coordinate and update the map with them.

For each cell, c, in the map, we conceptually construct a seed frustum starting from the sensor origin to the cell box containing all the points in the cell c, the red box shown in Fig. 2.5-(a). Starting from the seed frustum, we partition it into multiple ones, each of which traverses the same cells of the map. To do this, we design our algorithm to access grid cells slice-by-slice, where a slice contains cells in a line for the 2D data. For this process, we pick an axis, i.e., X or Y axis, for computing such slices and treat it as a processing direction. Fig. 2.5 shows a case that X axis is the processing direction.

For identifying which points are mapped to the same super ray, we introduce a *mapping line*, which is a line segment that overlaps between the cell c and the slice containing the cell c. Fig. 2.5-(a) shows an initial mapping line. Each segment of the mapping line corresponds to one of the super rays, while we also use the terms of frustums or super rays conceptually to explain our geometric concepts. The initial mapping line starts with a single line segment representing a super ray, but it can be divided into multiple segments corresponding to multiple super rays.

One key observation for generating the mapping line to represent different frustums is that the traversal patterns of cells differ along grid points, when we consider cells slice-by-slice. Fig. 2.5-(b)

Algorithm 1 BUILD MAPPING LINE

Require: C_{box}, a cell box in 2D, O, a sensor origin in 2D

Ensure: M_{line}, a mapping line

```
M_line ← InitMappingLine(C_box)
S_slice ← InitSlices(O, C_box)
for i = 1 to length(S_slice) − 1 do
    g ← ComputeGridPoints(S_slice[i], C_box)
    for j = 1 to length(g) do
        M_line.insert(Projection(g_j))
    end for
end for
return M_line
```

shows a grid point, shown in the red circle within the initial frustum. Given the grid point, the traversed cells differ, and thus we need to partition the seed frustum into two different ones, resulting in two segments on the mapping line (Fig. 2.5-(b)).

Based on this observation, the key operation is how we efficiently generate the mapping line within the frustum. Let out^i of i-th slice to denote the faraway line of the slice along the processing direction. Fig. 2.5-(b) shows an example of out^1 for slice 1. We can then compute the two intersection points, int_{min} and int_{max}, of the seed frustum for each i-th slice like the blue circles in Fig. 2.5.

Suppose that the first slice containing the sensor origin is slice 1 and the last slice containing point clouds is slice N. Our algorithm of generating a mapping line works in an iterative manner from slice 1 to slice $N-1$. To find grid points that differentiate the traversal pattern, we first compute two points, int_{min} and int_{max}, within out^i of each slice starting from slice 1. We then project all the grid points between int_{min} and int_{max} onto the mapping line. Suppose that there are m grid points, $g = \{g_1, g_2, \cdots, g_m\}$. These grid points partition the current frustums into at most $m + 1$ sub-frustums, resulting in $m + 1$ corresponding segments on the mapping line. Each pair of two consecutive points projected onto the mapping line implicitly defines a segment and its associated frustum (and its super ray). Note that we can easily find these grid points and compute segments of the mapping line thanks to the discrete nature of occupancy maps, resulting in a fast update method. The pseudocode of generating a mapping line for a seed frustum is shown in Alg. 1.

2.3.2 Generating Super Rays using the Mapping Line

After we generate the mapping line of each cell at the prior step, we use it for computing which rays should be merged into the same super ray. To perform this process, we map all the input rays onto the mapping line and count how many rays are assigned to each segment of the mapping line (Fig. 2.6).

The rays assigned to the same segment of the mapping line have the same traversal patterns in terms of cells traversed for updating the map. We can merge the rays into a single super ray with a weight as the number of merged rays. We use the weight information associated with the super ray to efficiently update the occupancy map without losing the representation accuracy, because the super ray traverses all the same set of cells where its associated rays traverse. We skip here proving the completeness of using the mapping line to classify rays traversing the same set of cells. Instead, we show the completeness of our algorithm as Theorem 1 in Appendix.

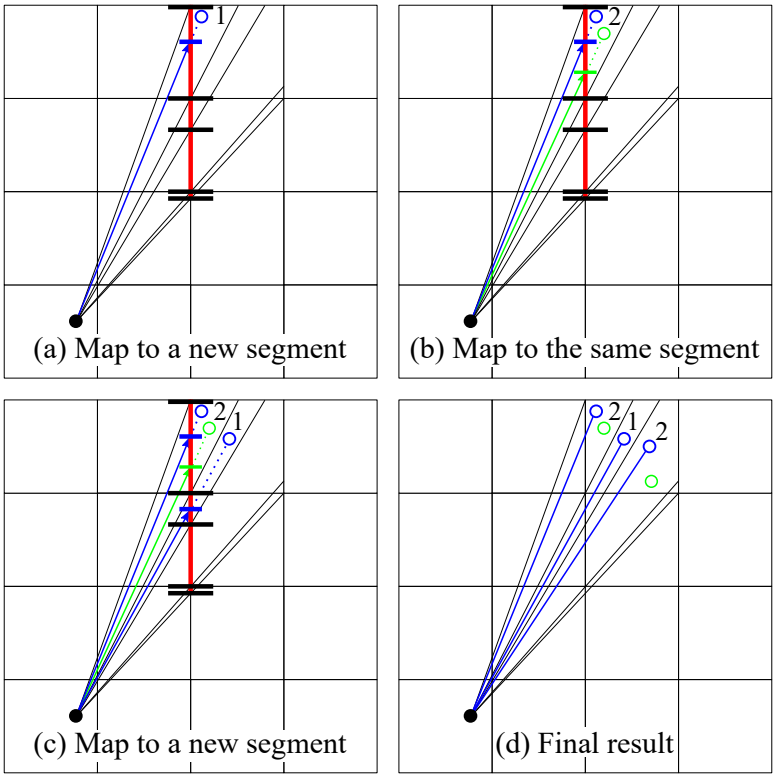


Figure 2.6: A process to generate three different super rays out of five rays using the mapping line. (a) A new ray maps to a new segment, and we treat it as a new super ray with a weight of one. (b) Another ray maps to the prior segment, and we increase its weight to two. (c) The new ray maps to a new segment, and a new super ray is assigned to it. (d) The figure shows the final, three super rays with their weights.

2.3.3 Extension to the 3D Case

In this section, we explain how we extend our prior 2D approach into handling 3D point clouds. We first introduce a concept of *mapping plane* for generating super rays in the 3D case. Different regions in the mapping plan indicate different traversal patterns. We then propose to use three orthogonal mapping lines defining such different regions in the mapping plane, to handle the 3D case efficiently. For the sake of clarity, these corresponding concepts in 2D and 3D cases are summarized in Table 2.1.

Similar to the 2D case, we first compute a bounding volume containing point clouds in the map representation. We also construct a seed frustum traversing to the volume and then partition the frustum into sub-frustums, each of which traverses the same set of cells.

Table 2.1: Corresponding concepts or 2D and 3D cases.

Case	2D	3D
Data structure for generating super rays	Mapping line	Mapping plane
Element of the mapping data structure	Segment	Region
Geometric entity differentiating traversal patterns of rays	Grid points	Edges of cells

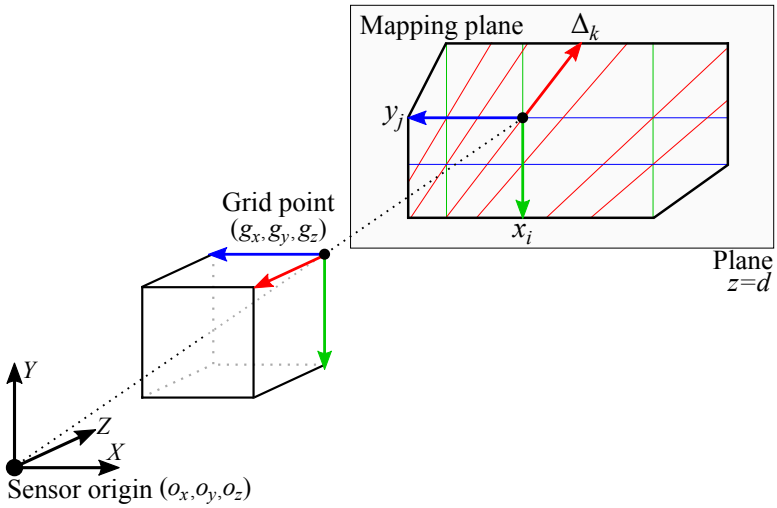


Figure 2.7: An example of a mapping plane on the plane $z = d$ in the 3D case. The projected lines, which three edges (red, green, and blue) of all grid points are projected to, partition the plane into regions, each of which is associated with a unique traversal pattern.

The key observation for the 3D case is that traversal patterns of cells differ along edges of cells, not just at the grid points. Based on the observation, we can partition the seed frustum into sub-frustums, each of which has the same traversal pattern and thus is constructed as a super ray. Fig. 2.7 shows an example of the mapping plane consisting of regions associated with super rays. In this example, we pick one of the planes that are orthogonal to axes, e.g., $z = d$, and explain our approach based on this example plane for the sake of clarity; other mapping planes can be treated in a similar manner. As you can see in Fig. 2.7, those projected lines, i.e., red, green, and blue lines, in the mapping plane creates many complex regions.

Conceptually, we can use the mapping plane for generating super rays in a similar manner to using the mapping line for the 2D case described in Sec. 2.3.2. Simply speaking, we map all input rays onto the mapping plane and count how many rays are assigned to each region of the mapping plane. The rays assigned to the same region of the mapping plane have the same traversal pattern in terms of cells traversed for updating the map. Using the mapping plane, we can merge multiple rays into a single super ray. The completeness of our algorithm using the mapping plain to generate super rays is given as Theorem 2 in Appendix.

We now go into details of our 3D approach using mapping plane to generate super rays, starting from introducing geometric values shown in Fig. 2.7. Let (x_i, y_j) be a 2D point, where the 3D grid point (g_x, g_y, g_z) is projected onto the plane $z = d$. They are expressed as follows:

$$x_i = \left(\frac{d - o_z}{g_z - o_z}\right)(g_x - o_x) + o_x, \tag{2.3}$$

$$y_j = \left(\frac{d - o_z}{g_z - o_z}\right)(g_y - o_y) + o_y, \tag{2.4}$$

where the line $x = x_i$ partitions the plane along the X axis, since a ray mapped on the left side of the line has a different traversal pattern to another ray mapped on the right side. The line $y = y_j$ partitions the plane along the Y axis, in the same manner. Note that the value x_i is computed only with X and Z coordinates of the grid point, without the Y coordinate. As a result, the line $x = x_i$ can be computed in the 2D $X - Z$ space. Another partitioning line $y = y_j$ is treated similarly in the $Y - Z$ space.

Algorithm 2 GENERATE SUPER RAYS

Require: P, a set of points in a cell, O, a sensor origin

Ensure: S_{ray}, a set of super rays

1: $C_{box} \leftarrow ComputeCellBox(P)$
2: $M_{xy} \leftarrow BuildMappingLine(C_{box}(X,Y), O(X,Y))$
3: $M_{yz} \leftarrow BuildMappingLine(C_{box}(Y,Z), O(Y,Z))$
4: $M_{zx} \leftarrow BuildMappingLine(C_{box}(Z,X), O(Z,X))$
5: $S_{ray} \leftarrow GenerateSuperRays(M_{xy}, M_{yz}, M_{zx}, P)$
6: **return** S_{ray}

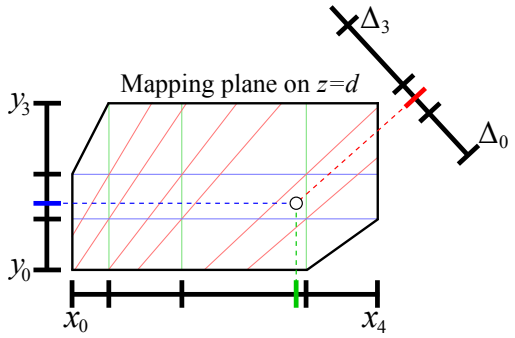


Figure 2.8: A process to generate super rays using a mapping plane. In this example, we project a ray on the region of the mapping plane for finding the traversal pattern of a ray.

For the red edge projected in the mapping plane $z = d$ shown in Fig. 2.7, it partitions the plane in a slant line. Its gradient, Δ_k, is computed by the slope from the sensor origin (o_x, o_y) and the projected point (x_i, y_j):

$$\Delta_k = \frac{y_i - o_y}{x_j - o_x} = \frac{g_y - o_y}{g_x - o_x}. \tag{2.5}$$

While the slant line seems to behave differently from other orthogonal lines, we can know that the gradient of the slant line consists of only X and Y coordinates without the Z coordinate, indicating that this can be treated in the 2D $X - Y$ space.

We now explain the geometric interpretation of utilizing the mapping plane to generate super rays shown in Fig. 2.8. Let x_i and x_{i+1}, y_j and y_{j+1}, and Δ_k and Δ_{k+1} be the X, Y, and gradient values of lines, respectively, which construct the smallest region of the mapping plane. To classify rays of point clouds having the same traversal patterns and then generate super rays, we map all the rays into one of the regions on the mapping plane. This classification task of a ray can be expressed as:

$$\begin{cases} (a) x_i \le x_p < x_{i+1}, \\ (b) y_j \le y_p < y_{j+1}, \\ (c) \Delta_k \le \Delta_p < \Delta_{k+1}, \end{cases} \tag{2.6}$$

where (x_p, y_p) is a projected point of a ray onto the mapping plane and Δ_p is a gradient between the sensor origin and the projected point.

Implementation of generating super rays. We use Eq. 2.6 to classify rays with the same traversal patterns and generate super rays. Eq. 2.6 consists of three sub-tests that each of them is computed with two of three coordinates, as we mention in Eq. 2.3, Eq. 2.4 and Eq. 2.5. Each sub-test is identical to process finding segments of mapping line for generating super rays in 2D. For example,

Eq. 2.6-(a) uses X and Z coordinates without Y coordinate as can be seen in Eq. 2.3, and has the same formulation to finding segments of mapping line in $X - Z$ space. As a result, we can classify rays with the same traversal patterns by finding rays projected into all the same segments of three mapping lines in sub-spaces. Using this fact, we can implement our method to generate super rays in 3D using our 2D approach we introduce in Sec. 2.3.1. The pseudocode of generating super rays for a cell in the 3D case is shown in Alg. 2.

2.3.4 Updating Occupancy Map using Super Rays

To update occupancy maps with computed super rays, we use existing update methods with a minor modification. To determine cells needed for the update, we use the 3DDDA based algorithm [31]. Because all the points in a super ray update the same set of cells of a map, we traverse and update those cells in only a single traversal. Since a super ray is generated for multiple points, we take account of the weight of the super ray w (the number of contained points), and use the following, modified inverse sensor model:

$$L(x|z_t) = \begin{cases} w \cdot l_{occ}, & \text{if the endpoint of a super ray is in the cell,} \\ w \cdot l_{free}, & \text{if a super ray passes through the cell.} \end{cases}$$

It is then guaranteed that we achieve the same occupancy map to that computed by processing points individually with multiple traversals.

Batching based updates. For a high performance of updating tree-based occupancy maps, we use a batching technique implemented in OctoMap [9]. The batching method reduces the number of repeated accesses to cells from a leaf to the root for updating the occupancy probability of the leaf cell. The method batches cells that rays traverse, and then updates tree-based maps in a single time using the batched cells. This technique shows good performance in tree-based maps, but the time for batching the cells depends on the overhead of finding such cells. Fortunately, super rays can reduce the cost of the batching process thanks to a single traversal of super ray, instead of multiple traversals of points (Sec. 2.4.3).

2.3.5 Culling Region based Updates

In this section, we propose another approach, culling region based update method, which utilizes occupancy states of maps updated by previous scans. The method increases the performance of super rays when we use an occupancy map with a high resolution. The number of generated super rays can be similar to the number of points. Therefore, we cannot maximize the benefits of using super rays at such high-resolution maps. To overcome the issue, we propose its complementing method, a culling region based update method, for achieving a robust performance even with high resolutions.

As we mentioned in Sec. 2.2.1, the clamping policy prevents occupancy maps from having the over-confidence problem. This thresholding technique makes the maps to support dynamically changing environments. Given this clamping policy, we observe that some traversals of a ray do not affect occupancy probabilities of cells when those probability updates are limited by the thresholds (Fig. 2.4). Before a new update, cells of the map can have occupancy states updated during previous time steps. For updating the map with a new point cloud, the rays generated from the new sensor data traverse and then update occupancy probabilities of the map. However, some of the traversals do not change the occupancy states of the map. We aim to reduce such unnecessary traversals of a ray for achieving higher update performance.

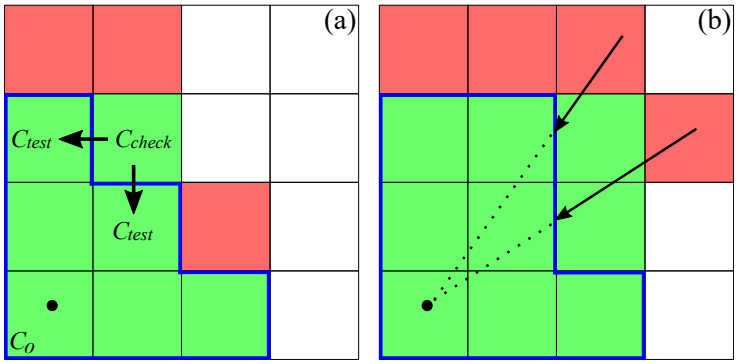


Figure 2.9: Examples of building and using our culling region. The blue outline represents a culling region, and the fully free cells are shown as green cells. (a) Our test checks whether a cell C_{check} can be inserted into the culling region or not, during the process to build the region. We insert the cell C_{check} into the culling region because both two neighbor cells C_{test} are in the culling region and C_{check} itself is in the fully free state. (b) We show the updated map with new measurements, while the generated culling region allows skipping the remaining traversals of the rays represented by the dotted lines.

Based on the observation, we define a culling region as a set of cells, where traversals from a cell in the region to the sensor origin are unnecessary. By utilizing the property of the culling region, we propose an efficient method to build and use the culling region for map updates without redundant computations. Our culling region-based method consists of two steps per scan: 1) building the culling region by using the occupancy states of the map as described in Alg. 3 (Fig. 2.9-(a)), and 2) updating the map by utilizing the generated region that reduces the unnecessary traversals (Fig. 2.9-(b)). Note that for supporting a dynamic environment, our approach efficiently constructs and re-initializes a new culling region per scan.

Building the culling region. At the given map, we first initialize and construct the culling region by checking whether a cell satisfies the properties of culling region or not. A naive approach would be to consider a frustum generated from a cell to the sensor origin and to check whether all the cells of the frustum are in the fully free states. Nonetheless, this naive approach may require a high computational overhead, which can even lower down the overall performance at the worst case.

We therefore propose a method of identifying the culling region that incrementally utilizes our simple tests, instead of the naive and time-consuming method. Our approach makes a culling region by incrementally extending the region from the given C_O cell containing the sensor origin at new measurements. If the origin cell has a fully free state, we insert the cell into the culling region because the origin cell guarantees the properties of the region. After inserting the origin cell into the culling region, we then prepare to extend the region using our simple tests on the origin's neighbor cells (line 2:4 in Alg. 3).

In our approach, we define a test cell, C_{test}, to be one of the neighbor cells of the current cell, C_{check}, which is located right outside of the current culling region. As a result, the test cell has the shorter Manhattan distance to the origin cell than the distance between the current cell and the origin cell. These cells are shown in Fig. 2.9 and thus we have: $dist(C_{test}, C_O) \equiv dist(C_{check}, C_O) - 1$ and $dist(C_{check}, C_{test}) \equiv 1$. In other words, the neighbor cells C_{test} for the test are candidate cells that a ray passing through the current cell C_{check} can traverse in the next step toward the origin cell (Fig. 2.9-(a)). According to the definition of test cells C_{test}, we can have up to two and three test cells in the 2D and 3D cases, respectively (line 8 in Alg. 3).

