Self-supervised Semantic Learning of LiDAR Point Clouds for Large-scale Scene Understanding

by

Haowei Zhang

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Institute for Aerospace Studies
University of Toronto

Abstract

Self-supervised Semantic Learning of LiDAR Point Clouds for Large-scale Scene Understanding

Haowei Zhang
Master of Applied Science
Graduate Department of Institute for Aerospace Studies
University of Toronto
2021

Semantic segmentation is a challenging task in the robotic vision community to classify various objects in a scene. While recent works employ supervised deep learning approaches using hand-annotated examples, these training samples are often costly to obtain.

In this thesis, we present a self-supervised semantic learning method for large-scale scene understanding. Our offline method retrieves point cloud annotations by combining a mapping and localization solution with ray-tracing algorithms. Through multi-session navigation experiences in the same environment, our method labels points as members of four semantic classes: ground, non-movable, long-term movable, and short-term movable. These semantic labels allow us to train a semantic segmentation network without any hand annotations, which can then be used online to remove dynamic points in point clouds.

For a qualitative and quantitative analysis, we demonstrate our method on a simulation dataset. We also provide a qualitative evaluation on a real-world dataset. Furthermore, by semantically filtering out movable points, results show that our method improves existing localization performance.

Acknowledgements

I am grateful for different groups of people who support me through the completion of this thesis. They provide me with great support during an important transition in my academic career and personal life.

I would like to give special thanks to my supervisor, Dr. Tim Barfoot, for his expert guidance and great passion throughout the degree in helping me grow as a researcher.

Thank you David Yoon and Hugues Thomas for your support in always being open to research discussions and innovative ideas. I would also like to thank all the members of Autonomous Space Robotics Lab for creating a friendly atmosphere through the hard times.

A special thanks to Keith Leung and Andrew Lambert from Applanix Corporation in providing technical help and research advice throughout my thesis.

Finally, tremendous thanks to my parents and my girlfriend Yuxuan Li for everyday support and motivations that carried me through the degree. I would not have made it without you.

Contents

List of Tables

List of Figures

Notation

$\underrightarrow{\boldsymbol{\mathcal{F}}}_a$	A reference frame for a three-dimensional coordinate system
$SE(3)$	The special Euclidean group, a matrix Lie group used to represent poses
$\mathfrak{se}(3)$	The Lie algebra associated with $SE(3)$
$\mathbf{T}_{a,b}$	A matrix in $SE(3)$ that transforms vectors from frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ to frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_a$
$\mathbf{T}_k$	Short form for $\mathbf{T}_{i,k}$, a matrix in $SE(3)$ that transforms vectors from $\underrightarrow{\boldsymbol{\mathcal{F}}}_k$, the frame representing the pose at time t_k, to $\underrightarrow{\boldsymbol{\mathcal{F}}}_i$, the inertial frame
$\exp(\cdot^\wedge)$	A Lie algebra operator mapping from $\mathfrak{se}(3)$ to $SE(3)$
$\ln(\cdot)^\vee$	A Lie algebra operator mapping from $SE(3)$ to $\mathfrak{se}(3)$
$\mathbf{1}$	The identity matrix
$\mathbf{0}$	The zero matrix

Acronyms

LiDAR Light Detection And Ranging

FOV Field Of View

MLP Multi-Layer Perceptron

SLAM Simultaneous Localization And Mapping

ICP Iterative Closest Point

NDT Normal Distributions Transform

DOF Degree Of Freedom

EM Expectation Maximization

SAM Smoothing And Mapping

BEV Bird Eye View

GPS Global Positioning System

IMU Inertial Measurement Unit

Chapter 1

Introduction

1.1 Motivation

Using the current state-of-the-art autonomous navigation stack, we can autonomously navigate a robot in a predictable environment. A good example is visual teach and repeat [22], a system that enables long-range rover autonomy with an on-board stereo camera sensor. However, our navigation stack falls short in performance when the environment becomes more complex and unpredictable. Therefore, we believe that it is crucial for a robot to retrieve and analyze semantic information from the environment it interacts with to improve scene understanding. Depending on the needs of particular tasks, semantic information can be used in different ways to improve the task performance. One may choose to filter out points belonging to dynamic objects from the map when building a mapping algorithm, and an object detection algorithm may prefer to focus on points that are identified as dynamic points. A good example is the work by Pagad et al. [42], which describes a dynamic object removal algorithm that uses object detection result to remove dynamic points from a point cloud map.

Cameras create a visual representation of the world and are the most common sensors employed on an autonomous robot thanks to their inexpensive cost. Recent literature addresses scene understanding in camera images [2, 50, 39]. However, the disadvantage of using camera images for scene understanding is that they are sensitive to poor illuminations and require a stereo setup to estimate depth, which is critical in understanding

the 3D geometry. On the opposite side, LiDAR (Light Detection and Ranging) provides accurate depth information and remains invariant to challenging lighting conditions, despite its relatively expensive cost and sparsity.