Algorithm 3 BUILD CULLING REGION

Require: C_O, the cell containing a sensor origin

Ensure: CR, a culling region

1: $CR = \emptyset, queue = \emptyset$
2: **if** C_O is fully free **then**
3: $CR.insert(C_O)$
4: $queue.push$(neighbor cells of C_O)
5: **end if**
6: **while** $queue$ is not empty **do**
7: $C_{check} \leftarrow queue.pop()$
8: $C_{test} \leftarrow Compute\ Test\ Cells(C_O, C_{check})$
9: **if** C_{check} is fully free $\wedge$ all the C_{test} are in CR **then**
10: $CR.insert(C_{check})$
11: $queue.push$(neighbor cells except C_{test})
12: **end if**
13: **end while**
14: **return** CR

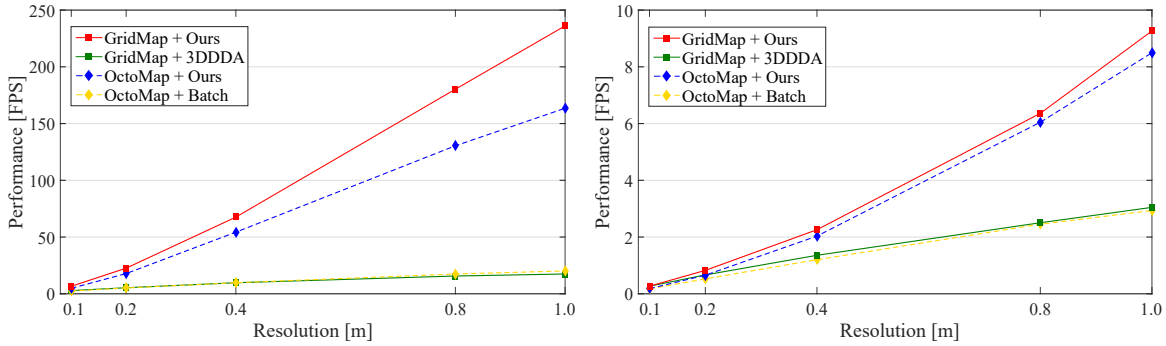
When C_{check} does not have a fully free state, we should update the cell C_{check}. When C_{check} is in the fully free state, we check whether it can be included in the current culling region or not. For doing that, we can simply check whether its test cells C_{test} are in the culling region or not. When C_{test} are in the culling region, it is guaranteed that all the traversal from C_{test} are unnecessary given the definition of the culling region.

If the current cell C_{check} passes such simple tests (line 9 in Alg. 3), traversals of a ray from the current cell to the origin cell are guaranteed to be unnecessary under the clamping policy. We therefore insert the C_{check} into the culling region and then continue the simple tests on neighbor cells that have not been tested (line 10:11 in Alg. 3). Finally we make the culling region efficiently through such simple tests in the 2D as well as 3D cases. Note that the culling region constructed by our method has a convex shape, surrounded by cells that do not have fully free states.

Updating the map using the culling region. Our approach uses the generated culling region to remove the unnecessary traversals of rays. We make a ray traverse from its endpoint to the sensor origin, which is different from the common direction of traversal. If a ray encounters a cell of the culling region, our method can stop and terminate the remaining traversals from those cells to the sensor origin, as shown in Fig. 2.9-(b). We therefore skip the updates for remaining traversals which do not affect the occupancy probabilities of a map, and then only update the traversed cells outside of the culling region. As a result, our approach achieves the high update performance with preserving the occupancy representation of the map.

2.4 Results and Discussions

We mainly test our update methods and others on grid-based maps against two datasets, indoor and outdoor datasets, used in OctoMap [9]. The indoor dataset consists of 66 scans captured in a corridor (Fig. 2.1-(a)), and the outdoor dataset consists of 81 scans captured in a campus (Fig. 2.1-(b)). Scans of the indoor and outdoor datasets have point clouds consisting of 89,446 points and 247,817 points on



(a) Indoor Dataset (b) Outdoor Dataset

Figure 2.10: The average performance, Frame Per Second (FPS), in two scenes according to various resolutions. Note that *Ours* represents the combination of our two methods using both super rays and the culling region. The solid and dashed lines represent the performances of each method on GridMap and OctoMap, respectively.

average, respectively. We first give the update performances of various methods on grid-based maps in the indoor and outdoor scenes (Sec. 2.4.1), and then discuss the issues related to our methods in the following sections (Sec. 2.4.3 and Sec. 2.4.4).

Furthermore, we test the grid-based maps as well as learning-based maps by using a new public dataset, KITTI [30], to show the performance of updates as well as the accuracy of mapping (Sec. 2.4.2). The KITTI dataset[1] that we use consists of 395 scans captured by a 3D laser scanner and has 119,801 points in a scan on average (Fig. 2.1-(c)).

We first introduce implementation details of optimizing our methods, followed by comparisons over prior methods. Note that in our earlier version of this work, we had tested super rays with the datasets using a single core. In this chapter, we implement the test methods by applying parallel computation with 12-threads to updating maps as well as generating super rays.

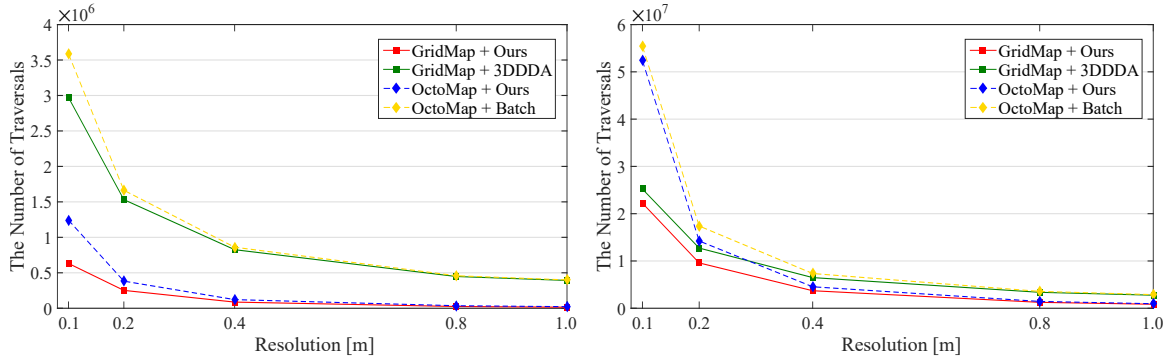
Implementation detail of super rays. Our super ray-based method has a pre-processing cost induced by generating super rays, while it is designed for an efficient process. At the worst case, each super ray can have only a single point, demonstrating only the overhead of our method without any benefits. To prevent such a case, we use a threshold value, k, as the minimum number of points in a cell for generating super rays. We set the value to 20 for all experiments and found that the threshold is enough to handle the problem. The detailed discussion about super rays is shown in Sec. 2.4.3.

Implementation detail of the culling region. Our culling region-based method finds fully-free cells on the updated map and checks whether those cells can be inserted into the culling region or not. In the worst case in this process, we can insert some cells that do not trigger culling, and thus spend unnecessary time on computing those cells. This redundant computation occurs outside cells of the sensor's field-of-view. Therefore we limit a range of the culling region using the input sensor origin and the point clouds.

2.4.1 Performance comparison for update methods

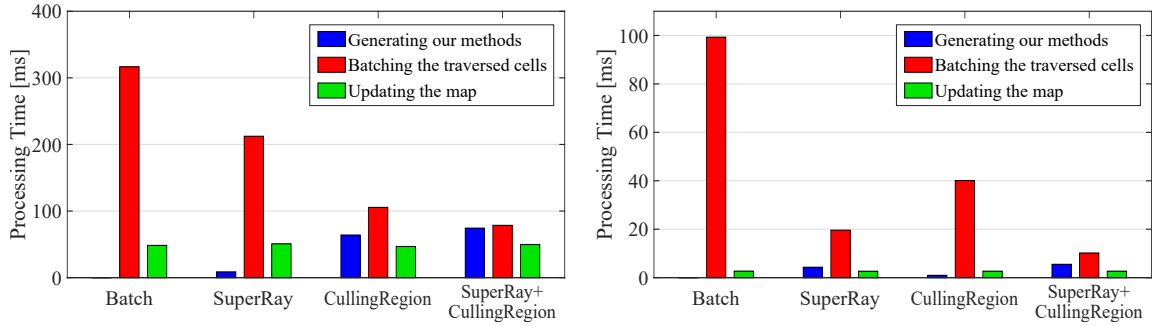
In the following experiments, we compare our method against the 3DDDA and batch based update methods on GridMap, the uniform grid map in 3D, and OctoMap, the octree-based occupancy map, respectively. The overall performance of our methods includes all the processing time for updates such

[1] This dataset has the name, 2011_09_26_drive_0039, in the residential category



(a) Indoor Dataset (b) Outdoor Dataset

Figure 2.11: The number of traversals on average of available scans in two datasets according to various resolutions. The reported results are related to the performance graph, Fig. 2.10.



(a) Indoor dataset with 0.1 m resolution (b) Indoor dataset with 0.4 m resolution

Figure 2.12: The processing time that four different methods spend for each process on OctoMap with 0.1 m and 0.4 m resolutions. Overall, our methods give the high performance improvement compared to the batching based method, despite the computation time to generate super rays or culling region.

as the generation time of both super rays and the culling region; we analyze the performance of each component, super rays and culling region, of our method in the following sections. For all the experiments, we use the same settings of resolutions and parameters used in the prior work [9]; $l_{occ} = 0.85$ and $l_{free} = -0.4$, which are the log-odds values of occupancy probabilities to update cells of maps, and $l_{min} = -2$ and $l_{max} = 3.5$, the log-odds values of the clamping policy.

We measure all the computation time of generating super rays, building the culling region, and updating the maps for both indoor and outdoor datasets with various resolutions, and report the average frame (scan) per second (FPS) computed with all the available scans in Fig. 2.10. As shown in the graph, our method shows the highest performance in most of the tested cases. In the indoor scene, we achieve 7.7 times and 5.3 times faster performance compared to the 3DDDA based method on GridMap and the batch based method on OctoMap, respectively. In the case using outdoor dataset, our method shows the 1.9 times performance improvement for updates on average across resolutions, compared to the prior works on both GridMap and OctoMap.

To analyze reasons for achieving such overall performance improvements, we also measure the number of traversed cells for updates. In the case of the tree structure, OctoMap, we count the number of cells updated from the leaf to all the parent nodes. As shown in Fig. 2.11-(a), our proposed method reduces the number of traversals by a factor of 13.0 times and 9.0 times on average in the indoor scene,

compared to the 3DDDA based method in GridMap and the batch based method in OctoMap respectively. In the outdoor scene, our method removes about half of traversals for updates compared to the prior works, as reported by 2.0 times and 1.9 times reductions on average in GridMap and OctoMap (Fig. 2.11-(b)). As a result, it enables a significant decrease in the update time of our method. The detailed results are reported in Table 2.4.

Note that our method provides the same maps to those computed by the 3DDDA or batch based updates, since our method does not sacrifice any accuracy of the grid-based maps. Additionally, we also measure numerically how well our method updates occupancy probabilities compared to the prior methods. For this purpose, we measure mean squared errors of occupancy probabilities between our occupancy map and the map updated by the prior methods. We verify that the numerical errors turn out to be zero across all the tested settings.

For the specific analysis of our methods, we report the time breakdown of processes for each method on OctoMap with 0.1 m and 0.4 m resolutions in the indoor scene. Fig. 2.12-(a) represents that our approach using the culling region shows the better performance than using super rays in the map with the high resolution (0.4 m). As shown in the blue bars of the figure, the culling approach spends 63.9 ms to generate the culling region, while the method using super rays reports less time 8.7 ms to generate the super rays. However, the culling region achieves a much larger benefit, 211 ms decrease in the batching process compared to the prior work, than 104 ms decrease of using super rays.

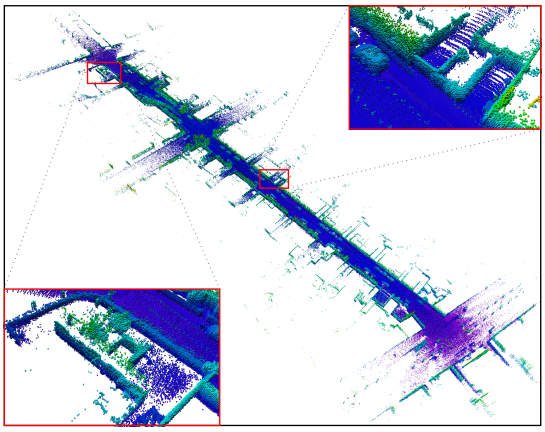
Unlike the case of the high resolution, the super ray-based method shows the better performance than the culling region-based method for the low resolution case (Fig. 2.12-(b)). In this case, the culling region achieves the 59 ms time reduction on the batching process despite the 0.9 ms time consumption to generate the culling region. Using super rays, on the other hand, reduces 80 ms on the batching process with 4.3 ms time spent on generating super rays.

As shown in Fig. 2.12, our combined method using both super rays and culling region shows the best performance compared to the prior work and our methods using only one of them. Our two update methods using super rays and culling region improve the update performance utilizing different types of information. Super ray based updates consider geometric relations among point clouds in the current scan. In other words, this method generates a super ray by grouping points associated with rays traversing the same set of cells. On the other hands, the culling region based method utilizes occupancy states of the updated map. This culling method skips rays whose updates on related cells are conservatively identified to be unnecessary. As a result, these two different methods complement to each other and combining them together shows the best performance.

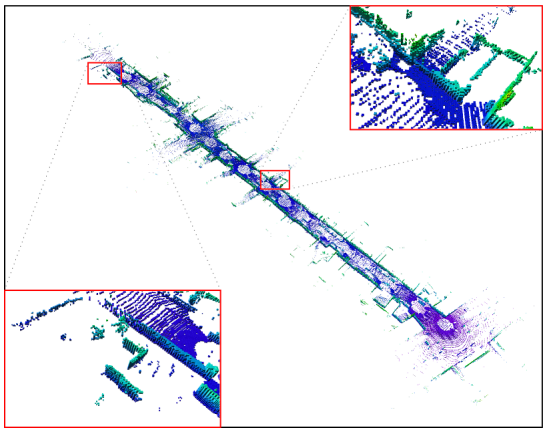
2.4.2 Comparison of mapping algorithms

Recently, learning based approaches have been proposed to learn a correlation of occupancy observations and to predict occupancy states of unobserved regions. These methods handle the mapping problem using classification and regression. In this section, we compare the performance in an aspect of the update speed as well as the representation accuracy of the grid-based maps, OctoMap [9], ours, and the learning-based maps, BGKOctoMap [28] and LARD-HM [19]. For the test, ours represents the octree map using the combined method of super rays and culling region for updates. We use the public KITTI dataset shown in Fig. 2.1-(c) for all points and Fig. 2.13-(a) for test points.

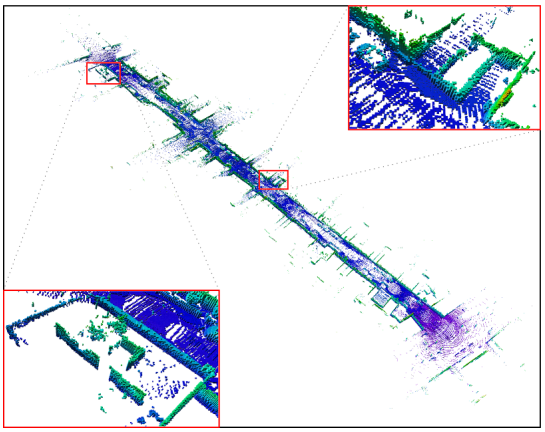
Each update method of map trains its representation with 80% of point clouds in 395 scans and uses the remaining 20% points for testing the accuracy of mapping. We prepare the test points with occupancy states using the remaining points for computing the representation accuracy; free points



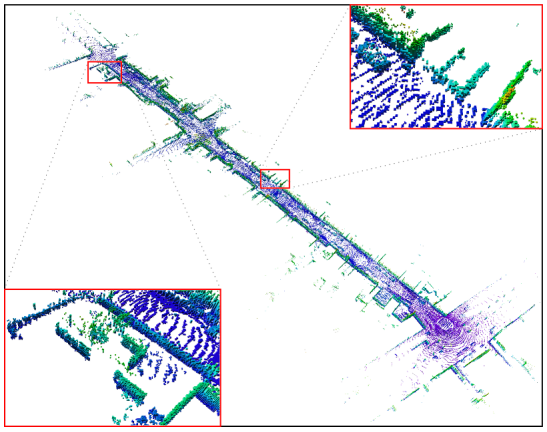
(a) Occupied test points (ground truth)



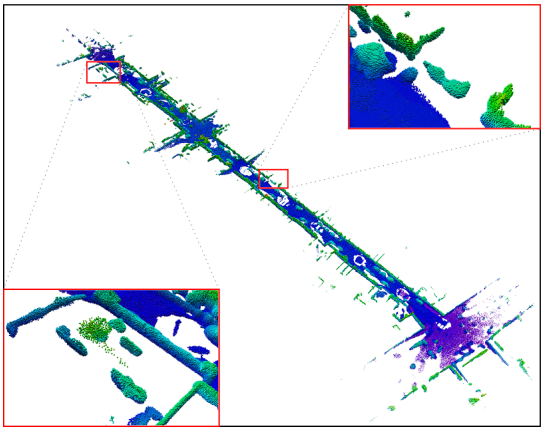
(b) OctoMap (83.6%, 2.19 sec/scan) [9]



(c) Ours (87.1%, 1.10 sec/scan)



(d) BGKOctoMap (81.2%, 2.63 sec/scan) [28]



(e) LARD-HM (86.0%, 3.59 sec/scan) [19]

Figure 2.13: On-the-fly occupancy mapping. These figures visualize the points that each map classifies the test points to be occupied in our navigation scenario. We do not visualize the free points in this figure to avoid cluttered visualization, but consider them to compute the representation accuracy. The color represents the relative height of points, and the number in parenthesis is the representation accuracy and the update speed of a map. Our method shows the fastest update performance resulting in the highest representation accuracy.

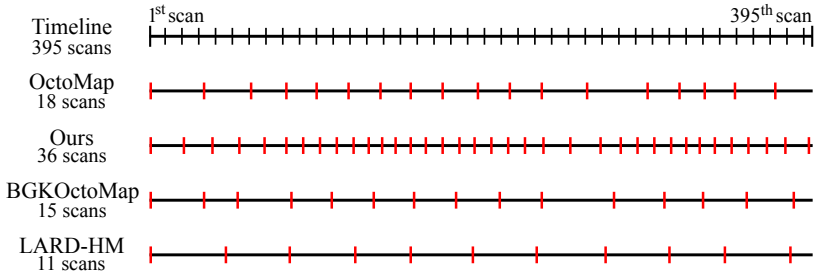


Figure 2.14: Map update timestamps. This figure shows timestamps, represented by red bars when each map uses point clouds in a scan for updates. A faster method can process more scans. The timeline of 395 scans captured by a 3D laser scanner at 10 Hz represents each of 10 scans as a black bar.

in the test set are selected randomly along rays traversing from the sensor origin to endpoints which have occupied states. For computing the representation accuracy, we consider points in the test set with occupancy probabilities less than 0.3 and larger than 0.7 values to have free and occupied states, respectively. In such test setting, we measure the rate of correct prediction of occupancy states, i.e., the rate of prediction showing the same occupancy states to the pre-computed states at all the test points, and report the measurement as the representation accuracy for each map. Note that OctoMap and ours use the 0.2 m resolution to represent the environment, and BGKOctoMap and LARD-HM use all the same values of parameters reported in their corresponding papers.

The KITTI dataset we used in the test consists of raw point clouds captured by a 3D laser scanner at 10 Hz and the recorded configuration of a vehicle that had driven in a residential area. Using a playback tool *bag* on ROS (Robot Operating System), we can simulate a navigation scenario that a vehicle navigates the region and builds a map using captured sensor data in real-time. On the simulation, we let each map to discard point clouds of scans acquired during processing another scan, s, and process an available scan right after finishing the update process of the scan s. The playback of this scenario runs about 40 seconds because the KITTI dataset consists of 395 scans captured at 10 Hz.

In such a navigation problem, the ability of more frequently updating an accurate map can be considered as a better reaction ability to avoid suddenly appeared obstacles. For demonstrating such importance of real-time performance, we measure the time stamps of scans used for updating a map during the simulation (Fig. 2.14), and report the representation accuracy of the updated map using the test points (Fig. 2.13). As shown in Fig. 2.14, ours handles raw sensor data most frequently and uses the most number of scans for updating the map compared to the other mapping algorithms. As a result, the combined method using super rays and culling region deals with the number of scans by a factor of two over the OctoMap.

Such a high update speed results in a high mapping accuracy since our map uses occupancy information of many sensor measurements. Fig. 2.13 shows the visualization of test points that each map classifies to have occupied states. As shown in the figure, our map has the highest representation accuracy, 87.1%, over the other occupancy maps, thanks to the best performance on the update speed. Compared to OctoMap, ours represents the environment in more details as reporting a high accuracy of mapping. BGKOctoMap makes the sharp representation despite relatively a small number of scans. On the other hand, LARD-HM shows the dense representation for occupied points. However, the map has a relatively low accuracy of mapping for free states. In this test, our combined method shows the closest update speed to the scanning speed of sensor compared to other methods. As a result, we achieve the high representation accuracy, while utilizing a high number of point clouds in many scans.

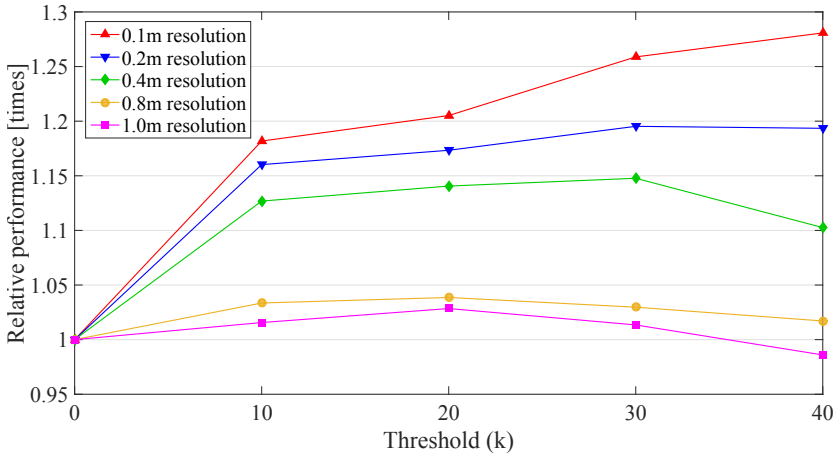


Figure 2.15: The relative performance to the case computed without using the threshold, i.e., $k = 0$. A higher value indicates faster performance. We report the performance of batch-based updates using super rays on OctoMap with various resolutions. We pick $k = 20$ for all the other tests.

Table 2.2: The number of generated super rays with different resolutions.

# of points	Indoor [89,446]		Outdoor [247,817]	
Evaluation	# of super rays	# of points / super ray	# of super rays	# of points / super ray
0.1m	43605	2.1	186504	1.3
0.2m	25064	3.6	150453	1.6
0.4m	10668	8.3	102076	2.4
0.8m	3072	29.1	52906	4.7
1.0m	2073	43.1	40833	6.1

2.4.3 Analysis of super ray based updates

We analyze the performance of our super ray based updates in terms of two factors; the grouping ratio depending on various resolutions and the user-defined threshold limiting to generate super rays.

Super rays improve the update performance by reducing the number of rays used for updating the maps. To analyze such performance improvement, we measure the number of super rays with its grouping ratio in Table 2.2 for the test settings. As can be seen in this table, our method shows varying grouping ratios depending on resolutions. Intuitively a higher grouping ratio of our super rays leads the update methods to reduce the number of traversals more, which results in the high performance improvement.

Overall, our super rays give the high performance improvement to the update methods despite the computation time for generating super rays. They, however, show slightly lower performance gain in the maps with a very high resolution, e.g., 0.1 m, due to a low grouping ratio and its overhead for generating super rays in the high resolution. To lower the overhead, we use only a simple heuristic identifying based on the factor, the number of points in a cell. We use a simple threshold value, k, as the minimum number of points in a cell for generating super rays. In other words, for a cell with points less than k, we simply consider each point in the cell as a super ray with weight 1. For the rest of other cells, we apply our

Table 2.3: The number of unnecessary traversals occurred by batching and culling region based methods.

# of unnecessary traversals	Indoor		Outdoor	
Evaluation	Batch	Culling region	Batch	Culling region
0.1m	2649K	473K	11.0M	9.8M
0.2m	1277K	230K	7.2M	6.1M
0.4m	608K	109K	3.8M	2.9M
0.8m	240K	39K	1.7M	1.2M
1.0m	215K	27K	1.3M	0.8M

method using super rays (Sec. 2.3.3). In practice, we found that 10 to 30 for k work reasonably well, and pick 20 since this setting shows robust performance gain across different tested resolutions over the case of $k = 0$ (Fig 2.15).

2.4.4 Analysis of culling region based updates

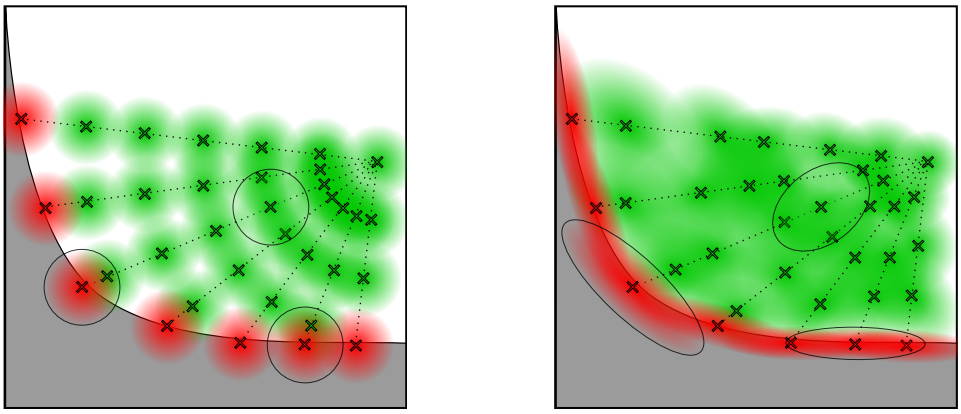
Our culling region improves the update performance by reducing the number of unnecessary traversals that do not affect the occupancy probabilities of maps. To analysis such performance improvement, we measure the number of unnecessary traversals processed in the prior work and our culling-region based method, as shown in Table 2.3. The culling approach reduces a huge amount of such traversals, 83.4% on average, in the indoor scene. As a result, this method enables the performance improvement for updating a map without sacrificing the occupancy information of sensor data. For example in the test shown in Fig. 2.12-(a), the culling region consisting of 57,842 cells removes the 82.1% unnecessary traversals accounting for 60.7% of entire traversals of the batch-based method. As a result, our culling region-based method improves 1.7 times performance over the prior work. For the outdoor scene, our method shows less decrease in the number of unnecessary traversals, 22.9% on average, and achieves the 1.2 times performance improvement on average.

Table 2.4: Overall time (FPS) including time spent on generating super rays and culling region (Proc.) and time spent on updating maps (Update). The number within the parenthesis indicates the number of traversed cells for updates.

Indoor Dataset															
Resolution	0.1m			0.2m			0.4m			0.8m			1.0m		
Evaluation	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]
OctoMap + Batch	2.7	0	365.1 (3586K)	5.4	0	184.1 (1663K)	9.8	0	102.0 (861K)	15.8	0	63.2 (458K)	17.7	0	56.4 (400K)
OctoMap + Ours	**4.9**	74.3	128.2 (1241K)	**18.0**	14.7	40.9 (382K)	**54.5**	5.5	12.9 (122K)	**138.0**	3.4	3.9 (35K)	**164.3**	3.7	2.4 (22K)
GridMap + 3DDDA	2.9	0	343.2 (2972K)	5.8	0	172.6 (1531K)	10.8	0	92.7 (826K)	18.3	0	54.7 (448K)	21.0	0	47.7 (392K)
GridMap + Ours	**6.9**	63.4	82.2 (632K)	**23.0**	10.8	32.7 (252K)	**69.0**	3.3	11.2 (87K)	**184.8**	2.1	3.4 (25K)	**246.9**	2.0	2.1 (15K)
Outdoor Dataset															
Resolution	0.1m			0.2m			0.4m			0.8m			1.0m		
Evaluation	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]	**FPS**	Proc. [ms]	Update [ms]
OctoMap + Batch	0.18	0	5427.7 (55.4M)	0.59	0	1681.8 (17.4M)	1.39	0	719.4 (7.3M)	2.62	0	381.1 (3.6M)	3.18	0	314.6 (2.9M)
OctoMap + Ours	**0.19**	89.1	5289.4 (52.3M)	**0.67**	42.6	1441.3 (14.2M)	**2.06**	21.2	463.6 (4.5M)	**6.11**	14.4	149.3 (1.4M)	**8.61**	13.3	102.8 (1.0M)
GridMap + 3DDDA	0.26	0	3817.4 (25.2M)	0.69	0	1441.3 (12.7M)	1.43	0	698.7 (6.5M)	2.77	0	361.4 (3.4M)	3.35	0	298.8 (2.8M)
GridMap + Ours	**0.27**	37.0	3672.2 (22.2M)	**0.84**	21.5	1175.9 (9.6M)	**2.29**	12.2	423.9 (3.7M)	**6.44**	8.2	147.1 (1.3M)	**9.38**	7.6	99.0 (0.8M)

Chapter 3. AKIMap: Adaptive Kernel Inference for Dense and Sharp Occupancy Grids

3.1 Introduction



(a) Fixed kernel inference (b) Adaptive kernel inference

Figure 3.1: Comparison between two different types of kernel inference models in occupancy mapping. (a) shows the fixed kernel inference using isotropic estimation and (b) represents our adaptive kernel inference on the occupancy samples. Outlines of the circles or ellipses represent each support region of kernel estimation at the sample. We denote the occupied and free states by the red and green colors, respectively, and the object is colored by the gray.

LiDARs and laser scanners can capture a sparse point cloud representing partial occupancy observations of an environment. Regression-based approaches such as [13,18] utilize the correlation of occupancy observations for representing the occupancy state of the environment densely. These maps predict the occupancy probabilities and update their occupancy representations of the environment whenever the sensor captures a new point cloud. However, a well-known drawback is a high computation overhead when applying the approaches to on-the-fly systems. Recently, Doherty et al. [29] proposed a Bayesian non-parametric kernel inference model, which estimates occupancy with a reduced computational cost. The model shares the same isotropic bandwidth across the kernel shown in Fig. 3.1-(a) in order to achieve fast processing speed. Nonetheless, we found that using such fixed kernel can cause a false occupancy representation or unsupported region, especially when the distribution of occupancy observations is sparse and highly nonuniform.

In this chapter, we propose a new adaptive kernel inference model, AKIMap, which reconstructs a sharp, but dense occupancy grid, while handling the non-uniformly distributed sparse occupancy observations. As our main technical contribution, we leverage an anisotropic kernel in robust and efficient kernel inference model for occupancy predictions of the environment (Fig. 3.1-(b)). In our approach, the kernel shape varies locally so that occupancy boundaries can be estimated appropriately. Specifically, our adaptive method optimizes the kernel shapes according to the distribution of occupancy observations. In addition, we present a two-stage optimization performed per kernel locally for the real-time performance of occupancy mapping.

Table 3.1: Summary of notations

$\mathbf{x} \in \mathbb{R}^3$	Observation point
$y \in \{0, 1\}$	Occupancy label; free state: 0, occupied state: 1
$\{\mathbf{x}_i, y_i\}$	i-th occupancy sample
$\mathbf{c}_m \in \mathbb{R}^3$	Center point of m-th cell
$\mathbf{H}_i$	Kernel bandwidth matrix of $\{\mathbf{x}_i, y_i\}$
$\mathbf{\Sigma}_i$	Covariance matrix of $\{\mathbf{x}_i, y_i\}$
s_i	Bandwidth scale; $\mathbf{H}_i = s_i \mathbf{\Sigma}_i$
$k(\cdot)$	Kernel function

We first demonstrate the robust occupancy estimation of our approach using two synthetic datasets for numerical performance comparison. We compare our adaptive approach with the state-of-the-art techniques [28, 29] that employ an isotropic inference. Given the equal amount of sensor data as well as the equal time budget, our approach shows outstanding performance in the occupancy estimation over the tested methods. This result is mainly achieved by our simple, yet effective anisotropic kernel inference. We also test our method in a real environment. For on-the-fly occupancy mapping, our method shows an incremental reconstruction of the dense as well as sharp occupancy representations from noisy point clouds. Furthermore, we evaluate the benefits of the proposed method compared to the orthogonal approaches using anisotropic Gaussian distributions.

3.2 Backgrounds

Our goal is to predict an occupancy probability at a query region from new sparse occupancy observations. To estimate a high-quality occupancy prediction, we adapt the well-established multivariate kernel estimator so that its kernels can be optimized in a data-driven way. The mathematical framework of the kernel estimator is discussed in this section, followed by our novel adaptation for occupancy mapping in Sec. 3.3. Table 3.1 summarizes the main notations used in this chapter.

3.2.1 Multivariate Kernel Estimator

Let $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$ be a d-dimensional input point and its response, respectively, which is an observation pair in a multivariate system, i.e., $d > 1$. Given the observations, the Nadaraya-Watson estimator [34, 35], one of the well-known kernel estimators, predicts an output, y_q, at a query point, $\mathbf{x}_q$, as the conditional expectation:

$$E[y_q|\mathbf{x}_q] = \frac{\sum |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{x}_q, \mathbf{x}_i, \mathbf{H}_i) y_i}{\sum |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{x}_q, \mathbf{x}_i, \mathbf{H}_i)}, \tag{3.1}$$

where $k(\cdot)$ is a kernel function that defines the weight of the response y_i from the nearby input sample $\mathbf{x}_i$ to the query point $\mathbf{x}_q$. The bandwidth matrix $\mathbf{H}_i$, which is a positive definite matrix, controls shape of the kernel function.

In the 3-D occupancy mapping, let $\mathbf{x} \in \mathbb{R}^3$ be a point in 3D space, and $y \in \{0, 1\}$ be an occupancy state, where 0 and 1 represent a free and an occupied state, respectively. We define an occupancy sample, $\{\mathbf{x}_i, y_i\}$, as an i-th observation pair with the occupancy state in our algorithm; we denote such occupancy samples as *occupied sample* or $free$ *sample* depending on their observations of occupancy states. Each sample is associated with a kernel bandwidth, $\mathbf{H}_i$, adaptive to the distribution of occupancy samples, as shown in Fig. 3.1.

3.2.2 Motivation

It is well-known that in the multivariate case, the accuracy of kernel inference highly depends on the bandwidth matrix $\mathbf{H}_i$ [36], as shown in Fig. 3.1. Unfortunately, the crucial parameter, bandwidth matrix, was not fully optimized for the reconstruction of the occupancy map. For example, the recent kernel inference method [29] used an identity matrix with a fixed scale for the bandwidth matrix. This isotropic kernel estimation is simple, but its estimation quality can be degraded when the map is reconstructed from samples on non-uniformly distribution (e.g., representing accurate surfaces of objects). Fig. 3.1-(a) illustrates this scenario. The fixed kernel inference produces dense estimation results nearby the sensor origin, but fails to estimate some occupied parts along the object's surface or does not preserve the boundary between occupied and unoccupied parts. We observe that this limitation is mainly caused by the isotropic kernel estimation, which does not take account for the non-uniform distribution of the occupancy samples adequately.

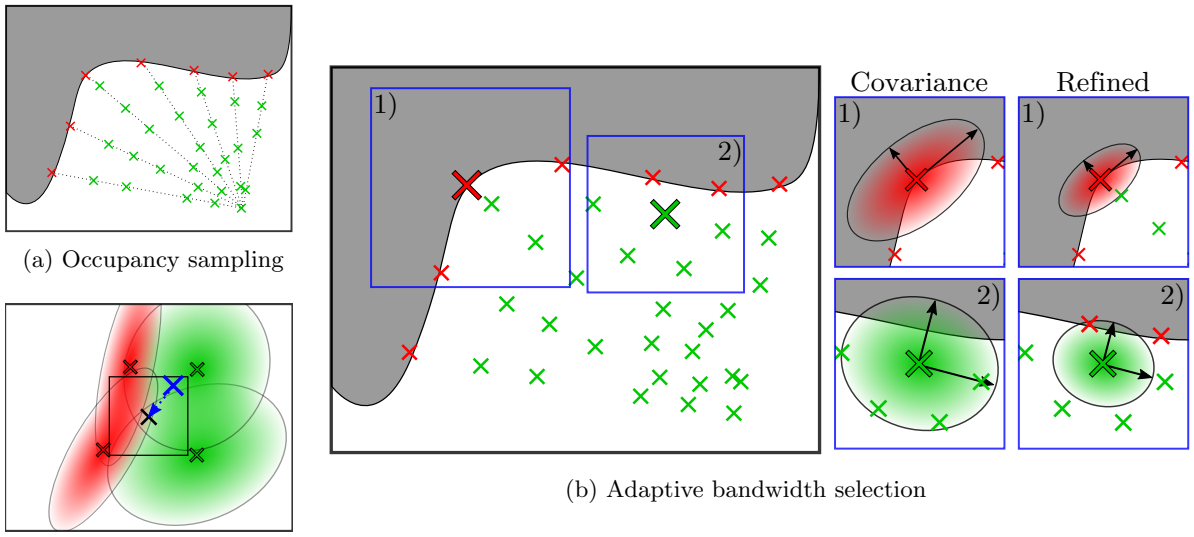
To mitigate this problem, we optimize the bandwidth matrix $\mathbf{H}_i$ at each sample according to the distribution of the samples so that the occupancy state can be estimated robustly given a non-uniformly distributed sample set. As shown in Fig. 3.1-(b), our approach adapts the size and orientation of the anisotropic kernels along the object surface by changing the matrix $\mathbf{H}_i$, and thus preserves the sharp occupancy boundary well, compared to the isotropic approach.

3.3 Adaptive Kernel Inference for Grid-based Occupancy Map

3.3.1 Overview of Our Approach

To reconstruct a high-quality occupancy grid with dense, yet sharp properties from sparse data points with noise, we utilize the multivariate kernel estimation (Sec. 3.2) in a data-driven way for our problem with the occupancy grid.

We first introduce procedures of our mapping framework, AKIMap, as depicted in Fig. 3.2 (Sec. 3.3.2). We describe a simple policy to extract observations about occupancy states from the sparse data as inputs to our inference model. Based on the distribution of the occupancy observations, our method optimizes the bandwidth matrix $\mathbf{H}_i$ so that the occupancy states can be estimated robustly using our anisotropic kernel estimation (Sec. 3.3.3). For efficient runtime updates of the occupancy grid, our adaptive kernel inference estimates and accumulates the occupancy states in cell centers of the occupancy grid incrementally (Sec. 3.3.4). Once a query point is given, we identify a cell containing the point and utilize its estimation information. Note that we depict the 2D examples in the figures for the simple description, but our method operates in the 3D environment.



(a) Occupancy sampling

(b) Adaptive bandwidth selection

(c) Kernel estimation update

Figure 3.2: Framework of Adaptive Kernel Inference Occupancy Grid (AKIMap). (a) represents occupancy samples gathered from the sparse sensor data. In (b), each blue box represents a search region for finding neighbor samples of each kernel center. As an initial kernel bandwidth, our method computes a covariance matrix from the neighbor samples having the same occupancy state to the kernel center. We then refine the bandwidth matrix adaptive to the local distribution of positive and negative neighbor samples. (c) Finally, our model incrementally estimates and accumulates its information at cell centers. Once a query point (blue X mark) is given, the cell containing the point is used for final occupancy prediction.

3.3.2 Kernel Inference for Occupancy Grid

The kernel inference, Eq. 3.1, requires the occupancy samples and their bandwidth matrices to estimate an occupancy state at a query point. However, it is intractable to store all of the data to the map during the on-the-fly mapping. Concerning this issue, a recent work [29] proposed a mapping framework that incrementally updates the kernel estimations of cells in the occupancy grid.

We extend the kernel inference framework with a fixed, isotropic bandwidth into the inference using our adaptive bandwidth matrices. Given the Bernoulli likelihood $p(y_i|\theta_m)$ of occupancy probability θ_m and its conjugate Beta prior $Beta(\alpha_0, \beta_0)$, the posterior at m-th cell of the framework follows $Beta(\alpha_m, \beta_m)$ expressed as:

$$\alpha_m = \alpha_0 + \sum |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i) y_i, \tag{3.2}$$

$$\beta_m = \beta_0 + \sum |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i)(1 - y_i), \tag{3.3}$$

where α_0 and β_0 originate in the Beta prior; setting the both values to 1 makes an uniform prior. Two parameters of this posterior, α_m and β_m, represent the accumulated kernel estimations from occupied and free samples, respectively. The maximum a posterior (MAP) of the occupancy at the cell then becomes:

$$\theta_m^{MAP} = \frac{\alpha_m}{\alpha_m + \beta_m} \approx \frac{\sum |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i) y_i}{\sum |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i)}, \tag{3.4}$$

where $\mathbf{c}_m$ is a center point of the cell. Note that the MAP, Eq. 3.4, approximates the multivariate kernel estimation, Eq. 3.1. Furthermore, the kernel estimations at the cell enable the efficient incremental updates and occupancy queries that we discuss in Sec. 3.3.4.

Kernel Function. For our mapping problem, we define the kernel function as a bounded kernel, as an unbounded one requires updating all cells in the map, which is computationally prohibitive. Specifically, the sparse kernel [37] we choose is as follows:

$$k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i) = \begin{cases} \frac{2+\cos(2\pi r)}{3}(1-r) + \frac{1}{2\pi}\sin(2\pi r) & \text{if } r < 1, \\ 0 & \text{if } r \geq 1, \end{cases} \tag{3.5}$$

where $r = \sqrt{(\mathbf{c}_m - \mathbf{x}_i)^T \mathbf{H}_i^{-1} (\mathbf{c}_m - \mathbf{x}_i)}$.