In the recent literature, supervised deep learning approaches have been demonstrated to learn semantic information from LiDAR data [46, 44, 45, 54, 38]. These methods represent LiDAR data in the form of either a point cloud or a LiDAR image, and learn to assign a semantic label to each point to represent to what semantic class it belongs. Even though these approaches demonstrate excellent performance in scene understanding, they fall short in that all of them require groundtruth annotations for training, which are often expensive to obtain.

In this thesis, we are interested in acquiring annotations using an automated approach to avoid hand annotations. These annotations would be used to train a semantic segmentation network offline, which could then be used for online scene understanding.

1.2 Contributions

In this thesis, we introduce a self-supervised approach to semantically segment LiDAR point clouds for large-scale scene understanding. The foundation of our approach is the pipeline proposed by Thomas et al. [53]. We extend the pipeline to large-scale scenes, especially autonomous driving scenarios. Combined with a mapping and localization solution, our approach acquires point level annotations using an automated annotation pipeline and learns the semantic segmentation of LiDAR point clouds using the latest LiDAR input. To qualitatively and quantitatively evaluate our method, we collect a simulation dataset using the CARLA simulator. We also make qualitative analysis on a real-world dataset to validate our approach.

The main contributions of our proposed method are:

- a self-supervised semantic learning approach for point cloud scene understanding in a large-scale environment

- a simulation dataset that contains multi-session navigation experiences in the same

environment and under multiple different self-driving scenes

- a qualitative and quantitative analysis of our method on the simulation dataset
- a qualitative analysis of our method on the real-world dataset

1.3 High-Level Overview

The thesis is organized as follows. Chapter 2 presents background information on common LiDAR data representations, existing mapping and localization methods, and existing semantic segmentation approaches for LiDAR point clouds. In Chapter 3, we introduce our self-supervised semantic learning approach for large-scale scene understanding. In Chapter 4, we provide more details on the simulation dataset we collect using the CARLA simulator and a real-world LiDAR dataset collected using Velodyne VLS-128. In Chapter 5, we show our experimental results and make qualitative and quantitative analysis on both datasets. Finally, in Chapter 6 we draw our concluding remarks and propose future research directions.

Chapter 2

Background

We describe background and relevant literature in this chapter. We begin by describing LiDAR data and different forms of representations commonly used in robotics applications. Next, we discuss existing literature on LiDAR-based mapping and localization. Then, we move our focus onto LiDAR semantic learning and end the chapter by looking at how existing literature improves mapping and localization by exploiting the semantics.

2.1 LiDAR Data Representations

LiDAR, short for Light Detection and Ranging, has become a commonly used sensing component for any robotics applications in the past decades. Laser beams inside the LiDAR unit emit pulsed light waves into the surrounding environment and use the time for each pulse to return to calculate the distance each pulse travels. Repeating this process millions of times per second creates a precise, real-time 3D map of an environment. There are mainly two types of LiDAR in the market for robotics applications, mechanically spinning LiDAR and solid-state LiDAR. This thesis focuses on mechanically spinning LiDAR as it is the more common type of LiDAR for autonomous navigation to this date. Therefore, from now on, we refer to mechanically spinning LiDAR as *LiDAR*. Furthermore, we also consider a laser beam emitting pulsed light waves as a *firing* to clarify the term carefully.

Multiple configuration parameters are essential to a LiDAR product, including hor-

izontal and vertical field-of-view (FOV), horizontal and vertical resolutions. LiDAR's continuous sweep of the environment provides a 360-degree horizontal FOV. Based on the physical placement of laser beams, different LiDAR products offer different vertical FOV and resolutions. Meanwhile, the horizontal resolution of a LiDAR depends on laser beam firing frequency in operation.

Velodyne is a top supplier of LiDAR products and has pioneered the high-performance LiDAR for automotive use. It offers different configurations, including 16, 32, 64 and 128 lasers, for research and industry use. Although more companies (e.g., Ouster and Hesai) have released similar LiDAR products to this date, Velodyne LiDAR remains the leader in providing state-of-the-art LiDAR for autonomous navigation. This thesis uses a Velodyne Alpha Prime with 128 lasers to collect a dense and high-resolution real-world dataset. From now on, we refer to Velodyne Alpha Prime as Velodyne VLS-128 for simplification purposes.

2.1.1 Range & Bearing

In Figure 2.1 we define the LiDAR base frame $\underrightarrow{\mathcal{F}}_b$ rigidly attached to the robot platform. When each laser beam fires and receives the returned signal, it takes a range measurement r of the endpoint p to the origin of $\underrightarrow{\mathcal{F}}_b$ based on time of travel. We often use bearing angle to represent the laser beam's relative orientation in $\underrightarrow{\mathcal{F}}_b$, but bearing angle is a term usually associated with the angle in a 2D quadrant defined by the cardinal directions. To represent the 3D spherical measurement, we separate bearing into azimuth and elevation angle.

In Figure 2.1 we define each azimuth angle α to be measured in the X-Y plane from the Y-axis and elevation angle ω to be measured in the Y-Z plane from the Y-axis. We know each laser beam's azimuth and elevation angle at the time of firing with respect to LiDAR base frame $\underrightarrow{\mathcal{F}}_b$. Therefore, we can represent LiDAR data using the range, azimuth, and elevation measurements, which are commonly referred to as range and bearing. Furthermore, this thesis refers to the range and bearing measurements as the raw measurements from a LiDAR sensor because they are the unprocessed output.