Occupancy Sampling. Our approach extracts the occupancy samples $\{\mathbf{x}_i, y_i\}_{i=1:N}$ from sparse sensor measurements captured by sensors. A point cloud data is simply observations about the occupied state at the points sampled from the surrounding objects. Therefore, we use the sensor data directly as the occupied samples. On the other hand, we can observe the free space on a sensor ray traversing from the sensor origin to each hit point of the point cloud. Specifically, our method randomly selects one free sample per sampling distance, e.g. 0.5 m, on sensor rays, similar to ones used in the prior approaches [18, 19]. The random free samples on the rays prevent that our adaptive shapes of kernels are over-fitted to the sampling patterns, instead of the distribution of free space, occurring in a uniform sampling.

3.3.3 Adaptive Bandwidth Selection

In this section, we propose an adaptive technique that varies the kernel bandwidth matrix $\mathbf{H}_i$ of each occupancy sample for computing the occupancy map robustly. Our bandwidth selection reflects the following high-level observations to the adaptive bandwidth. 1) If the kernel center and its neighbor sample have the same occupancy state, we could observe another sample sharing the state in the space between them. 2) Otherwise, we could have a low chance of observing a new sample having the state of the kernel center.

A bandwidth matrix in a 3D environment can be composed of six-parameters: three for a rotation matrix and the others for a length scale of each rotated basis. Ideally, one can opt to optimize all the parameters of the kernel bandwidth. However, the optimization with a large number of parameters requires a huge amount of samples. Since a few occupancy samples from sparse sensor data are available in our mapping system, such optimization can be unstable due to the curse of dimensionality.

We therefore propose an efficient, yet robust way to optimize the bandwidth matrix $\mathbf{H}_i$ with a reduced number of parameters. Our method utilizes a covariance matrix $\mathbf{\Sigma}_i$ so that the kernel can be adapted per sample according to its local distribution of nearby samples. Using the covariance matrix as an initial guess, our bandwidth selector then finds the best scalar scale $\hat{s}_i$ to compose the bandwidth $\mathbf{H}_i = \hat{s}_i \mathbf{\Sigma}_i$.

In the first step, our method computes the covariance matrix $\mathbf{\Sigma}_i$ of an occupancy sample $\{\mathbf{x}_i, y_i\}$. We retrieve a set of the neighbor occupancy samples $\{\mathbf{x}_j, y_j\}_{j=1:M}$ of the kernel center $\mathbf{x}_i$ within a search box, i.e., the one colored by blue in Fig. 3.2-(b). Our approach computes the covariance matrix only from the positive neighbors, i.e., the neighbor samples having the same occupancy states with the occupancy sample $\{\mathbf{x}_i, y_i\}$ under the kernel estimation. Such covariance leads to a scaled and biased shape of the anisotropic kernel toward its positive neighbors. For example, the covariance of occupied samples makes the kernel estimation follow the local surface, which results in the sharp occupancy boundary (Fig. 3.2-(b)).

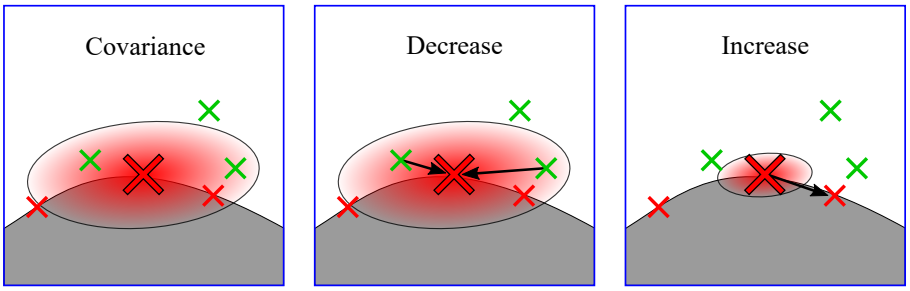


Figure 3.3: Illustration of our adjustment process of kernel shapes with neighbor occupancy samples. (a) represents the initial kernel shape using the covariance matrix. The negative neighbors in (b) decrease the scale, while the positive neighbor in (c) increases it. We find the best scale given the kernel shape of the covariance matrix by minimizing the estimation errors, Eq. 3.7.

While it is very efficient, a kernel estimation only using the covariance can be incomplete. For an example shown in Fig. 3.2-(b), there could be false estimation due to negative samples within the kernel support, once we do not consider them for computing the covariance matrix. Therefore, our approach refines the bandwidth matrix by re-scaling the covariance matrix to reduce such errors by considering such negative samples, while preserving the kernel shape.

At a high-level, our method finds a bandwidth scale s_i to remove the error caused by the false estimation, while keeping the kernel estimation towards the positive samples (Eq. 3.7). To achieve our goal, we design an estimation error of the kernel at a neighbor sample point, and then minimize it by an optimization technique. The false estimation at a negative neighbor, if existing, is caused by a kernel signal even reaching the negative neighbor, as shown in Fig. 3.3-(b). In our bandwidth optimization process, the negative neighbor shrinks the estimation range by pushing the kernel into its center. In contrast to the negative case, suppose a positive neighbor out of the kernel support, which causes no signal at the positive neighbor. Such positive neighbor in our bandwidth refinement prevents the kernel estimation from having too small range, shown in Fig. 3.3-(c).

Specifically, we express a target estimation signal $t(\cdot)$ for computing the estimation error at the neighbor sample $\{\mathbf{x}_j, y_j\}$:

$$t(\cdot) = \begin{cases} k(\mathbf{x}_j, \mathbf{x}_i, \mathbf{\Sigma}_i) & \text{if } y_i \equiv y_j, \\ 0 & \text{if } y_i \not\equiv y_j. \end{cases} \tag{3.6}$$

At the negative neighbor, we set its target signal to be minimum signal value, i.e. 0. On the other hand, we set its target estimation signal at the positive neighbor to be a kernel value using the covariance matrix.

Given our definition of the target estimation signal and the estimation errors, our bandwidth refinement method finds the best bandwidth scale $\hat{s}_i$ based on M neighbor samples within the search region:

$$\hat{s}_i = \underset{s_i}{\operatorname{argmin}} \frac{|s_i \mathbf{\Sigma}_i|^{-\frac{1}{2}}}{2} \sum_{j=1}^{M} \left(t(\cdot) - k(\mathbf{x}_j, \mathbf{x}_i, s_i \mathbf{\Sigma}_i)\right)^2, \tag{3.7}$$

where $|s_i \mathbf{\Sigma}_i|^{-\frac{1}{2}}$ is the signal weight of kernel estimation in Eq. 3.4. Based on the computed scale, we finally select the refined kernel bandwidth $\mathbf{H}_i = \hat{s}_i \mathbf{\Sigma}_i$ for the i-th sample. Note that as an alternative, we tested a separate regularization term with negative samples, but found that the aforementioned approach works well in an efficient manner.

3.3.4 Estimation Update on Occupancy Grid

Our framework aims to support the incremental updates of the occupancy map during on-the-fly mapping. The m-th cell of our map holds two kinds of values, α_m (Eq. 3.2) and β_m (Eq. 3.3). Given the N occupancy samples with their bandwidth matrices at a time step t, these equations can be reformulated to the update rule that accumulates the adaptive kernel estimations during the time steps from 1 to t:

$$\alpha_m^{1:t} = \alpha_m^{1:t-1} + \sum_{i=1}^{N} |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i) y_i, \tag{3.8}$$

$$\beta_m^{1:t} = \beta_m^{1:t-1} + \sum_{i=1}^{N} |\mathbf{H}_i|^{-\frac{1}{2}} k(\mathbf{c}_m, \mathbf{x}_i, \mathbf{H}_i)(1 - y_i). \tag{3.9}$$

In a query step, we can achieve an occupancy probability of a cell based on Eq. 3.4 with these two equations, in a lazy evaluation manner for our adaptive kernel inference.

3.4 Results and Discussions

We have evaluated our approach using two synthetic scenes quantitatively and qualitatively, and have tested it to on-the-fly mapping scenario in the real environment. Specifically, we have compared our approach, AKIMap, with BGKOctoMap [28] and BGKOctoMap-L [29], which are the state-of-the-art techniques that rely on an isotropic kernel inference with a fixed bandwidth.

In all the following experiments, the parameters for BGKOctoMap and BGKOctoMap-L (e.g., the scale and the signal weight of isotropic kernel) were set by the values recommended in the previous papers. Note that the scales and signal weights of our anisotropic kernels are automatically adapted according to the distribution of the occupancy samples.

We set the sampling distance for free observations to 0.5 m used in BGKOctoMap and make one free sample per the sampling distance randomly. For the search range of neighbor samples, we can simply use a fixed search range of 0.5 m, the sampling distance. However, for a higher computational efficiency without degrading the estimation accuracy, we found that we can use a smaller search range near the sensor origin, since as the observed data is close to the sensor origin, there is a higher chance that the data is densely populated. Based on this simple observation, we linearly decrease the search range from the maximum one of 0.5 m at the maximum sensor range to 0.05 m, the voxel size, at the sensor origin.

3.4.1 Performance Comparison

To compare our method with the state-of-the-art techniques, we have used the two scenes, "structured" (Fig. 3.5) and "unstructured" (Fig. 3.6), which were also used in the prior work [28]. Each virtual environment has dimensions of $10.0 \times 7.0 \times 2.0$ m in the Gazebo simulator. Both scenes consist of 12 scans captured at four different locations, and each scan has 3500 points. Fig. 3.5-(a) is the ground truth for the benchmark scene. In this test, we aim to build the occupancy grid at a high resolution, i.e., 5 cm cell size, for observing the dense and sharp characteristics of the tested maps.

To evaluate our method quantitatively, we report the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve and the Mean Squared Error (MSE) of occupancy probability. The ROC curve shows the mapping ability for representing occupancy states given various occupancy thresholds; a higher AUC value represents higher robustness of the inference model. The MSE value of

Table 3.2: Top rows of each dataset show the equal-time comparison w/ varying usages of the data, and bottom rows show performance as we use all the data. Bold numbers represent the best performances in the comparison.

Structured dataset					
Name (Usage)	AUC	MSE	Time [sec]	Accuracy (occupied)	(free)
AKIMap (75%)	**0.912**	**0.063**	2.76	**0.678**	**0.981**
BGKOctoMap-L (100%)	0.892	0.109	2.82	0.476	**0.981**
BGKOctoMap (90%)	0.875	0.120	2.81	0.377	0.979
AKIMap (100%)	**0.937**	**0.060**	3.99	**0.678**	**0.983**
BGKOctoMap-L (100%)	0.892	0.109	**2.82**	0.476	0.981
BGKOctoMap (100%)	0.881	0.119	3.06	0.378	0.979

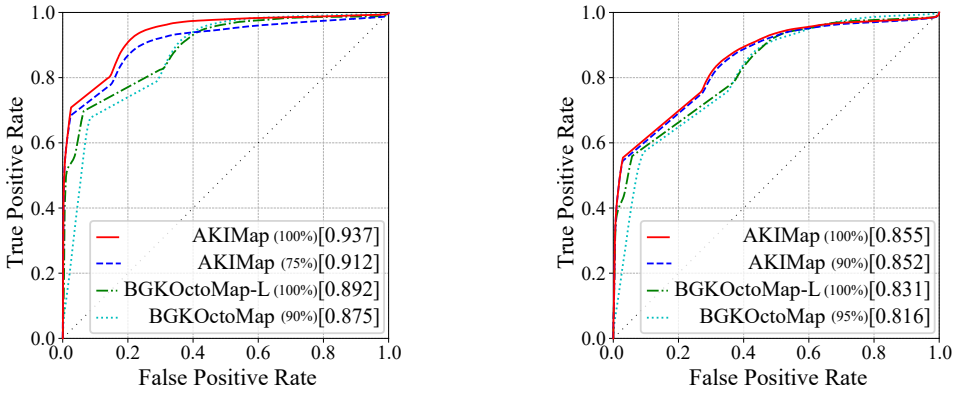
Unstructured dataset					
Name (Usage)	AUC	MSE	Time [sec]	Accuracy (occupied)	(free)
AKIMap (90%)	**0.852**	**0.102**	2.58	**0.664**	**0.965**
BGKOctoMap-L (100%)	0.831	0.133	2.54	0.502	**0.965**
BGKOctoMap (95%)	0.816	0.145	2.52	0.391	0.962
AKIMap (100%)	**0.855**	**0.101**	2.86	**0.665**	**0.966**
BGKOctoMap-L (100%)	0.831	0.133	**2.54**	0.502	0.965
BGKOctoMap (100%)	0.817	0.145	2.61	0.391	0.962

the occupancy map represents the average estimation error of the cells compared to the ground truth map. We report all the results on average in 10 experiments because our work uses a random approach to make free samples. Note that we do not show the standard deviation of performances explicitly, simply because the maximum variation, 0.002, of all the performance metrics is too small.

We first see the performances of different methods using all the sensor measurements in the structured and unstructured environments. As shown in the bottom rows of Table 3.2, the proposed work reports the highest AUC and the lowest MSE compared to the other methods. For example, our method, AKIMap (100%), shows the robustness of occupancy estimation as reporting 0.937 AUC and 0.060 MSE in the structured scene.

These different methods take varying running time, and we thus see how they behave given the same running time budget. For the equal-time comparison, we decrease 5% of sensor data until the processing time of a mapping approach becomes similar to the time of the fastest one. For this reason, we denote the usage of sensor measurements in all the reports by the number in parenthesis next to the name of each mapping algorithm; e.g. (75%) means that its map uses 75% of the data, for reconstructing the occupancy representation.

For the structured dataset, we pick 2.8 s that BGKOctoMap-L can process all the observation data. However, our method and BGKOctoMap can process only 75% and 90% of the data, due to their lower running performance, compared to BGKOctoMap-L. Nonetheless, our method shows the highest AUC result, thanks to its high estimation accuracy (Table 3.2). For the unstructured dataset, we pick 2.5 s as



(a) Structured dataset (b) Unstructured dataset

Figure 3.4: Equal-time comparison. These graphs represent the ROC curves in structured (a) and unstructured (b) scenes of different methods running in the same time budget, except AKIMap (100%), as we vary the occupancy threshold. The number within the bracket is the AUC score of the map.

the running time budget, since BGKOctoMap-L can process all the data given that budget. In this test data, ours achieves the highest estimation accuracy given the same running time. Fig. 3.4 shows ROC curves and their AUC values of the mapping approaches in the structured and unstructured environments in the equal-time comparison.

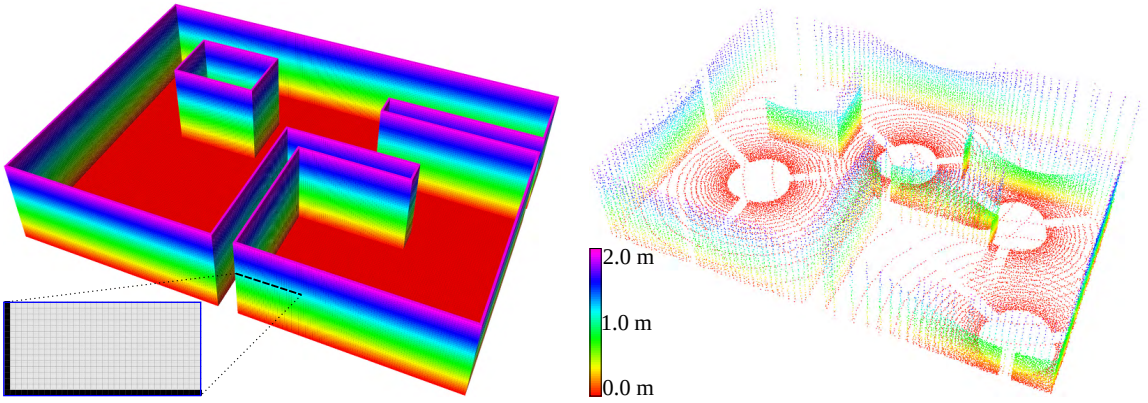
In this test, our approach outperforms the state-of-the-art methods, although it uses less data than the other methods. Our work shows 0.937 and 0.855 AUC scores in the structured and unstructured scenes, respectively. The top rows of Table 3.2 show the reconstruction errors of different occupancy maps in the equal-time comparison. Similar to the AUC scores, our approach produces the lower MSE scores, 0.063 and 0.102 for the two different scenes. These improvements are achieved mainly thanks to our anisotropic model, which allows for estimating the occupancy states robustly.

3.4.2 Qualitative Analysis

We visualize mapping results of the prior experiment using the occupancy threshold of 0.5 at the equal time comparison. Fig. 3.5 and Fig. 3.6 show 3-D visualizations of the various maps for the structured and unstructured scenes, respectively. We only draw the occupied cells classified by the threshold, and the color indicates the elevation of the cell. In addition, the 2D grayscale image located in the left-bottom of each sub-figure shows the occupancy estimations of the cells at a particular region of a specific height, 1.0 m. In this analysis, we report the accuracy scores of occupied and free representations of various methods (Table 3.2).

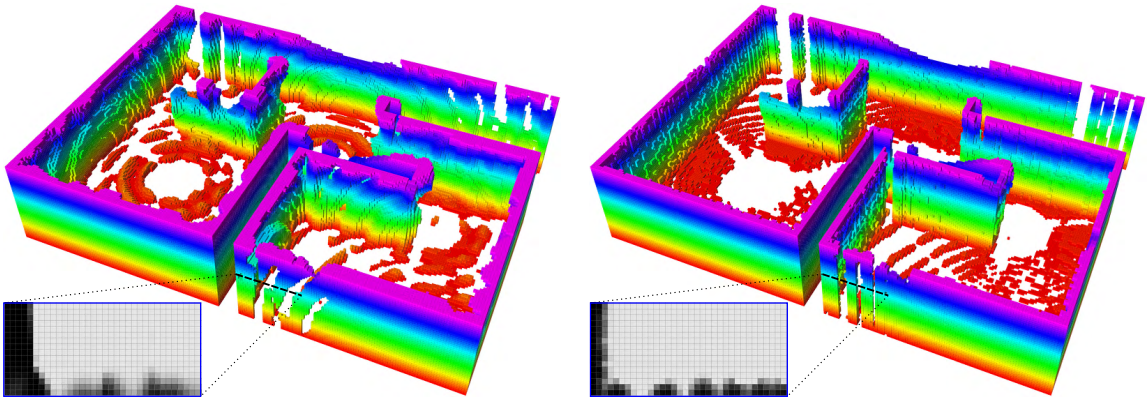
The isotropic inference models (Fig 3.5-(c) and Fig 3.5-(d)) produce dense estimation results given the original, sparse sensor measurements shown in Fig 3.5-(b). These maps, however, do not robustly estimate occupied regions far from the sensor origin, where the data is more sparse than others. On the other hand, our anisotropic model makes such regions (e.g., wall) to have the occupied state thanks to our adaptive bandwidth selection guided by the local distribution of samples, as shown in Fig. 3.5-(e).

Furthermore, our method reconstructs the occupied space more accurately than the previous techniques. As shown in the 2D images of Fig. 3.5-(c) and (d), the isotropic kernel models tend to produce over-estimation results for the occupied space. On the other hand, our anisotropic kernel maintains the sharp representation for the occupied region, shown in Fig. 3.5-(f), by adapting its kernel shapes locally according to the distribution of the sensor data.