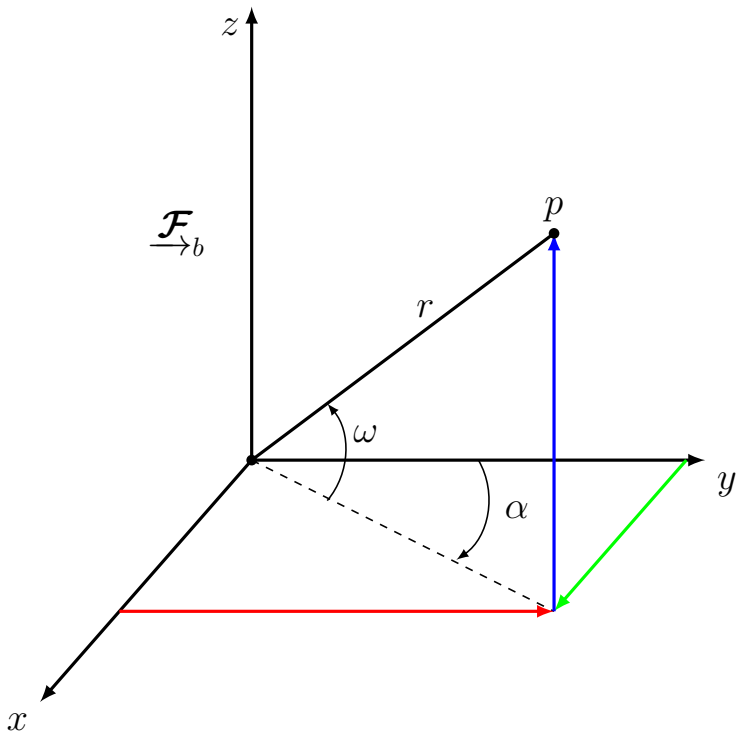


Figure 2.1: An illustration of a measurement p in a LiDAR base frame $\underrightarrow{\mathcal{F}}_b$ when involved in converting from a range and bearing measurement to a 3D LiDAR point. Range measurement r defines the Euclidean distance from p to the origin of $\underrightarrow{\mathcal{F}}_b$. Azimuth angle α is measured in the X-Y plane from the Y-axis whereas elevation angle ω is measured in the Y-Z plane from the Y-axis. The red, green and blue directed lines represent x, y and z coordinates of point p in a 3D space.

2.1.2 Point Cloud

A point cloud is a collection of points formed by each laser beam's hit positions on a physical object. Every point in a point cloud consists of x, y, and z coordinates, with additional intensity information based on physical properties of the material and incidence angle, etc. Every point is computed from a range and bearing measurement from the last section. We can explain this conversion process more clearly with mathematical notations. From now on, we carefully define a *scan* to be a point cloud containing point measurements during a full 360° sweep of an environment.

Figure 2.1 shows an example of a point captured by a LiDAR in the LiDAR base frame $\underrightarrow{\mathcal{F}}_b$. As discussed in the last section, the raw measurements for each point include range r, elevation ω and azimuth α with respect to $\underrightarrow{\mathcal{F}}_b$. Therefore, to convert these raw

measurements to a point cloud representation, the corresponding x, y, and z coordinates in $\underrightarrow{\mathcal{F}}_b$ are

$$\begin{aligned} x &= r\cos(\omega)\sin(\alpha) \\ y &= r\cos(\omega)\cos(\alpha) \\ z &= r\sin(\omega) \end{aligned} \tag{2.1}$$

The point cloud is the most common data representation for LiDAR data, but it is often misinterpreted as an instantaneous snapshot of the environment. Due to the moving-while-scanning operations of LiDAR sensors, laser beams emit pulses sequentially at slightly different timestamps during a sweep of the environment. Unlike most cameras that scan the entire environment simultaneously, if a LiDAR is mounted on a moving platform, its observed scene would be easily distorted due to the rolling shutter effect, which we often refer to as motion distortion.

We emphasize the term motion compensation as the method to correct motion distortion, and we discuss it in more detail in Chapter 3. We can easily identify distinct discontinuities near the start and end of the motion distorted scans because the start and end are captured at slightly different timestamps. Motion compensation aligns points in a motion distorted scan to the same timestamp. It is an essential pre-processing step for point clouds before proceeding with further operations.

Point clouds are widely used in many computer vision and robotics applications, including deep learning research. Unlike camera images, point clouds represent geometric data in an irregular format and thus are often difficult to apply the traditional convolution operation for deep learning. We mention existing literature in this section that addresses point cloud representation within a deep learning framework and introduces the use of point clouds in robotics applications. Su et al. [52] try to render 3D point clouds into 2D images and apply 2D convolutional networks for classification. However, this rendering strategy is unnecessarily voluminous and causes issues. To efficiently represent point clouds, Qi et al. [44, 45] pioneer the use of unordered point sets on point clouds and relies on Multi-Layer Perceptron (MLP) to keep the permutation invariance.