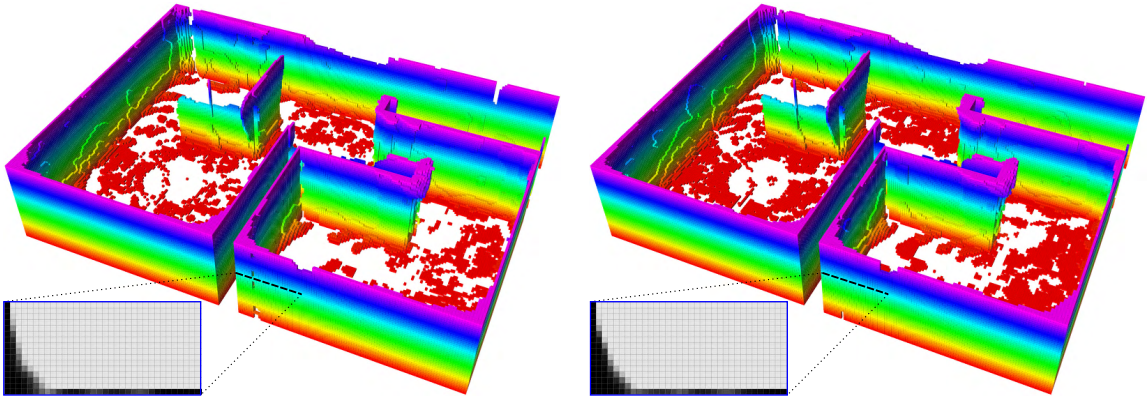
(a) Ground truth

(b) Raw sensor data



(c) BGKOctoMap (90%) - 0.875 AUC

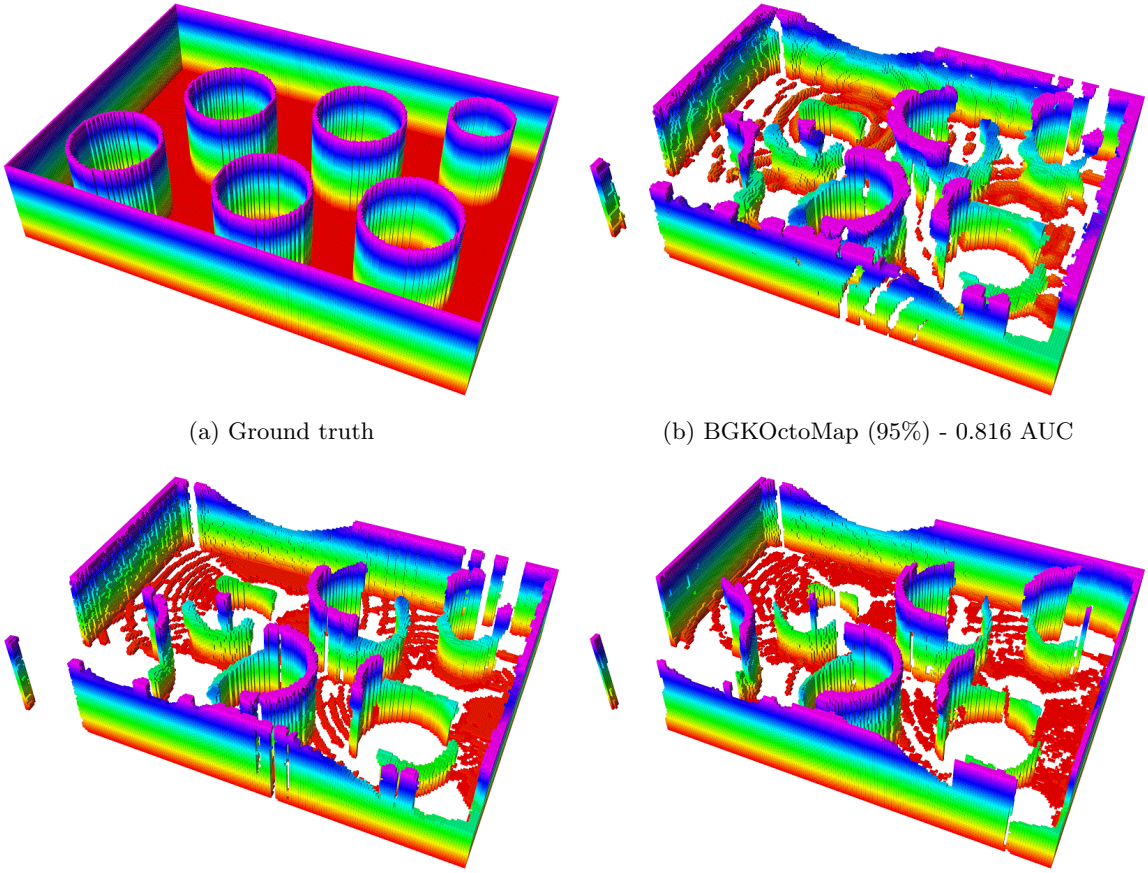
(d) BGKOctoMap-L (100%) - 0.892 AUC



(e) AKIMap (75%) - 0.912 AUC

(f) AKIMap (100%) - 0.937 AUC

Figure 3.5: Occupancy mapping results in the structured scene. We visualize the occupied cells classified based on the occupancy threshold (0.5), where the color represents the elevation from 0.0 m to 2.0 m. The gray-scale 2D image located in the left-bottom represents the occupancy probabilities of cells of a L-shape at the 1.0 m height; colors from white to black represent occupancy probability from the free to occupied states. The number in each parenthesis indicates the amount of used sensor measurements for each algorithm.



(a) Ground truth

(b) BGKOctoMap (95%) - 0.816 AUC

(c) BGKOctoMap-L (100%) - 0.831 AUC

(d) AKIMap (90%) - 0.852 AUC

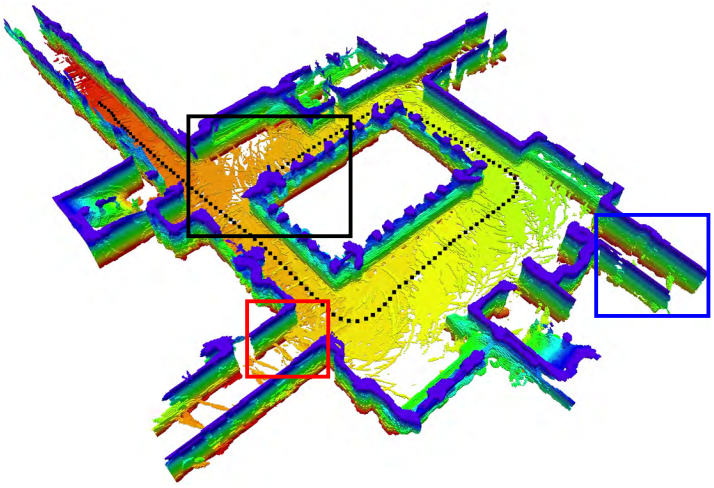
Figure 3.6: Occupancy mapping results using the same time budget in the unstructured scene. These figures show the occupied cells of the different algorithms, where the color indicates a height value.

As a result, shown in Table 3.2, the proposed method shows the highest accuracy of occupied representation, 0.678 (structured) and 0.664 (unstructured), compared to the prior, isotropic estimations. This result shows that our work based on adaptive kernel inference is able to handle the sparsely distributed data where its density varies in a non-uniform manner.

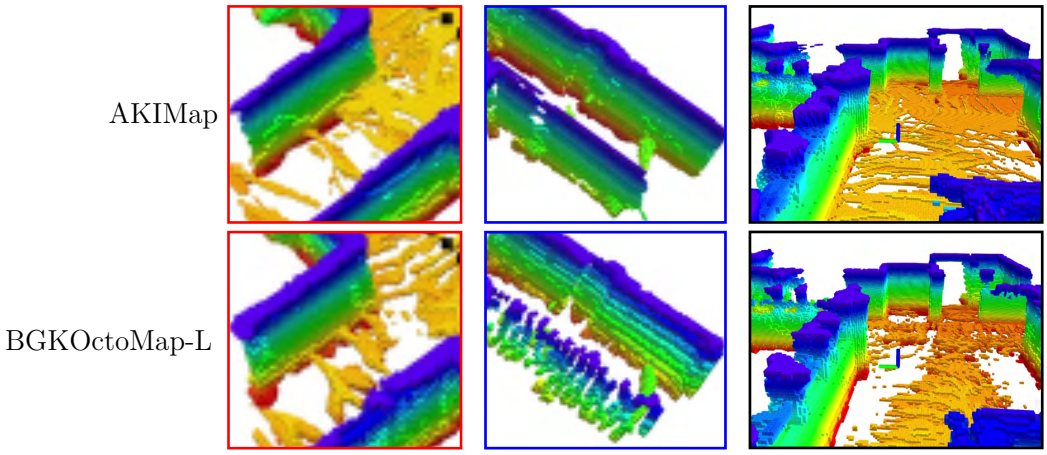
3.4.3 On-the-fly Mapping Scenario

We have tested ours and the prior work, BGKOctoMap-L, in the real environment ($74.4 \times 49.2 \times 2.0$ m). Fig. 3.7 shows the occupied cells of maps with 5 cm voxel size, in which we reconstruct using a mobile robot that equips with LiDAR and IMU sensors. While the robot roams a corridor in 163 seconds, we find its pose in real-time by a localization method [38] and update the occupancy map from sensor measurements on-the-fly. The LiDAR sensor provides a raw point cloud at 10 Hz, but the mapping approaches use a new sensor data right after finishing their map updates. Due to the computational cost, these maps discard the sensor data acquired during the map updates.

In this test, we set the maximum sensing range to 10.0 m and the others to be the same as the settings used in the synthetic datasets, e.g., the maximum search range of our approach is set to follow the sampling distance, 0.5 m. The whole process of on-the-fly mapping can be shown in the presentation of this dissertation defense.



(a) Corridor scene - AKIMap

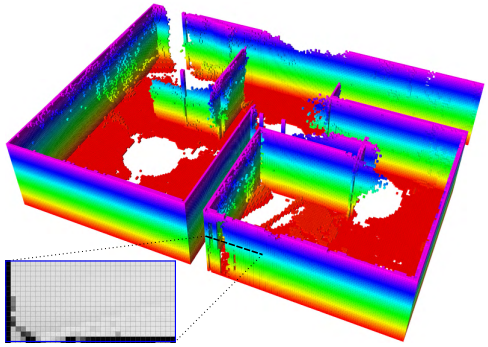


(b) Enlarged visualization

Figure 3.7: On-the-fly mapping result using a mobile robot. The dotted line in (a) indicates a trajectory of the robot, and the red and blue boxes show enlarged regions. The image associated with the black box represents the visualization from the follower's viewpoint. As shown in these boxes, ours represents the more sharp and dense wall and ground surfaces in the corridor than the prior work.

In the experiment, ours and the prior work use 130 and 134 scans for building and updating the occupancy maps, respectively. Both approaches show the dense occupancy representations for the environment, as shown in Fig. 3.7, but our method produces the sharper occupancy states compared to the prior method.

In Fig. 3.7, we enlarge the 3D visualizations at two regions within the red and blue boxes, and represent the occupancy probabilities associated with the regions as the 2D grayscale images. In the region within the red box, ours and the prior work show the dense reconstruction of an area in the corridor. But the proposed work makes the more sharp representation of the wall's surface compared to the prior work. In the case of the blue box, the prior work cannot reconstruct the wall precisely, since it does not consider to adapt its kernel shape according to the density of data. On the other hand, ours enables to reconstruct the sharp and dense occupancy representations given the environment, and this is achieved mainly by our anisotropic kernels that take account for the local distribution of samples.



(a) GMM-OM (100%)

Structured dataset		
	GMM-OM (100%)	AKIMap (100%)
Time [sec]	90.91	**3.99**
- training	32.86	**1.13**
- updating	58.05	**2.86**
AUC	0.939	0.937
MSE	0.065	0.060

(b) Performance comparison

Figure 3.8: Efficiency comparison of bandwidth optimization. This figure (a) shows the visualization of 70-component GMM-OM in the structured scene, and the table (b) shows the computational performances when two methods report the similar AUC and MSE scores. In the test, our approach using local bandwidth optimization processes much faster than the prior work based on global approach.

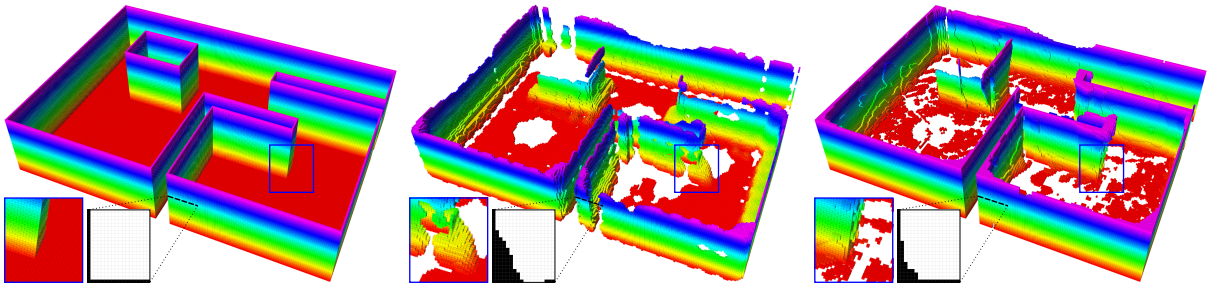
3.4.4 Analysis of Adaptive Bandwidth Selection

Efficiency of local bandwidth optimization. Our adaptive bandwidth selection uses a local optimization strategy for the computational efficiency and due to the curse of dimensionality, as we mentioned in Sec. 3.3.3. Recently, a method using Gaussian mixture model (GMM) [39] was proposed to reconstruct an occupancy map, as an alternative approach of global bandwidth optimization. A covariance matrix of GMM controls the shape of each Gaussian distribution, like our anisotropic kernel bandwidth. This work, however, optimizes the covariance matrices globally with the other parameters such as means and weights of the Gaussian components, where the number of components is much less than the occupancy samples. In this section, we test the computation efficiency of our local adaptation of the bandwidth matrix, comparing with the global optimization approach, GMM-OM.

Fig. 3.8 shows the GMM-OM in the structured scene when it makes 10^6 resamples from 70 component GMM. For comparing the computational efficiency, we choose the number of components at the setting that GMM-OM reports similar AUC and MSE scores with ours, AKIMap (100%) shown in Fig. 3.5-(f). For the number of resamples, we set it to the value recommended in the paper.

In this setting, the map reports 0.939 AUC and 0.065 MSE, while the global optimization for training the GMMs takes 32.86 seconds, shown in Fig. 3.8-(b). On the other hands, for achieving the similar representation performances, our bandwidth optimization takes 1.13 seconds, and finally we reconstruct the map within 4 seconds. These results show that our local adaptive method is more computationally efficient than the GMM-OM based on the global approach, while producing the robust occupancy representation through the anisotropic kernel inference. Furthermore, we observed that the larger number of components of GMM requires much higher computational cost, although it can improve the representation quality. This implies that in our approach, the bandwidth optimization based on global manner can be intractable in on-the-fly mapping, since our model has the much larger number of bandwidth parameters than the GMM.

Effectiveness of bandwidth optimization. Normal distributions transform (NDT) methods [3, 40] have been studied for point cloud registration with memory efficiency. These approaches model the object's surfaces as a set of Gaussian distributions. Furthermore, to support modeling static as well as dynamic objects, the variants of NDT-based methods [41, 42] adopt occupancy information. Each



(a) Ground truth

(b) ONDT - 0.528 F1
0.451 Pre | 0.636 Rec

(c) AKIMap - 0.682 F1
0.678 Pre | 0.686 Rec

Figure 3.9: Comparison of geometry reconstructions. These figures show the reconstruction results for occupied space in the structured scene, where the color represents the relative height of the 3D structure. In each sub-figure, the left blue box shows a highlighted region, and the right black box visualizes the binary occupancy states representing a 2D L-shape at the 1.0 m height. The black and white colors indicate the occupied and non-occupied states, respectively.

Gaussian distribution uses its covariance matrix to represent the object's surface robustly, similar to our adaptive kernels of occupied observations. Therefore, the NDT-based approaches can reconstruct the dense geometry representation of the environment from sparse point clouds. In this experiment, we test the robustness of geometry reconstruction, comparing ours with the NDT-based occupancy map, ONDT.

We use one of the recent NDT-based maps, ONDT [42], with the default parameters provided in its source code. For example, the method uses a 1.0 m voxel size to have one Gaussian distribution per cell and a 0.169 confidence threshold to determine an object's surface. Since this model aims to estimate the observability of objects, not occupancy state, we measure the reconstruction robustness of the environment's geometry; precision (Pre), recall (Rec), and F1 scores.

Fig. 3.9 shows the reconstruction results of ONDT and ours on the 5 cm grid cells. Our method reports the 0.678 precision and 0.686 recall scores in this test, while ONDT shows 0.451 precision and 0.636 recall values. These numerical results show that ours can reconstruct the structure of the environment more sharply and densely compared to the prior method. In addition, the enlarged visualizations in the boxes represent the qualitative reconstructions of object's surface. In the detailed views, we observe that the objects' representation of ours is sharper than the ONDT, reporting the remarkable improvement in precision. This result shows the effectiveness of our bandwidth optimization process mentioned in Sec. 3.3.3. Each initial kernel bandwidth becomes a covariance matrix in the optimization. However, we select the best bandwidth scale to avoid the over-estimation of occupied space by free samples. Thanks to the refinement process, our method achieves the outperforming robustness of geometry reconstruction compared to the prior method.

Chapter 4. Implicit LiDAR Network: Resolution-free LiDAR for Robust Occupancy Map Representation

4.1 Introduction

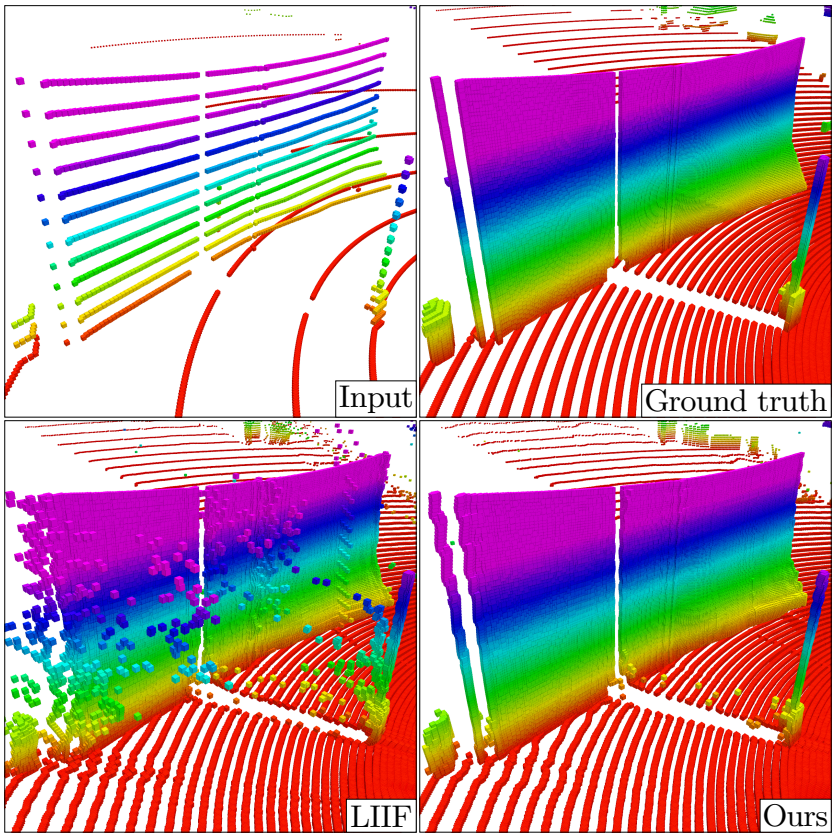


Figure 4.1: LiDAR super-resolution results. This figure shows the reconstruction of dense LiDAR points using the sparse input, where color represents relative elevation of structures.

Various range-based sensors such as LiDAR and laser scanner have different hardware specifications, and thus capture point clouds having various density levels. The density of the sensed point cloud can affect the performance of many robotics/vision tasks such as object detection, recognition, tracking, and motion planning. In the occupancy mapping problem, the robustness of occupancy representation is susceptible to the density of the sensor data, i.e., the resolution of the LiDAR range image. In this chapter, we aim to improve the density of sensed point cloud for robust occupancy representation.

Increasing the sensing resolution takes longer capture time, much more energy consumption, and also higher cost. Hence, super-resolution techniques generating a higher resolution image from a lower resolution one have been actively studied and applied to LiDAR scan data. The advance of deep learning has made significant improvements in super-resolution techniques. The typical approach in previous work [43–46] is to build an autoencoder-style architecture based on convolution/deconvolution layers. Although this approach recovers fine details well, its network is constrained to produce the output image with a specific target resolution used in training and thus restricts its applicability to diverse systems.

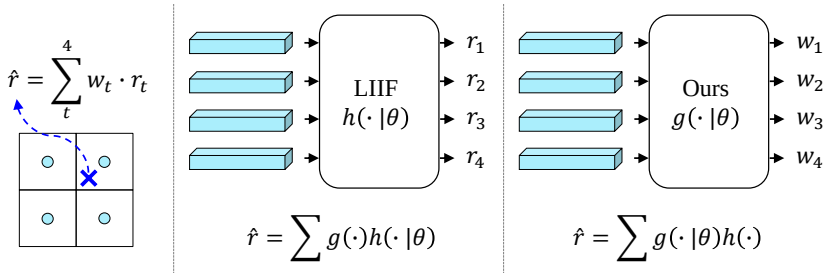


Figure 4.2: Architecture comparison between LIIF and ours. This figure summarizes the difference between two implicit networks - value prediction (LIIF) and weight prediction (Ours), where θ indicates the learning parameters of network. Our method predicts the interpolation weight w_t instead of depth value r_t, resulting in the robust super-resolution shown in Fig. 4.1.

In light of the recent success of learning an implicit function for 3-D shape reconstruction [47–51], Chen et al. [52] first proposed to view an image as *continuous* 2-D data and predict an implicit function that returns color for a given query point. Their method, called Local Implicit Image Function (LIIF), showed how a super-resolution network could be trained without specifying the output resolution, while even achieving better results than the autoencoder-style networks.

Although LIIF can be directly applied to upscale LiDAR, the detailed idea has some problems, particularly when used for the LiDAR range images. LIIF represents the implicit function as a linear interpolation of neighbor pixels, but instead of the pixel values (depths in our case) in the input, *predicted* depth values are used. Thus, it turns out that the network does not *fill* the missing detailed information in the super-resolution but *creates* a new image looking similar to the input, which makes the training very time-consuming. Also, the problem becomes a regression problem in a very high-dimensional space (the dimension of the number of pixels), which also adds more difficulty in network training. Moreover, while the depths for the input image pixels are learned, still the depths are *linearly* interpolated, meaning that *sharp edges* are prone to be blurred, as shown in Fig. 4.1. For LiDAR range images, it is crucial to precisely reconstruct the sharp edges since small errors in the image pixels can result in a significant difference in the 3-D space, largely affecting the performance in occupancy mapping.

To handle these problems, we propose a novel network, Implicit LiDAR Network (ILN), which stems from the idea of learning implicit function for super-resolution. In contrast to LIIF, our ILN does not predict the depths of input image pixels but the *weights* for the interpolation; see the difference in Fig. 4.2. This change makes a big difference in the network training since it does not learn how to make a *new* image, but how to blend the pixel values to *fill* the fine details. Furthermore, such network design makes the training to be converged much faster. In our model, the weights for each query to the neighbor pixels can also be viewed as *attentions*, and thus a recent attention module such as one in Transformer [53] can be leveraged to achieve the best performance. Most importantly, sharp edges can be reconstructed more accurately as shown in Fig. 4.1 since the interpolation is no longer linear.

To this end, we introduce an architecture predicting the weights for interpolation based on the Transformer self-attention module and then conduct experiments by training networks with a novel synthetic large-scale dataset created using CARLA simulator [54] (LiDAR scanning in virtual outdoor scenes). We compare our method with three baselines, bilinear interpolation, LiDAR-SR (the most recent autoencoding-style network) [46], and LIIF [52], and show that our method achieves the best performance. Furthermore, using real LiDAR data, KITTI odometry [55], we discuss the benefits of our method for occupancy mapping in the real environment.

4.2 Problem Definition and Motivation

A range-based sensor shoots multiple lasers and measures the depth (detection distance) of each laser. Let v and h be vertical and horizontal directions of a laser, and r be its measurement depth value. Then, we can represent the measurement points in the sensor coordinate as a set of 2-D depth samples, where each sample indicates the depth r of the laser (v, h). Based on this sensor model, the sample set can be represented as a range image since a real LiDAR has a sensing resolution. In the range image, each pixel indicates the depth r at the pixel center (v, h).

The goal of resolution-free LiDAR is to predict a detection distance $\hat{r}$ of a query laser $\mathbf{q}$ based on an input range image I. Then, our problem becomes finding an unknown function $f(\cdot)$ expressed as

$$\hat{r} = f(I, \mathbf{q}), \tag{4.1}$$

where $\mathbf{q}$ means the query laser's direction (v, h) within the sensor's field of view. The state-of-the-art method, LIIF [52], solves this problem as:

$$\hat{r} = \sum_{t}^{4} g(\cdot)h(\cdot|\theta) = \sum_{t}^{4} \frac{S_t}{S} \cdot h(\mathbf{z}'_t|\theta), \tag{4.2}$$

where $g(\cdot)$ and $h(\cdot)$ denote the weight and value functions, respectively. The network $h(\mathbf{z}_t|\theta)$ predicts the value of a neighbor pixel by using the local feature $\mathbf{z}'_t$ of query and the learning parameters θ, while computing each weight S_t based on the distance between a pixel center and the query point. The problem of this approach is that the network learns new values (depths in our case) for input pixels instead of using *given* values, and thus the output can largely deviate from the input in the early stage of training. Moreover, LiDAR range images typically have lots of sharp edges, while the sharp edges may not be reconstructed well with linear interpolation (Fig. 4.1).

To overcome these problems, this work proposes a novel approach, named Implicit LiDAR Network (ILN), predicting the weights:

$$\hat{r} = \sum_{t}^{4} g(\cdot|\theta)h(\cdot) = \sum_{t}^{4} g(\mathbf{z}'_t|\theta) \cdot r_t. \tag{4.3}$$

Our model utilizes the neighbor pixel value r_t of input image instead of the prediction value. The proposed network $g(\mathbf{z}'_t|\theta)$ predicts the interpolation weight with the deep feature embedding prior knowledge. The predicted weight determines which neighbors are valuable to infer the detection distance $\hat{r}$ of query laser $\mathbf{q}$. This approach focuses on how to fill the unmeasured information with the neighbor pixels (sensor observations), resulting in fast convergence speed and robust LiDAR points reconstruction as well.

4.3 Implicit LiDAR Network: LiDAR Super-Resolution via Interpolation Weight Prediction

In this section, we introduce technical details of the proposed structure to predict interpolation weights. Fig. 4.3 shows a diagram of our network structure trained by a regression loss, L1 loss, in a supervised manner. At the testing step, we extract local features of a query laser from an encoded deep feature map (Sec. 4.3.1). Then, our feature transformer utilizes a self-attention mechanism and fuses the local feature information to improve performance (Sec. 4.3.2). Finally, the proposed network predicts the interpolation weights and computes a detection range of the input query laser (Sec. 4.3.3).

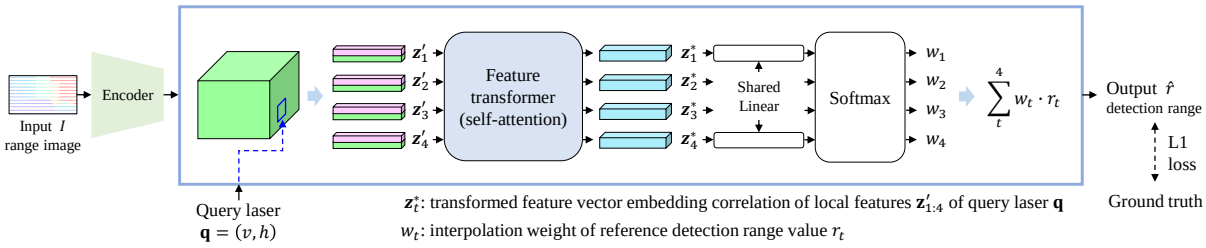
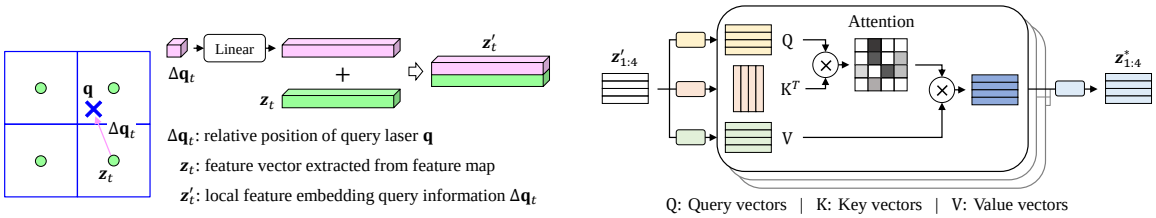


Figure 4.3: Framework of Implicit LiDAR Network (ILN). The proposed model predicts the interpolation weights $w_{1:4}$ with local deep features $\mathbf{z}'_{1:4}$ of the query laser $\mathbf{q}$. Noticeably, the self-attention module enables the accurate detection range prediction $\hat{r}$ of the query laser. See Fig. 4.4 for more details.



(a) Embedding query information to feature vector

(b) Self-attention module in feature transformer

Figure 4.4: Local query embedding and self-attention module. (a) Local deep feature $\mathbf{z}'_t$ of query $\mathbf{q}$ is composed by neighbor's feature $\mathbf{z}_t$ and relative position $\Delta\mathbf{q}_t$. (b) The self-attention module extracts the correlation of local features $\mathbf{z}'_{1:4}$ as an attention map. Then, the feature transformer using the attention data produces the correlated features $\mathbf{z}^*_{1:4}$ for accurate detection range prediction.

4.3.1 Local Feature Extraction

An input range image I consists of the sensor observations, i.e., detection distances of lasers. Nonetheless, individual pixel associated with each observation has insufficient information to predict interpolation weights robustly. We, therefore, extract deep features from the input low-resolution range image. The pixel-based representation of the input enables to utilize the well-studied feature extractors [56–58]. In this work, we opt the feature encoder [58] that the state-of-the-art method [52] uses. The encoder module captures local contexts of pixels through deep convolutional operations, and represents the input range image as a feature map. Each high-dimensional feature vector is assigned to individual pixel of the feature map.

When a query laser is given, we can retrieve the deep features $\mathbf{z}_{1:4}$ located in the query's neighbor pixels of the feature map. Our implicit model utilizes the deep features to predict a detection distance $\hat{r}$ of a query laser $\mathbf{q}$. However, since each feature vector has no query information for the detection range prediction, it needs to embed such information into its neighbor feature vector. As shown in Fig 4.4-(a), our model uses a relative position $\Delta\mathbf{q}_t$ between a pixel center and the query point, similar to the local implicit model [52]. On the other hand, unlike the prior work, our model adopts the positional embedding on feature space in the light of their great success [50, 53, 59]. This process makes a local feature vector $\mathbf{z}'_t$ by fusing the expanded position $\Delta\mathbf{q}_t$ and the feature vector $\mathbf{z}_t$.

4.3.2 Feature Transformation using Self-Attention

Each local deep feature $\mathbf{z}'_t$ can be used directly to predict its interpolation weight w_t of pixel value r_t. However, we found that four predicted weights $w_{1:4}$ determine which reference value should be focused on for the robust final prediction. Based on this observation, we consider that the interpolation process shares a goal of *attention* from a query to its neighbor pixels, and thus leverage an attention mechanism to achieve performance improvement.

In the previous stage, we make four local features $\mathbf{z}'_{1:4}$ embedding query and its neighbor pixels' information. Therefore, by applying a self-attention module to the local features, our model (Fig. 4.3) can predict the interpolation weights $w_{1:4}$ from the query to its neighbor pixels robustly. We found that the self-attention have achieved outstanding performance in Transformer models [53,59] on natural language processing and vision tasks as well. In the light of the achievement, our method uses the self-attention mechanism of the recent model [53].

In a high-level idea, the self-attention of feature transformer fuses the information of local features $\mathbf{z}'_{1:4}$ so that the predicted weights $w_{1:4}$ determine the reference values $r_{1:4}$ reasonably. Fig. 4.4-(b) shows a self-attention process that represents correlation among the local features as an attention map. The Q and K vector sets, originated from the local features, extract the self-correlation that can lead to a good choice for interpolation. Then, this module combines the extracted attention map and the transformed V vectors, resulting in the correlated features $\mathbf{z}^*_{1:4}$.

In training step, the transformer learns its parameters so as to catch the best correlation of input features and thus predict the detection range accurately. As a result, this proposed approach shows the significant performance improvement in our experiment (Sec. 4.4.4).

4.3.3 Interpolation Weight Prediction

As shown in Fig. 4.3, the shared linear layer projects the output features $\mathbf{z}^*_{1:4}$ of the transformer into weight scores, and then the softmax function computes the interpolation weights $w_{1:4}$. At the final stage, we apply Eq. 4.3 to infer the detection distance $\hat{r}$ of query laser $\mathbf{q}$ by combining the reference values $r_{1:4}$ and the predicted weights $w_{1:4}$. Our method utilizes the interpolation values $r_{1:4}$ from the input range image I, while the recent approach [52] predicts the values via a deep network (Eq. 4.2). In the LiDAR super-resolution, we observed that the frequent sharp transitions of range values could make unstable predictions of missing information, and thus result in lots of undesired artifacts (Fig. 4.7). Under this observation, our network focuses on learning the adaptive weights prediction through deep prior knowledge, instead of the values prediction.

4.4 Experimental Results

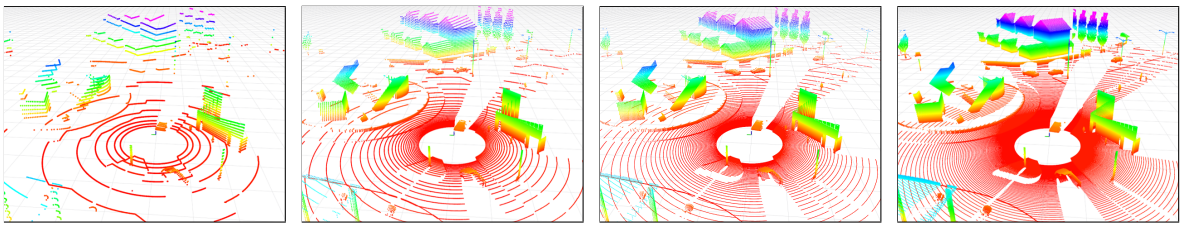
4.4.1 Dataset for resolution-free LiDAR

An implicit network predicts an output signal at any continuous query point. Hence, it could be the best solution for training the network with detection distance samples from infinite resolution LiDAR. Unfortunately, there is no sensor having such hardware specification in the real world, and thus it is challenging to prepare the dataset. In this chapter, we use the CARLA simulator [54] to overcome this problem. The simulator supports ray-based sensing simulation in various realistic environments such as large-scale urban scene.

(a) Simulation scene

LiDAR specification	vertical angle [deg.]	-15 ∼ 15
	horizontal angle [deg.]	-180 ∼ 180
	max. range [m]	80
Scenes	train set (# of scenes)	Town 01 ∼ 06 (22,244)
	test set (# of scenes)	Town 07 & 10 (2,847)

(b) Dataset configurations: LiDAR specification and scene split.



(c) 16 × 1024 (d) 64 × 1024 (e) 128 × 2048 (f) 256 × 4096

Figure 4.5: CARLA dataset. We simulate LiDAR sensors at about 24 K poses with various resolutions reported in (b). In the example scene (a), $(c) - (f)$ represent the point clouds captured by LiDARs having different resolutions; their labels indicate the vertical and horizontal resolutions, respectively.

We can measure ground truth detection distances at various resolution settings in the simulation environments. Furthermore, ideally, it can be possible to train an implicit network while obtaining range samples at any continuous laser direction. However, such online sampling and learning on simulation need intractable training time as well as computational resources. This work avoids this practical issue by simulating extremely high-resolution LiDAR and collecting tremendous detection range samples. We gather the LiDAR data at the maximum 256 and 4096 for vertical and horizontal resolutions, respectively, in which our computation resources are available.

As shown in Fig. 4.5, we prepare the data with four different resolutions. Such multi-resolution settings enable to train the implicit as well as pixel-based super-resolution approaches. In addition, we can measure the super-resolution performances of implicit networks in the various test resolutions.

4.4.2 Experimental Settings

We select the bilinear interpolation algorithm as a baseline approach, which computes the weights of four neighbors. Also, the state-of-the-art implicit method, LIIF [52], is evaluated to check the effectiveness of our weight prediction approach. These methods, including ours, aim to solve Eq. 4.1, LiDAR super-resolution without resolution constraint. On the other hand, LiDAR-SR [46] up-scales the low-resolution range image to its trained resolution only. Using the pixel-based LiDAR super-resolution method, we check the benefits of the implicit model.

The experiments perform LiDAR super-resolution, which upscales the range image from the low 16×1024 resolution to higher resolutions. In the test, we use three test resolutions; 64×1024, 128×2048, and 256×4096. We train a single model of each implicit network with the 128×2048 resolution data and then evaluate it in the various test resolution settings. To reconstruct range images at a specific test resolution, we make a set of query lasers matching pixels' center. Note that it needs to train a pixel-based network at a fixed upscale factor to compare performance. Therefore, we train and evaluate individual LiDAR-SR networks at each test resolution setting in this experiment.

Table 4.1: Quantitative comparison for LiDAR data generation on CARLA dataset. The bold texts represent the best performance on each metric. *Pixel-based super-resolution networks were trained to generate each target resolution individually.

Method	MAE	IoU	Precision	Recall	F1
Test resolution: 64×1024					
LiDAR-SR [46]*	1.560	0.233	0.370	0.377	0.373
Bilinear	2.372	0.202	0.322	0.328	0.325
LIIF [52]	1.558	0.258	0.403	0.409	0.406
Ours	**1.536**	**0.329**	**0.483**	**0.486**	**0.484**
Test resolution: 128×2048					
LiDAR-SR [46]*	1.746	0.161	0.262	0.288	0.274
Bilinear	2.591	0.165	0.268	0.287	0.277
LIIF [52]	1.714	0.236	0.372	0.388	0.379
Ours	**1.690**	**0.331**	**0.483**	**0.498**	**0.491**
Test resolution: 256×4096					
LiDAR-SR [46]*	**1.735**	0.127	0.207	0.245	0.224
Bilinear	2.646	0.163	0.256	0.303	0.277
LIIF [52]	1.923	0.158	0.221	0.356	0.272
Ours	1.763	**0.232**	**0.353**	**0.396**	**0.373**

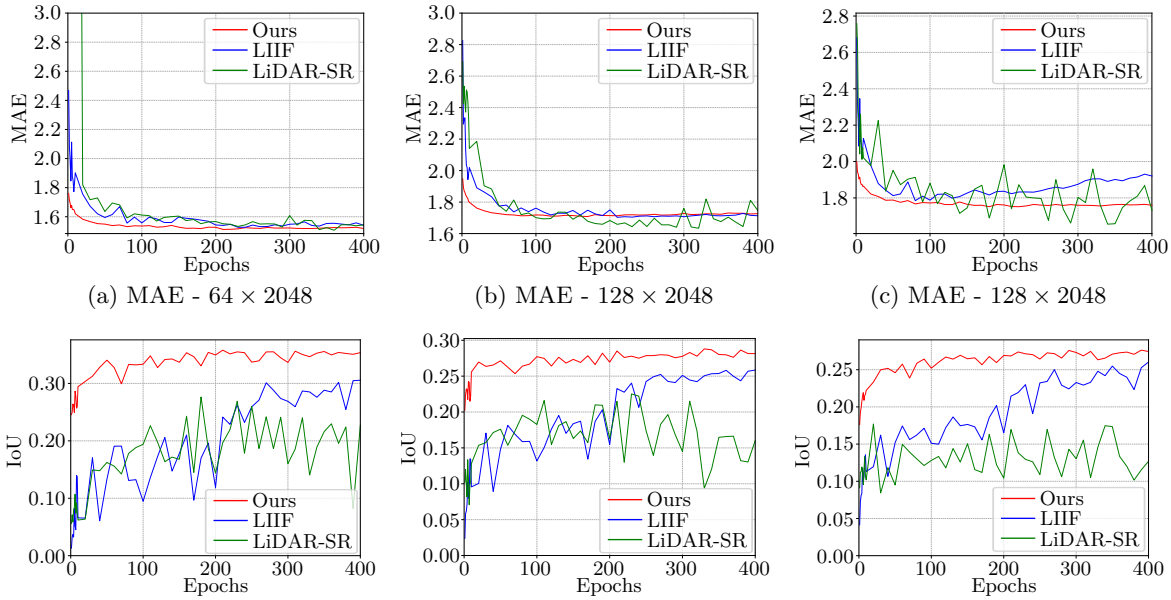
We use two Tesla V100 32GB GPUs except training the LiDAR-SR network for 256×4096 resolution; it requires four GPUs. On the PyTorch framework, we train the prior methods with the parameters reported in their papers. Specifically, we train these models by Adam optimizer [60] with an initial learning rate 10^{-4}. The batch size is set to 16.

The LiDAR super-resolution networks reconstruct the up-scaled range image having test resolution. We measure the mean absolute error (MAE) of all the pixels in the predicted 2D range images. Furthermore, we measure the performances using the 3D points reconstructed by networks. Since various applications use the LiDAR point cloud as raw sensor data, the reconstruction performances represent methods' usefulness. Specifically, we measure the representation accuracy of the points with 0.1 m grid; intersection over union (IoU), precision and recall, and F1 score. These metrics show how well a method reconstructs LiDAR points similar to ground truth points.

4.4.3 Comparison with Prior Methods

In this section, we demonstrate the benefits of our method comparing with the prior methods. In summary, Table 4.1 reports the quantitative performances and Fig. 4.7 shows the qualitative results at the 128×2048 test resolution setting.

Comparison with weight computation approach. When comparing the approaches estimating the interpolation weights, our method outperforms the bilinear interpolation on all the metrics. The bilinear approach and ours utilize the same reference values from four neighbor pixels. Nonetheless,



(a) MAE - 64 × 2048 (b) MAE - 128 × 2048 (c) MAE - 128 × 2048

(d) IoU - 64 × 1024 (e) IoU - 128 × 2048 (f) IoU - 256 × 4096

Figure 4.6: Performances on the test set according to training epochs. We report the evaluation results at every 10 epochs. Comparing with the other methods, ours shows the outstanding convergence speed with stable performance.

the reported performances demonstrate that our deep network predicts the interpolation weights more robustly than the bilinear weight computation. The deep network exploits the prior knowledge and estimates the adaptive interpolation weights based on deep features. As a result, ours reports up to 2.0 times improvement on the IoU evaluation metric.

Comparison with implicit network. Our implicit network predicts the interpolation weights to compute detection distances of query lasers, while LIIF predicts the values (Fig. 4.2). Two implicit methods are trained with the resolution data only; thus, the experiments using the various test resolutions can have different data distributions from the training dataset. The test resolutions less than equal to the training resolution, 64×1024 and 128×2048, indicate the *in-distribution* test environments. Otherwise, the 256×4096 resolution becomes *out-of-distribution*.