The method demonstrates its state-of-the-art performance for classification and semantic segmentation tasks. With a similar motive, Thomas et al. [54] convolve point clouds directly with spherical kernel points to achieve kernel convolution. The method outperforms state-of-the-art classification and semantic segmentation approaches in several datasets. In addition to semantic learning, point clouds are often used as measurements for autonomous navigation. Zhang et al. [66] propose point cloud features that aid keyframe-based Simultaneous Localization and Mapping (SLAM), analogous to how visual features are often detected and matched in a visual odometry pipeline. As opposed to manually extracting point cloud features, Yoon et al. [64] introduce unsupervised learning of trajectory estimation by maximizing point cloud data likelihood through the learned keypoint detector and descriptors. Finally, point clouds can be processed to accomplish more vision-oriented tasks. Yuan et al. [65] feature a decoder-based network to address the shape completion problem using point cloud data. The method estimates the complete scene geometry from partial observations. Drawing inspirations from the natural language processing community, Pan et al. [43] apply Transformer [57] to point clouds for 3D object detection tasks.

We emphasize that in this thesis we work extensively with the point cloud representation of LiDAR data as it is most commonly used in robotics applications.

2.1.3 LiDAR Image

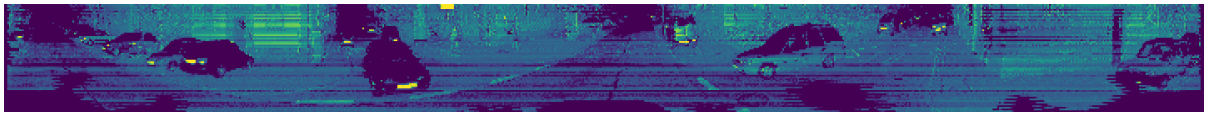
We can use elevation and azimuth measurements to construct a LiDAR image. It can be seen as an image coordinate-parametrized representation of LiDAR data. Consecutive measurements from each laser beam in a LiDAR scan comprise an individual row in a LiDAR image, and row ordering depends on each laser beam's elevation angle. If every entry in a LiDAR image contains an intensity channel of the associated measurement, we refer to it as a LiDAR intensity image. Similarly, we define a LiDAR range image if every entry contains the range measurement.

Because laser beams fire sequentially at slightly different timestamps during a sweep of the environment, if we are to fill each column based on the point timestamp, the columns would not align perfectly. To address this issue and also satisfy hardware configuration

and task-specific requirements, a LiDAR image is often uniformly down-sampled in the row direction based on an azimuth resolution. We elaborate on the down-sampling with more clear mathematical expressions. For a sweep of the environment, a LiDAR image would contain r_N entries in each row:

$$r_N = \frac{360^\circ}{\alpha_{\text{res}}}, \tag{2.2}$$

where α_{res} is the azimuth resolution in degrees. A common value $\alpha_{\text{res}} = 0.5^\circ$ is often empirically chosen to retain sufficient LiDAR image resolution while avoiding the missing pixel issue. Figure 2.2(a) and Figure 2.2(b) show an example of LiDAR intensity and range images produced from a 64-line Velodyne HDL-64.



(a) LiDAR intensity image

(b) LiDAR range image

Figure 2.2: LiDAR intensity and range images produced from a 64-line Velodyne HDL-64. The images both have 64 rows with each row representing measurements from one laser beam. We choose azimuth resolution $\alpha_{res} = 0.5^\circ$ so that it has $\frac{360^\circ}{0.5^\circ} = 720$ pixels on each row. We can observe clear contours of the vehicles and high-intensity returns from vehicle plates through this representation form.

The LiDAR image is often used in recent literature thanks to its efficient data representations. Unlike point cloud data, LiDAR image is a compact 2D representation of the 3D geometry world. Dong et al. [18] introduce LiDAR intensity image to match visually distinct features to recover motion of a ground vehicle, similar to how camera images are processed in a visual odometry pipeline. To emphasize this efficient representation in learning, Cho et al. [15] propose an unsupervised LiDAR odometry framework to learn distinct features from stacked LiDAR intensity and range images through a deep learning network with competing performance. Apart from the motion estimation, Milioto et al. [38] use LiDAR range images to learn a state-of-the-art semantic segmentation network,

unlike prevalent point cloud based approaches.

Although point clouds provide rich 3D geometry information, their data processing requires extensive computation resources, which are often absent on a mobile platform aiming at fast inference. As an alternative approach to represent LiDAR data, the LiDAR image has the benefit of being computationally friendly to large-scale scenes, thanks to the nature of its 2D representation. A good example is *RangeNet++*. Milioto et al. [38] empower RangeNet++ with LiDAR range image input to achieve more than 20 times faster inference than a point cloud based segmentation network, which makes it more applicable for real-world deployment.