Table 4.1 reports the experimental results using both in-distribution and out-of-distribution tests. Overall, our method outperforms the state-of-the-art implicit network in the various settings. On the in-distribution test, ours achieves higher performance than the prior work in both 2D range image and 3D points reconstruction. In particular, our method achieves remarkable performance gains for representation accuracy of reconstructed LiDAR points, as shown in Fig. 4.7. For example, our method shows 0.330 IoU performances on average of two test resolutions, while LIIF reports 0.247 IoU. Furthermore, we achieved significant performance improvements on the out-of-distribution test. Our method shows outperforming 3D points reconstruction, while reporting the meaningful improvement on MAE metric. This result represents that our implicit model can cover continuous queries at an even higher resolution.

We can achieve such improvements thanks to the interpolation weights prediction instead of the values. Like the qualitative results in Fig. 4.7, we observed that the value predictions of LIIF can lead undesired noisy artifacts on 3D representation. On the other hand, our weight prediction approach reconstructs the dense LiDAR points robustly. In our model, the predicted weights $w_{1:4}$ determine the valuable reference values $r_{1:4}$ of the input range image via local deep features $\mathbf{z}'_{1:4}$ and self-attention

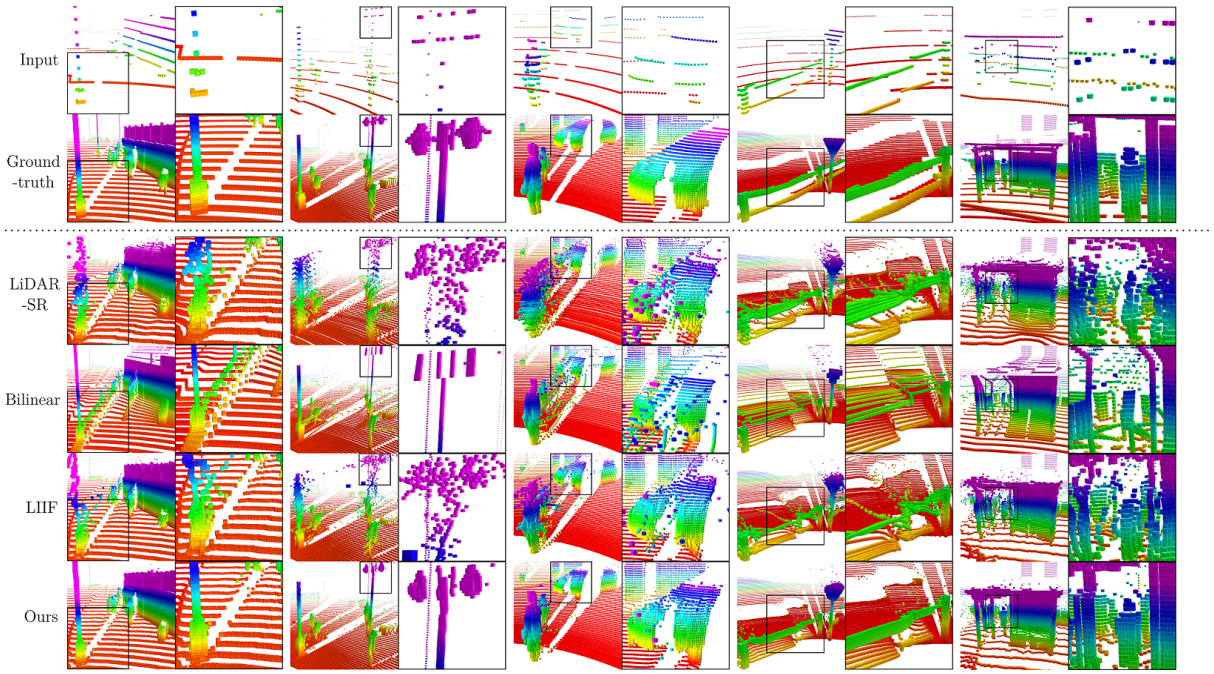


Figure 4.7: The qualitative results of LiDAR super-resolution via various methods. The highlighted region in the black box of each left figure is shown in its right side. Compared to the other methods, our method reconstructs the 3D points robustly with much less noisy artifacts. The color in the captures represents a relative height.

mechanism. The proposed method learns how to blend the input pixel values to fill the unmeasured information through non-linear weights. As a result, the approach shows the outstanding performances for LiDAR super-resolution through quantitative and qualitative results as well.

Comparison with pixel-based super-resolution. Ours and LIIF are based on implicit function; on the other hand, LiDAR-SR has pixel-based convolution/deconvolutional architecture. In this analysis, we check the benefits of our method based on implicit function.

Table 4.1 shows the our method reports the much higher performances on evaluation metrics, except slight lower performance in MAE at the 256×4096 case. Note that since we train our implicit model with the lower resolution, 128×2048, the result of ours shows performances in an untrained out-of-distribution environment. On the other hand, the LiDAR-SR network was trained for the test resolution. Despite such conditions, our model shows outstanding 3D points reconstruction with a similar MAE.

Our model shows such performance gains with a single trained network. The experiments using various test resolutions show that our implicit network can predict the detection distance $\hat{r}$ of query laser $\mathbf{q}$ given the sparse sensor observations, without resolution constraint. Furthermore, our network shows the robust LiDAR points reconstruction in the various test resolutions, compared to the pixel-based networks trained with each test resolution data.

Convergence speed. Our method predicts the detection distance $\hat{r}$ based on predicted interpolation weights with reference values of the input range image. This architecture design results in a significant convergence speed, as shown in Fig. 4.6. These graphs show that our method converges much faster than other methods in various metrics and test resolutions, reporting more stable performances. Such fast convergence speed is helpful for training a new model in a different environment without huge costs.

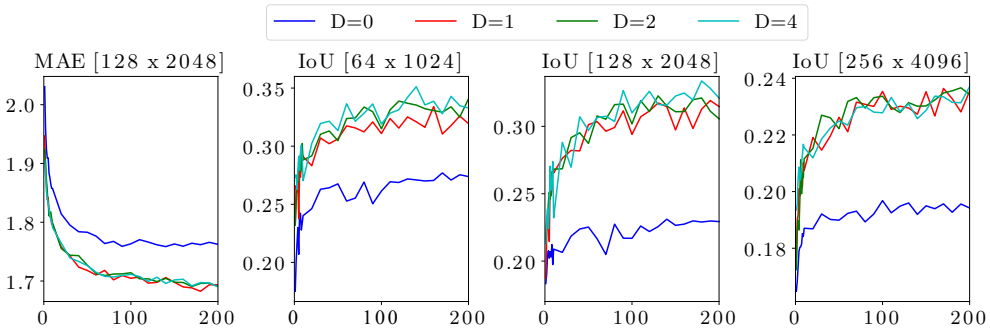


Figure 4.8: Performances of ours depending on the number of attentions, D.

4.4.4 Effectiveness of Self-Attention

Our network utilizes an attention mechanism to achieve performance improvement, as mentioned in Sec. 4.3.2. To show such benefits of attention in our model, we evaluate the performance gains over different numbers of self-attentions. The graphs in Fig. 4.8 show the experimental results at various test resolutions. The result shows slight improvement when applying more self-attentions to feature vectors. On the other hand, we find remarkable performance gains over all the tests when comparing the model with and without the attention module, $D = 1$ and $D = 0$, respectively. This result demonstrates the usefulness of the attentions in our model.

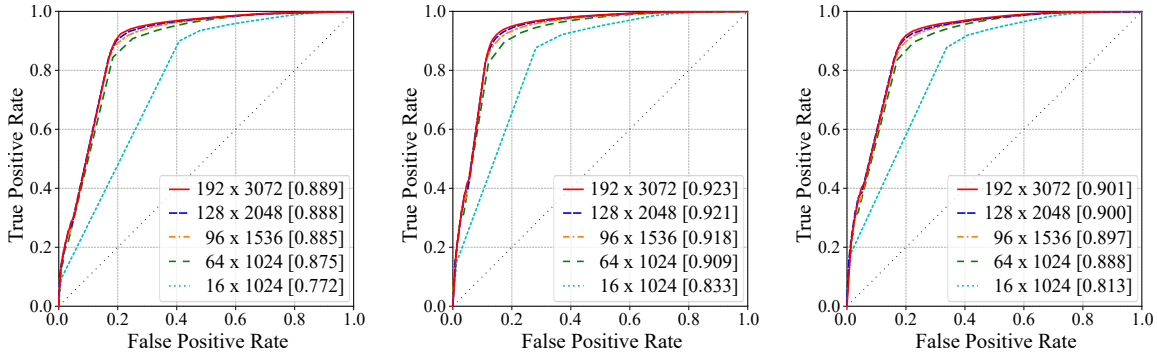
4.4.5 Occupancy Mapping with Real LiDAR Data

In this experiment, we analyze the performance of our network in occupancy mapping with real sensor data. Using the KITTI odometry dataset [55], we evaluate the methods trained with the synthetic Carla dataset without any fine-tuning. The KITTI dataset consists of points that a 64-vertical channel LiDAR has captured. However, we need a 16×1024 range image as an input of networks to measure the performance without new training. To deal with this issue, we down-sample the high-resolution points of each scan and make its low-resolution range image.

Our implicit network reconstructs dense LiDAR points from the 16-channel to various test resolutions; 64×1024, 96×1536, 128×2048, and 192×3072 resolutions are used in this test. Our mapping framework updates an occupancy grid having the 0.2 m voxel size from the 16×1024 sensed points as well as the reconstructed points.

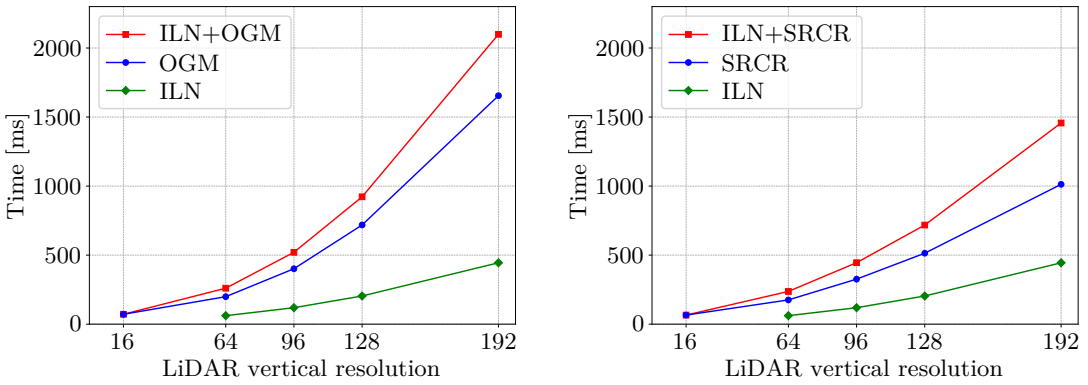
The AUC graphs of Fig. 4.9 show the representation accuracy of the occupancy maps using various resolutions in three different scenes, and Fig. 4.11-(a) represents the occupied regions of occupancy maps in the scene, KITTI 06. The graphs of all three scenes report that occupancy representation becomes more accurate as the proposed network reconstructs the more dense LiDAR points. Note that the LiDAR configuration of the KITTI dataset is different from the training dataset; a LiDAR of the KITTI dataset has the $-25° \sim 3°$ vertical sensing angle. Due to the difference, the real LiDAR dataset can have unseen data distribution when training. Nonetheless, the map using our LiDAR super-resolution network achieves a more accurate occupancy representation than it using only sensed data. These results demonstrate that our network enables the maps to represent the occupancy states of the environments robustly.

Noticeably, we found a significant performance improvement between test settings using the 16-channel input points and the 64-channel reconstruction outputs. For example, Fig. 4.11-(b) shows the sparse representation caused by the limited number of sensed points, reporting a 0.833 AUC score. On



(a) KITTI scene 04 (b) KITTI scene 06 (c) KITTI scene 07

Figure 4.9: Representation accuracy of occupancy mapping. The 16×1024 indicates a resolution of input sparse sensor data, and the others represent the resolutions for point cloud reconstruction via our LiDAR super-resolution network. According to various test resolutions, these graphs show the ROC curves in three different scenes of the KITTI dataset. The number within the bracket is the AUC score of the map.

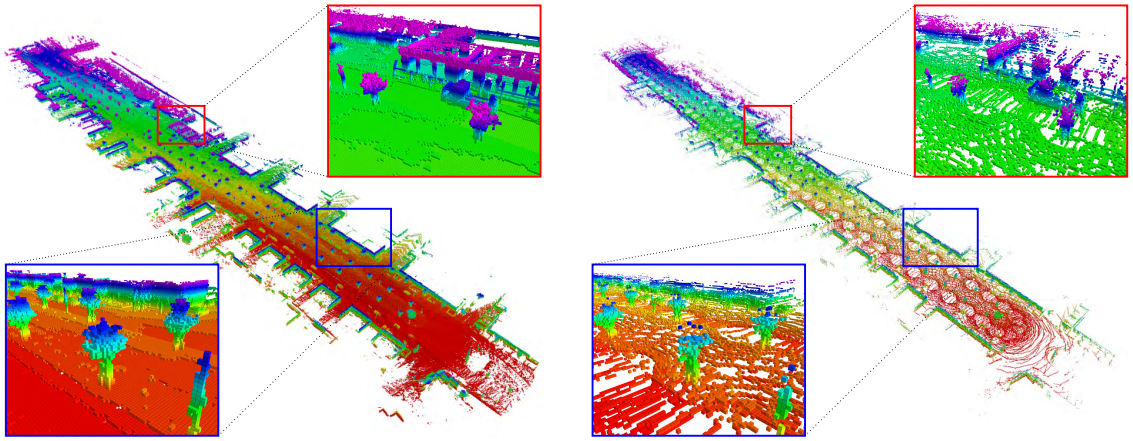


(a) Using traditional update method (OGM) (b) Using super rays and culling region (SRCR)

Figure 4.10: Update performance of occupancy grid in KITTI scene 06. Each graph shows the update time according to various density levels of the reconstructed points. The blue and green lines represent the map update time and LiDAR points reconstruction time, respectively.

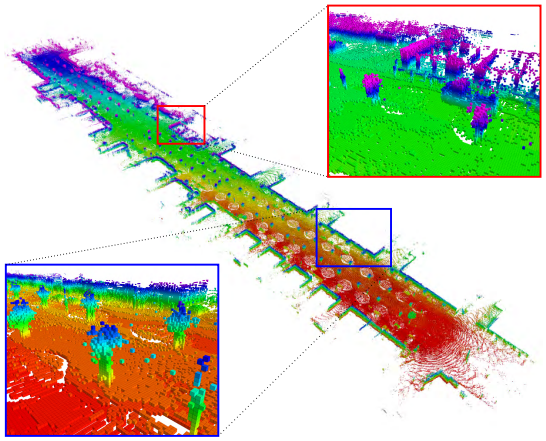
the other hand, as shown in Fig. 4.11-(c), applying the reconstructed points to occupancy mapping results in the dense and robust representation with a high AUC, 0.909. Moreover, we achieve up to 0.923 AUC score at the maximum test resolution in this experiment.

The large number of points require high computational overhead for updating an occupancy grid despite the dense occupancy representations. Fig. 4.10-(a) shows that the map update times (blue line) become the higher as the LiDAR resolutions are larger. Furthermore, the map update dominates the total processing time (red line) compared with the LiDAR point reconstruction (green line). To reduce the processing time, we apply the super rays and culling region-based update methods (Chap. 2). Fig. 4.10-(b) reports the performance improvement in update speed compared with the results using traditional updates (Fig. 4.10-(a)). The super rays and culling region accelerate the update speed of occupancy grid although using the dense points reconstructed by our implicit LiDAR network.

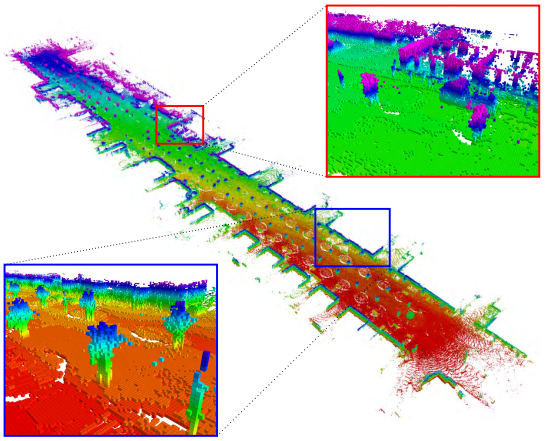


(a) Occupied test points (ground truth)

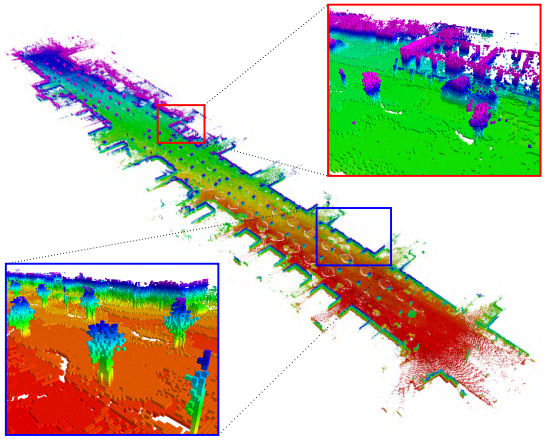
(b) Input 16×1024 (0.833 AUC)



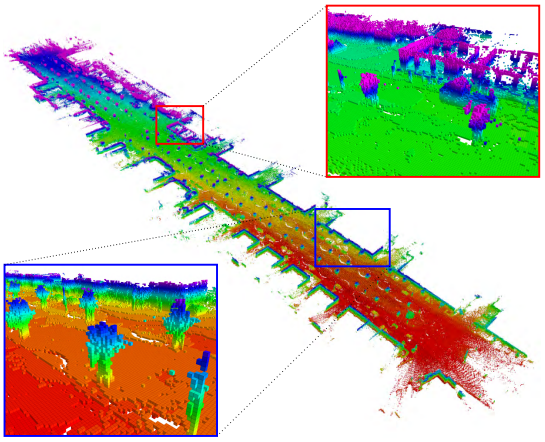
(c) ILN 64×1024 (0.909 AUC)



(d) ILN 96×1536 (0.918 AUC)



(e) ILN 128×2048 (0.921 AUC)



(f) ILN 192×3072 (0.923 AUC)

Figure 4.11: occupancy mapping results in KITTI 06 scene. These figures show the occupied cells of the occupancy maps updated from point clouds having various resolutions. The red and blue boxes of each figure represent the enlarged regions of the environment.

Chapter 5. Conclusions

In this dissertation, we propose three methods that exploit the spatial correlation of point clouds for dense occupancy mapping with real-time updates. We design 1) a real-time update algorithm based on geometric update patterns of grid-based occupancy maps. Furthermore, to achieve dense occupancy map representation, we employ 2) a regression approach using non-uniform distributions of occupancy observations and 3) a deep network learning the spatial correlation of measuring point clouds.

In Chapter 2, we have proposed two novel update methods for grid-based occupancy maps based on super rays and culling region. As our main algorithm, we construct a super ray that updates the same set of cells on occupancy maps in a single traversal. Specifically, we have proposed to use a mapping line for efficiently generating super rays in 2D case, and extend it to handle the 3D case. We also have proposed a culling region for reducing the number of unnecessary map updates. We have tested our methods using public datasets to reporting the update speed on grid-based maps, and achieve overall outstanding performance across all the tested configurations.

In Chapter 3, we have proposed a new method, AKIMap, to use an adaptive kernel inference for the dense and sharp representation on the occupancy grid. For high estimation accuracy, our work has proposed to find the adaptive kernel bandwidth based on the local distribution of occupancy samples efficiently. As a result, our method has shown the robust occupancy representations with efficient computing in two synthetic scenes, compared to the prior methods. Furthermore, we have demonstrated the practical benefits of our adaptive approach in on-the-fly mapping. Compared to the isotropic approaches, our method has shown the accurate occupancy representations of the environment, thanks to the varying shape and size of our adaptive kernel estimations.

In Chapter 4, we have proposed Implicit LiDAR Network (ILN), learning an implicit function for LiDAR range image super-resolution. Inspired by recent implicit network, our network views the LiDAR image as continuous 2D data and predicts the depth at the given query point by taking the depths in neighbor input pixels and interpolating them. However, in contrast to the prior work, our ILN learns the weights for the interpolation and blends the input depth values with possibly non-linear learned weights, which significantly improves reconstruction accuracy particularly for the sharp edge areas. The experiments with our novel large-scale synthetic benchmark demonstrate the outperformance of our method compared with the previous work and the pixel value prediction network. Furthermore, the experiment of occupancy mapping with real LiDAR data demonstrates that the proposed method enables a dense representation of an occupancy map.