2.2 Mapping and Localization

Mapping and localization refer to the process of constructing a map and localizing the vehicle within it. We explicitly distinguish this from Simultaneous Localization and Mapping (SLAM) because our pipeline does not require mapping and localization to happen simultaneously. This section describes common components in a LiDAR-only mapping and localization pipeline, without any use of a global positioning system (GPS) and inertial measurement unit (IMU).

2.2.1 Mapping

Mapping refers to constructing or updating a map of an unknown environment. Many mapping algorithms rely on a combination of odometry measurements, loop closure and pose graph optimization techniques. We describe each component and discuss the existing literature in this section.

2.2.1.1 Scan-to-scan odometry

Scan-to-scan odometry estimates relative pose change between two consecutive scans in a LiDAR-only mapping pipeline. It is significant to the mapping pipeline because it introduces odometry constraints between consecutive scans. The odometry measurements would later be optimized through a pose graph that we would describe in a later section.

LiDAR odometry is a well-explored problem in the robotics community. We carefully place existing approaches into two categories: classic approach and deep learning approach.

Iterative Closest Point (ICP) is the most common classic approach when it comes to scan-to-scan odometry. Arun et al. [3] and Besl et al. [8] introduce standard point-to-point ICP that iteratively finds an optimal transformation solution by minimizing the Euclidean distance between corresponding points. To achieve real-time performance, correspondences between two scans are computed using a KD-tree. As a more robust variant of the standard ICP, the point-to-plane ICP introduced by Chen and Medioni et al. [13] improves the scan matching performance by taking advantage of the surface normal information. The point-to-plane error metric minimizes the error along the surface normal direction as opposed to Euclidean distance between matched points. As demonstrated in Figure 2.3, source points s_1, s_2 and s_3 try to find an optimal transformation to minimize distance to tangent plane of the destination points d_1, d_2 and d_3. As a further extension, Segal et al. [47] combine standard ICP and point-to-plane ICP into a single probabilistic framework to model locally planar surface structure from both scans. This algorithm can be thought of as introducing another plane-to-plane error metric. Despite incremental improvements to the standard ICP, the fallback of most ICP-based approaches is that it requires a decent initial guess, which is unavailable in some cases.

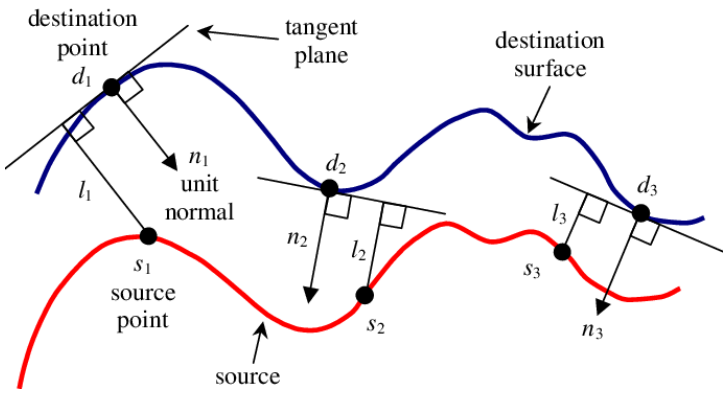


Figure 2.3: Point-to-plane ICP tries to find an optimal transformation that transforms source point (red) to destination point (blue). The point-to-plane metric minimizes the distance l from source points to the tangent plane of destination points. Image credit: Low et al. [33].

In addition to ICP and its variants, Normal Distributions Transform (NDT) is another classic approach that represents point clouds as differential multi-variate Gaussian distributions. Biber et al. [9] introduce this approach to sub-divide point clouds into cells and locally model the probability of point measurements. The method uses differentiable probability densities to match two scans with no explicit correspondences to be established. Magnusson et al. [36] show that NDT is more robust to poor initialization than ICP-based approaches yet the subdivision process could lead to problems due to discontinuities in surface representations.

Both ICP-based approaches and NDT operate on a down-sampled point cloud due to limitations on computational resources. As opposed to uniform down-sampling, an alternative is to extract point cloud features from consecutive scans to do feature matching, similar to a visual odometry approach. Zhang et al. [66] introduce extracting edge and planar features in point clouds based on local curvature and the ring number information. The method matches the detected features across consecutive scans and manages to achieve fast and accurate scan-to-scan odometry. An example of extracted point cloud features is demonstrated in Figure 2.4. Using a similar idea, Shan et al. [48] apply a point cloud segmentation technique to filter out noise in the point clouds prior to extracting distinctive edge and planar features. These hand-designed point cloud features are fast to compute and often used for real-time performance.

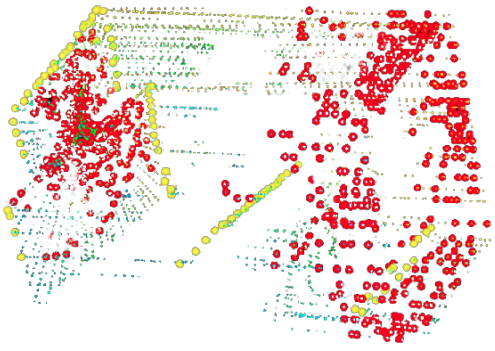


Figure 2.4: Extracted edge (yellow) and planar (red) features from a point cloud taken in a corridor [66]. Based on local curvature information, Zhang et al. [66] extracts sharp and flat points from point clouds. Image credit: Zhang et al. [66].

Although classic approaches have proved to be robust and computationally desirable

in most applicable scenarios, the enormous engineering efforts behind these approaches motivate the research community to investigate deep learning approaches. Deep learning approaches for LiDAR odometry often consume two point cloud scans as input and output a six degree-of-freedom (DOF) pose to represent the relative pose change. Li et al. [32] learn the complete odometry pipeline end-to-end by introducing a mask-weighted geometric constrained loss. This supervised method manages to achieve competent performance as compared to ICP-based approaches. To take the end-to-end learning even further, Cho et al. [14] propose an end-to-end unsupervised learning framework that uses LiDAR vertex and normal images as input and outputs a six DOF relative pose change. This is the first deep learning method that learns LiDAR odometry without groundtruth poses yet its odometry performance falls short compared to LO-Net [32] and other ICP-based methods.

As opposed to the fully learned methods, there are methods that combine the benefits from both classic and deep learning approaches. Chen et al. [12] improve on the classic SLAM framework Suma [7] by incorporating a pre-trained semantic segmentation network to remove dynamic points in the scan. Partly inspired by Suma++ [12], Yoon et al. [64] combine classic and deep learning approaches and introduce an unsupervised framework that can be trained end-to-end for trajectory estimation. The method learns keypoints and associated descriptors along with the 6×6 covariance by maximizing the point cloud data likelihood through Expectation-Maximization (EM). During inference, the learned keypoints and covariance are passed into a classic framework for batch trajectory estimation. The highlight of this approach is that it not only achieves better odometry performance than earlier mentioned deep learning methods but also can be trained end-to-end without groundtruth poses.

2.2.1.2 Motion Compensation

In Chapter 2.1.2, we introduce motion distortion as a common issue with point clouds due to the moving-while-scanning operation of LiDAR sensors. Motion compensation is important in mapping and localization because distorted point clouds affect map quality when aggregating multiple scans into a map. In this section, we describe existing

literature to address motion compensation.

Motion distortion is an issue because points in a scan fire at slightly different timestamps. To address this issue, Tong et al. [55] and Merali et al. [37] suggest stopping the vehicle occasionally to conduct the scan. However, this does not solve the issue from the root. From a completely different perspective, Sheehan et al. [49] introduce the continuous-time method to estimate robot motion at any given time. While the earlier method assigns one timestamp for every LiDAR scan, this method allows the robot to query six DOF poses for any laser points. Furthermore, Anderson et al. [1] formulate a continuous-time trajectory estimation framework with a physically motivated constant velocity motion prior. The framework enables motion compensation for scan-to-scan odometry using LiDAR data.

2.2.1.3 Keyframes

Scan-to-scan odometry produces relative pose change between consecutive scans. However, over a long trajectory, compounded relative poses from odometry solution tend to drift and cause dead reckoning. To mitigate this issue, existing literature introduces the concept of keyframes, also known as submaps.

The keyframe refers to a reference frame used by a SLAM system to localize within an environment. Generally, it contains data captured at a specific timestamp. In the context of LiDAR-only mapping and localization, a keyframe is usually a motion-compensated LiDAR scan at a specific timestamp. To retain global consistency and accuracy during mapping, scans are matched against their nearest keyframes and keyframes are collected along the robot trajectory to build a pose graph.

The concept of keyframes stems from a visual SLAM background and is often compared with the classic filter-based approaches. Strasdat et al. [51] compare the classic filtering approach with the keyframe-based approach in a visual SLAM setting. While filtering methods marginalize out past poses and summarize the information gained over time with a probability distribution, keyframe-based methods retain the optimization approach of global bundle adjustment by computationally selecting only a small number of past frames to process. To maintain a bounded-size optimization window for real-

time operation, Leutenegger et al. [30] improve keyframe-based visual-inertial SLAM by introducing keyframe marginalization to remove past states.

The concept of keyframes is also widely used in LiDAR mapping and a keyframe is often renamed as a submap to describe a sub-region of the complete map. Ni et al. [40] propose a submap-based approach to speed up smoothing and mapping (SAM) process when running SLAM in large environments. Hess et al. [23] introduce an indoor mapping algorithm that achieves real-time mapping and loop closure, and is known as the popular *Google Cartographer*. The method uses scan-to-submap matches as constraints, similar to how the keyframe concept is employed in a visual SLAM system.