We have shown the occupancy mapping approaches exploiting spatial correlation of point clouds for dense representation and real-time update. Nonetheless, the proposed methods can be more optimized by other upcoming occupancy estimators. For example, one can utilize a color image to improve the robustness of our network's points reconstruction, leading to robust occupancy map representation. In addition, the parallel computing power of GPU can enable an occupancy map to achieve real-time update performance. Like these approaches, we hope this dissertation provides meaningful research direction to future work for occupancy mapping.

Chapter 6. Appendix

6.1 Completeness of using the mapping line

Observation 1. *In the 2-dimensional space, if rays have different traversal patterns, these rays are mapped into two different segments, which are divided by a point projected from a grid point of a cell.*

Theorem 1. *All the rays of a super ray generated by mapping line in the 2D space have the same traversal pattern.*

Proof. Our algorithm makes a mapping line by projecting all the grid points within a seed frustum into an arbitrary line. Rays associated with the seed frustum are mapped to a finite region in the line: $[p_0, p_I)$, where p_0 and p_I are the boundary values of the projected seed frustum into the mapping line. Let $P_{proj} = \{p_1, p_2, \ldots, p_{I-1}\}$ be a set of all the projected grid points within the seed frustum; they are within the finite region and assumed to be sorted in the order of distance to p_0 such as $DISTANCE(p_0, p_i) \leq DISTANCE(p_0, p_{i+1})$ for $0 \leq i < I$. The final mapping line computed by our approach consists of a set of segments:

$$ML = \{[p_i, p_{i+1}) | 0 \leq i < I\}$$

(Refer Fig. 6.1-(a)).

Let us assume that rays mapped into the same segment of the mapping line have the different traversal patterns. This indicates that a new grid point should be projected into the segment according to Observation 1. This contradicts that no projected grid point exists within each segment $[p_i, p_{i+1})$ for $0 \leq i < I$. Therefore, rays mapped with the same segment of the mapping line have the same traversal pattern. This proves that all the rays in each super ray generated by our algorithm have the same traversal pattern. □

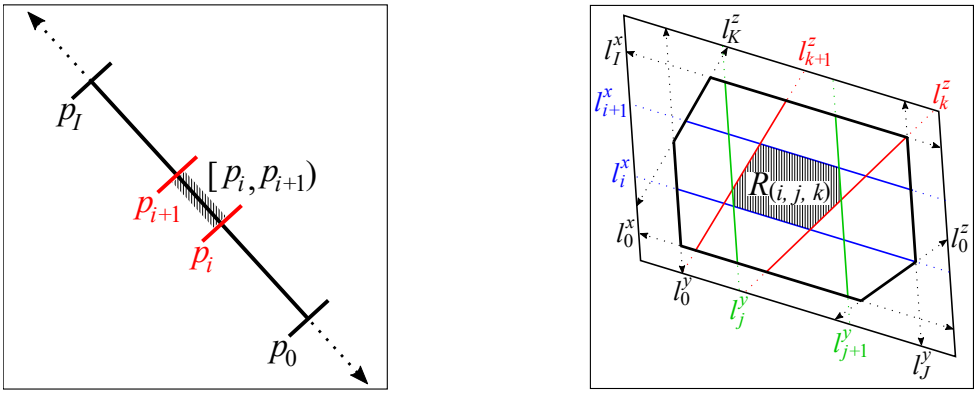
Observation 2. *In the 3-dimensional space, if rays have different traversal patterns, these rays are mapped into three different regions, which are divided by lines projected from edges of a cell.*

Theorem 2. *All the rays of a super ray generated by mapping plane in the 3D space have the same traversal pattern*

Proof. Our algorithm makes a mapping plane by projecting all the edges of grid points within a seed frustum into an arbitrary plane.

Let l_i^x, l_j^y and l_k^z be the lines projected onto the plane from edges aligned to each axis X, Y, and Z respectively. The projected seed frustum on the plane forms a finite region closed by the boundary lines: l_0^x and l_I^x, l_0^y and l_J^y, l_0^z and l_K^z. Let $L_{proj}^x = \{l_1^x, l_2^x, \ldots, l_{I-1}^x\}$, $L_{proj}^y = \{l_1^y, l_2^y, \ldots, l_{J-1}^y\}$, and $L_{proj}^z = \{l_1^z, l_2^z, \ldots, l_{K-1}^z\}$ be sets of projected lines from all the edges of grid points within the seed frustum. The elements of L_{proj}^x and L_{proj}^y are assumed to be sorted in the order of distance to the boundary lines, i.e., $DISTANCE(l_0^x, l_i^x) \leq DISTANCE(l_0^x, l_{i+1}^x)$ for $0 \leq i < I$. In the case of L_{proj}^z, its elements are assumed to be sorted in the order of angle to the boundary line, such as $ANGLE(l_0^z, l_k^z) \leq ANGLE(l_0^z, l_{k+1}^z)$ for $0 \leq k < K$. All the projected lines partition the plane and generate the final mapping plane, MP, consisting of a set of regions; each of which is expressed as $R_{(i,j,k)}$:

$$MP = \{R_{(i,j,k)} | (0 \leq i < I) \wedge (0 \leq j < J) \wedge (0 \leq k < K)\},$$



(a) Segment of mapping line

(b) Region of mapping plane

Figure 6.1: The notations that we use for proofs of Theorem 1 and Theorem 2. The hatched area in the figure (a) shows a segment of mapping line, $[p_i, p_{i+1})$, closed by two projected points(red). In the figure (b), the hatched area represents a region of mapping plane $R_{(i,j,k)}$, closed by the projected lines from edges of grid points.

where a region closed by the lines is defined as follows:

$$R_{(i,j,k)} = \begin{cases} [l_i^x, l_{i+1}^x), \\ [l_j^y, l_{j+1}^y), \\ [l_k^z, l_{k+1}^z) \end{cases}$$

(Refer Fig. 6.1-(b)).

We now assume that rays mapped into the same region of the mapping plane have different traversal patterns, for proof by contradiction. This indicates that a line should be projected within the region according to Observation 2. No projected lines, however, exists within the region $R_{(i,j,k)}$ for $0 \leq i < I$, $0 \leq j < J$, and $0 \leq k < K$, contradicting the assumption. Therefore, rays mapped onto the same region of the mapping plane have the same traversal pattern. □

Bibliography

[1] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz, "On fast surface reconstruction methods for large and noisy point clouds", in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3218–3223.

[2] Adam Leeper, Sonny Chan, and Kenneth Salisbur, "Point clouds can be represented as implicit surfaces for constraint-based haptic rendering", in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 5000–5005.

[3] Peter Biber and Wolfgang Straßer, "The normal distributions transform: A new approach to laser scan matching", in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. IEEE, 2003, vol. 3, pp. 2743–2748.

[4] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur, "A survey of simple geometric primitives detection methods for captured 3d data", in *Computer Graphics Forum*. Wiley Online Library, 2019, pp. 167–196.

[5] Yuval Roth-Tabak and Ramesh Jain, "Building an environment model using depth information", *Computer*, vol. 22, no. 6, pp. 85–90, 1989.

[6] H Moravec, "Robot spatial perceptionby stereoscopic vision and 3d evidence grids", *Perception,(September)*, 1996.

[7] Jane Wilhelms and Allen Van Gelder, "Octrees for faster isosurface generation", *ACM Transactions on Graphics (TOG)*, vol. 11, no. 3, pp. 201–227, 1992.

[8] Pierre Payeur, Patrick Hébert, Denis Laurendeau, and Clément M Gosselin, "Probabilistic octree modeling of a 3d dynamic environment", in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. IEEE, 1997, vol. 2, pp. 1289–1296.

[9] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees", *Autonomous Robots*, 2013.

[10] SAM Coenen, JJM Lunenburg, MJG van de Molengraft, and Maarten Steinbuch, "A representation method based on the probability of collision for safe robot navigation in domestic environments", in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 4177–4183.

[11] Simon O'Callaghan, Fabio T Ramos, and Hugh Durrant-Whyte, "Contextual occupancy maps using gaussian processes", in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 1054–1060.

[12] Simon T O'Callaghan, Fabio T Ramos, and Hugh Durrant-Whyte, "Contextual occupancy maps incorporating sensor and location uncertainty", in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3478–3485.

[13] Simon T O'Callaghan and Fabio T Ramos, "Gaussian process occupancy maps", *The International Journal of Robotics Research*, vol. 31, no. 1, pp. 42–62, 2012.

[14] Soohwan Kim and Jonghyuk Kim, "Building occupancy maps with a mixture of gaussian processes", in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 4756–4761.

[15] Soohwan Kim and Jonghyuk Kim, "Continuous occupancy maps using overlapping local gaussian processes", in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 4709–4714.

[16] Soohwan Kim and Jonghyuk Kim, "Gpmap: A unified framework for robotic mapping based on sparse gaussian processes", in *Field and service robotics*. Springer, 2015, pp. 319–332.

[17] Maani Ghaffari Jadidi, Jaime Valls Miro, and Gamini Dissanayake, "Warped gaussian processes occupancy mapping with uncertain inputs", *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 680–687, 2017.

[18] Fabio Ramos and Lionel Ott, "Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent", *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[19] Vitor Guizilini and Fabio Ramos, "Large-scale 3d scene reconstruction with hilbert maps", in *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016, pp. 3247–3254.

[20] Ransalu Senanayake, Lionel Ott, Simon O'Callaghan, and Fabio T Ramos, "Spatio-temporal hilbert maps for continuous occupancy representation in dynamic environments", in *Advances in Neural Information Processing Systems*, 2016, pp. 3925–3933.

[21] Ransalu Senanayake and Fabio Ramos, "Bayesian hilbert maps for dynamic continuous occupancy mapping", in *Conference on Robot Learning*, 2017, pp. 458–471.

[22] Vitor Guizilini and Fabio Ramos, "Towards real-time 3d continuous occupancy mapping using hilbert maps", *The International Journal of Robotics Research*, vol. 37, no. 6, pp. 566–584, 2018.

[23] Ransalu Senanayake, Anthony Tompkins, and Fabio Ramos, "Automorphing kernels for nonstationarity in mapping unstructured environments.", in *CoRL*, 2018, pp. 443–455.

[24] Vitor Guizilini, Ransalu Senanayake, and Fabio Ramos, "Dynamic hilbert maps: Real-time occupancy predictions in changing environments", in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 4091–4097.

[25] Weiming Zhi, Lionel Ott, Ransalu Senanayake, and Fabio Ramos, "Continuous occupancy map fusion with fast bayesian hilbert maps", in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 4111–4117.

[26] Jinkun Wang and Brendan Englot, "Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion", in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1003–1010.

[27] Kevin Doherty, Jinkun Wang, and Brendan Englot, "Probabilistic map fusion for fast, incremental occupancy mapping with 3d hilbert maps", in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1011–1018.

[28] Kevin Doherty, Jinkun Wang, and Brendan Englot, "Bayesian generalized kernel inference for occupancy map prediction", in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3118–3124.

[29] Kevin Doherty, Tixiao Shan, Jinkun Wang, and Brendan Englot, "Learning-aided 3-d occupancy mapping with bayesian generalized kernel inference", *IEEE Transactions on Robotics*, 2019.

[30] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun, "Vision meets robotics: The kitti dataset", *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[31] John Amanatides, Andrew Woo, et al., "A fast voxel traversal algorithm for ray tracing", in *Eurographics*, 1987, vol. 87, p. 10.

[32] Hans P Moravec and Alberto Elfes, "High resolution maps from wide angle sonar", in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*. IEEE, 1985, vol. 2, pp. 116–121.

[33] Manuel Yguel, Olivier Aycard, and Christian Laugier, "Update policy of dense maps: Efficient algorithms and sparse representation", in *Field and Service Robotics*. Springer, 2008, pp. 23–33.

[34] Elizbar A Nadaraya, "On estimating regression", *Theory of Probability & Its Applications*, vol. 9, no. 1, pp. 141–142, 1964.

[35] Geoffrey S Watson, "Smooth regression analysis", *Sankhyā: The Indian Journal of Statistics, Series A*, pp. 359–372, 1964.

[36] Artur Gramacki, *Nonparametric kernel density estimation and its computational aspects*, Springer, 2018.

[37] Arman Melkumyan and Fabio Tozeto Ramos, "A sparse covariance function for exact gaussian process inference in large datasets", in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[38] Tixiao Shan and Brendan Englot, "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)*. IEEE, 2018, pp. 4758–4765.

[39] Cormac O'Meadhra, Wennie Tabib, and Nathan Michael, "Variable resolution occupancy mapping using gaussian mixture models", *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2015–2022, 2018.

[40] Martin Magnusson, Achim Lilienthal, and Tom Duckett, "Scan registration for autonomous mining vehicles using 3d-ndt", *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827, 2007.

[41] Jari P Saarinen, Henrik Andreasson, Todor Stoyanov, and Achim J Lilienthal, "3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments", *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1627–1644, 2013.

[42] Cornelia Schulz, Richard Hanten, and Andreas Zell, "Efficient map representations for multi-dimensional normal distributions transforms", in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 2679–2686.

[43] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang, "Image super-resolution using deep convolutional networks", *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[44] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee, "Accurate image super-resolution using very deep convolutional networks", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.

[45] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu, "Residual dense network for image super-resolution", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2472–2481.

[46] Tixiao Shan, Jinkun Wang, Fanfei Chen, Paul Szenher, and Brendan Englot, "Simulation-based lidar super-resolution for ground vehicles", *Robotics and Autonomous Systems*, vol. 134, pp. 103647, 2020.

[47] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger, "Occupancy networks: Learning 3d reconstruction in function space", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4460–4470.

[48] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger, "Convolutional occupancy networks", in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 2020, pp. 523–540.

[49] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 165–174.

[50] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis", in *European conference on computer vision*. Springer, 2020, pp. 405–421.

[51] Christoph Rist, David Emmerichs, Markus Enzweiler, and Dariu Gavrila, "Semantic scene completion using local deep implicit functions on lidar data", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[52] Yinbo Chen, Sifei Liu, and Xiaolong Wang, "Learning continuous image representation with local implicit image function", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8628–8638.

[53] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al., "An image is worth 16x16 words: Transformers for image recognition at scale", in *International Conference on Learning Representations*, 2020.

[54] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun, "CARLA: An open urban driving simulator", in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[55] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[56] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[58] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee, "Enhanced deep residual networks for single image super-resolution", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need", in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[60] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

Acknowledgments in Korean

자율로봇의 인공지능을 연구하겠다는 목표로 시작한 대학원 과정이 다사다난했던 8년이라는 오랜 기간을 거쳐 마무리되었습니다. 로봇학제전공으로 시작하여 전산학부로 졸업하기까지, 윤성의 교수님의 끊임없는 연구 지도와 지원 덕분에 원했던 연구를 마칠 수 있었습니다. 여러 국제/국내 학회에 참가하며 연구 결과를 발표하는 등, 스스로가 성장할 수 있도록 많은 기회를 주신 것에 큰 감사의 말씀을 드립니다. 쉽지 않은 학위 과정이었지만 연구실에서 함께 지냈던 선후배 및 동기분들이 있었기에 좋은 추억들과 함께 즐거운 대학원 생활을 보낼 수 있었습니다. 특히, 오랜 시간 같이 지내면서, 로봇팀을 이끌어주고 기타도 가르쳐줬던 김동혁 박사님과, 팀은 달랐지만 게임도 즐기며 함께 연구했던 김수민 박사님에게 감사드립니다. 그리고, 지금도 열심히 연구하는 김태영, 강민철 형, 안인규 형에게 덕분에 재밌던 추억을 많이 간직할 수 있게 되어 고맙다는 말을 전하고 싶습니다.

SGLAB에서 SGVR Lab.으로 연구실 이름이 바뀌는 동안, 정말 많은 사람들과 기쁘거나 힘든 일들을 공유했던 경험이 저에겐 너무 큰 행운이었습니다. 연구를 시작하면서 알지 못했던 것들을 잘 알려주었던 이정환, 김덕수, 문보창, 허재필, Pio Claudio 박사님들, 그리고 연구실에 함께 들어와 같이 대학원 과정을 시작했던 윤웅직 형, 양현철 형, 이윤석 형 덕분에 연구실 생활을 빨리 적응하고 지금까지 잘 지낼 수 있었기에 정말 감사드립니다. 또한, 이윤석 형, 윤정수 형, 권영기, 송치완 등 졸업생 선후배들과 함께 수업도 듣고 서로 의지할 수 있어서 고마웠습니다. 앞으로 연구에 매진할 임우빈, 신희찬, 김재윤, 정준식, 김민철, 조윤기 및 연구실 후배들에게도 응원의 말을 전합니다. Thanks to Dr. Huo, Xu Yin, and Guoyuan An. Good luck to you! 그리고, 연구에 집중할 수 있도록 도와주신 김슬기나 선생님에게 감사드립니다.

영준이와 대학원 시작부터 끝까지 같이 기숙사 룸메이트로 지냈는데, 덕분에 대학원 과정을 잘 마칠 수 있어 고마웠고 앞으로의 미래도 응원합니다. 마지막으로, 저를 끝까지 믿어주시고 지원해주셨던 부모님과 동생에게 감사의 말씀을 전하고 싶습니다. 좋은 일이 있을 때 같이 기뻐하고, 곤란한 일이 생기더라도 공감하고 이해해주신 덕분에 박사학위 과정을 잘 마칠 수 있었습니다.

앞으로 새로운 환경에서 도전하겠지만, 대학원 과정을 밑거름 삼아 지혜롭게 이겨내고 성과를 거둘 수 있도록 노력하겠습니다. 감사합니다.

Curriculum Vitae in Korean

이　　름: 권 용 선

학　　력

2010. 3. – 2014. 2.　성균관대학교 전자전기공학부, 컴퓨터공학과 (학사, 복수전공)

2014. 3. – 2016. 2.　한국과학기술원 로봇공학학제전공 (석사)

2016. 3. – 2022. 2.　한국과학기술원 전산학부 (박사)

연 구 업 적

1. **Youngsun Kwon**, Minhyuk Sung, and Sung-Eui Yoon, *Implicit LiDAR Network: LiDAR Super-Resolution via Interpolation Weight Prediction*, (Under review)
2. Inkyu An, **Youngsun Kwon**, and Sung-Eui Yoon, *Diffraction and Reflection Aware Multiple Sound Source Localization*, IEEE Transactions on Robotics (T-RO), 2021
3. Moonyoung Lee, **Youngsun Kwon**, Sebin Lee, JongHun Choe, Junyong Park, Hyobin Jeong, Yujin Heo, Min-su Kim, Jo Sungho, Sung-Eui Yoon, and Jun-Ho Oh, *Dynamic Humanoid Locomotion over Rough Terrain with Streamlined Perception-Control Pipeline*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021
4. **Youngsun Kwon**, Bochang Moon, and Sung-Eui Yoon, *Adaptive Kernel Inference for Dense and Sharp Occupancy Grids*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020
5. Inkyu An, Byeongho-Jo, **Youngsun Kwon**, Jung-woo Choi, and Sung-Eui Yoon, *Robust Sound Source Localization considering Similarity of Back-Propagation Signals*, IEEE International Conference on Robotics and Automation (ICRA), 2020
6. Taeyoung Kim, **Youngsun Kwon**, and Sung-Eui Yoon, *Real-time 3-D Mapping with Estimating Acoustic Materials*, IEEE/SICE International Syposium of System Integration (SII), 2019
7. Hongsun Choi, Mincheul Kang, **Youngsun Kwon**, and Sung-Eui Yoon, *An Objectness Score for Accurate and Fast Detection during Navigation*, The World Congress on Advances in Nano, Bio, Robotics and Energy (ANBRE), 2019
8. **Youngsun Kwon**, Donghyuk Kim, Inkyu An, and Sung-Eui Yoon, *Super Rays and Culling Region for Real-Time Updates on Grid-based Occupancy Maps*, IEEE Transactions on Robotics (T-RO), 2019
9. Donghyuk Kim, **Youngsun Kwon**, and Sung-Eui Yoon, *Adaptive Lazy Collision Checking for Optimal Sampling-based Motion Planning*, International Conference on Ubiquitous Robots (UR), 2018

10. Mincheul Kang, **Youngsun Kwon**, and Sung-Eui Yoon, *Automated Task Planning using Object Arrangement Optimization*, International Conference on Ubiquitous Robots (UR), 2018
11. Donghyuk Kim, **Youngsun Kwon**, and Sung-Eui Yoon, *Dancing PRM* : Simultaneous Planning of Sampling and Optimization with Configuration Free Space Approximation*, IEEE International Conference on Robotics and Automation (ICRA), 2018
12. **Youngsun Kwon** and Sung-Eui Yoon, *Ray Distribution to Parallel Batching-based Updates*, IEEE International Conference on Robotics and Automation (ICRA) Workshop on Robotics and Vehicular Technologies for Self-driving cars, 2017
13. **Youngsun Kwon**, Donghyuk Kim, and Sung-Eui Yoon *Super Ray-based Updates for Occupancy Maps*, IEEE International Conference on Robotics and Automation (ICRA), 2016