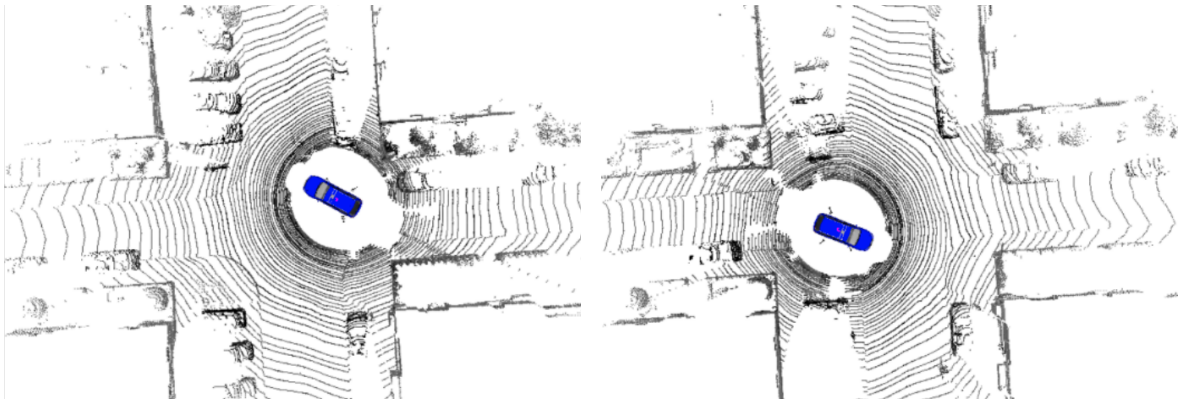
A common keyframe selection approach in LiDAR mapping is to select keyframes based on how far the robot has travelled using the odometry measurements. This ensures the robot to have a consistent window of keyframes over the trajectory.

2.2.1.4 Loop Closure

We introduce the concept of loop closure when a robot revisits a previously visited location in the map. In Figure 2.5(a), a vehicle visits an unmapped intersection and marks the current keyframe associated with this intersection. When the vehicle completes a loop in the environment and revisits the intersection from another viewpoint in Figure 2.5(b), the loop closure algorithm detects loop closure and matches the current scan to the earlier keyframe to form a loop closure constraint. Loop closure is an important component in building a globally consistent map because it adds this loop closure constraint into the pose graph for optimization. A loop closure example can be found in Figure 2.6.

Current loop closure detection approaches rely heavily on vehicle pose estimates to prompt loop closure. Over a short trajectory, compounded relative poses from the odometry solution do not drift much and can be used as a good source for loop closure detection. However, when the robot travels in an unknown environment over a long trajectory, pose estimates from the odometry solution become unreliable. Under such circumstances, the GNSS solution is sometimes a good candidate to globally localize the robot to help detect a loop closure, yet we require a more robust solution in GNSS-denied cases.

Scene-dependent visual appearances can be used to prompt loop closure. Ho et al.



(a) Vehicle at the intersection **(b)** Vehicle revisits the intersection

Figure 2.5: A vehicle visits an unmapped intersection and revisits from another viewpoint after completing a loop of the environment. Based on the geometry information in the LiDAR scans, the loop closure algorithm should inform the robot of a detected closed loop at the intersection despite viewpoint change. Image credit: Chen et al. [11].

[25] introduce a loop closure detection algorithm in a SLAM environment by combining visual and spatial appearances of local scenes. The method relies upon matching distinctive 'signatures' of individual local scenes to prompt loop closure and is completely independent of pose estimates in the mapping process. With a similar idea, Kim et al. [27] propose *Scan Context*, a non histogram-based global descriptor from a LiDAR scan, to describe a visited place using the structural information. The method computes a similarity score between two scan contexts in feature space and heuristically chooses a threshold to prompt loop closure. Furthermore, the use of structural information in this work ensures loop closure detection to be invariant to LiDAR viewpoint changes.

While classic approaches work well with loop closure detection, deep learning approaches address from a scene understanding perspective by learning the scene descriptors. The term *place recognition* is often used in the research community to recognize a visited place, similar to detecting a closed loop. Dubé et al. [20] extract segments from point clouds to form a data-driven descriptor. Correspondences are made between segments from the local and global maps by using k-NN retrieval in feature space which allows robust loop closure performance. Furthermore, Uy et al. [56] propose *PointNetVLAD* that learns global descriptors from point clouds for large-scale scenes. The method shows the feasibility of its network to the largely unexplored problem of point

cloud based retrieval for place recognition.

2.2.1.5 Pose Graph Optimization

A pose graph represents a robot's trajectory in the form of a graph, where each node is a pose and each directed edge is the relative pose from one node to another. Edges are weighted by the relative uncertainty in the robot's poses. The problem of pose graph optimization is to estimate a set of poses with respect to a fixed pose $\mathbf{T}_0$ from these relative pose measurements as illustrated in Figure 2.6. It is known to be a non-convex problem, and currently few techniques can guarantee the computation of a globally optimal solution.

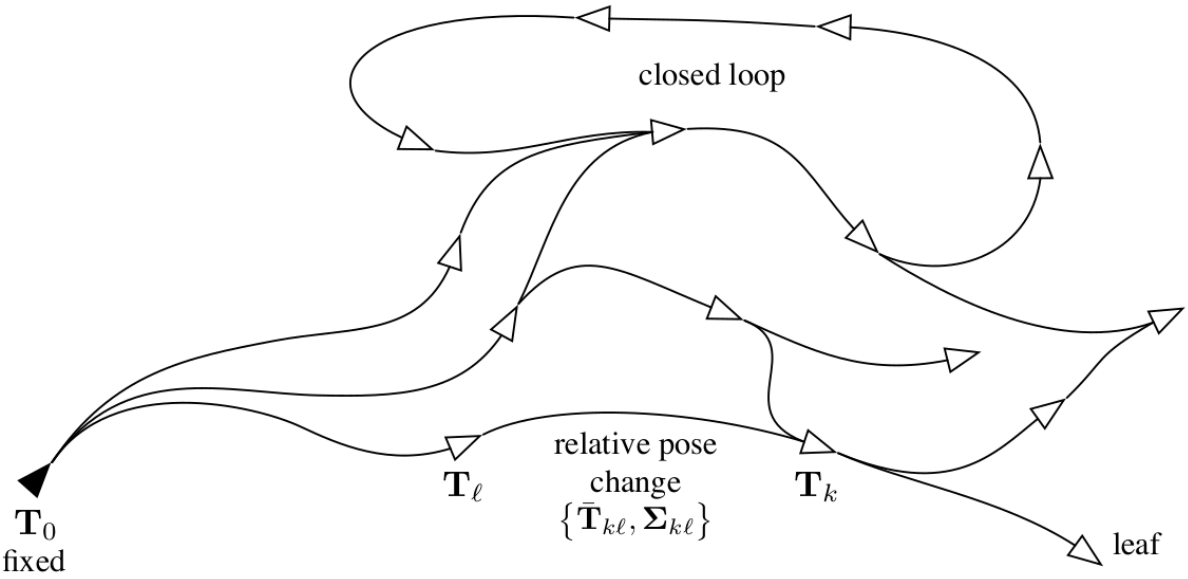


Figure 2.6: Pose graph with a closed loop. Each node is a pose and each directed edge is a relative pose measurement from one to another. The goal of this pose graph optimization is to estimate a set of poses with respect to a fixed pose $\mathbf{T}_0$. With the presence of a closed loop on top, we cannot simply compound the relative poses. Image credit: Barfoot et al. [4].

In the context of pose graph optimization for LiDAR mapping, each vertex represents a keyframe pose and each edge represents a measurement. As discussed in earlier sections, we have odometry and loop closure constraints added to the pose graph. Through pose graph optimization, we obtain optimized keyframe poses, which are essential to building a globally consistent map. Lu et al. [34] pioneer the formulation of pose graph optimization

in a robot mapping problem. The method achieves consistency using all the spatial relations as constraints to solve for the data frame poses simultaneously. Frese et al. [21] improve the performance by incorporating multi-level resolution optimization. The method proves its computation efficiency and performance on two real-world datasets. Furthermore, Olson et al. [41] address the convergence issue in pose graph optimization, especially with poor initial estimates. Through iterative optimization over only a subset of the information available in the problem, the method improves the convergence and rapidly recovers the robot trajectory even when given a poor initial estimate.

From the implementation perspective, g^2o and *gtsam* are the popular graph optimization libraries that support pose graph optimization. Kümmerle et al. [29] describe the general structure of graph optimization and present g^2o, an open-source C++ framework for optimizing graph-based nonlinear error functions. Similarly, Kaess et al. [26] present a novel data structure, Bayes tree, that encodes a factored probability density and maps naturally to the square root information matrix of the SLAM problem. This incremental method manages to achieve improvements in efficiency and results in the library *gtsam*.

2.2.2 Localization

Localization refers to localizing the robot within the map built from the mapping pipeline. It is a crucial capability for mobile robots and autonomous vehicles because it provides accurate pose estimates that are critical to downstream tasks, e.g., obstacle avoidance. In this section, we introduce existing literature addressing the LiDAR localization problem.

2.2.2.1 Localization Initialization

Given a map built from the mapping pipeline, localization initialization is crucial for the localization system because it provides an initial pose for the robot to start localizing itself within the map.

A simple approach to initialize localization is to use the pose estimate provided by a GNSS device [31, 60, 5, 59]. However, when the robot is in an area with poor GNSS coverage, it becomes crucial to have an alternative approach to initialize the localization

system.

Recent literature addresses localization initialization from a scene understanding perspective similar to loop closure detection methods. Existing approaches match current scans to the map in feature space to retrieve coarse pose estimates. Kim et al. [27] compress point clouds to scan context images and train a CNN based on these images. The method matches the scan context image for current scan to the one for every cell in the map and estimates current location by computing a correlation score between scan context images. Yin et al. [61] introduce a Siamese network to learn fingerprints of local scenes for place recognition. The method then obtains precise localization using a particle filter algorithm. Furthermore, Komorowski et al. [28] present an approach to compress point clouds into global descriptors for place recognition purpose. Unlike *PointNetVLAD*, this method captures both global and local geometric features and outperforms the state-of-the-art.

2.2.2.2 Precise Localization

The localization system obtains prior knowledge of the robot's initial pose in the map after localization initialization. Precise localization uses prior knowledge to achieve accurate pose estimation for robots within the map. In this section, we introduce existing literature on robots to localize themselves within a given map provided some prior knowledge of its pose. It is worth noting that many existing approaches either generate their own map representations as discussed in the earlier section or rely on a third-party company to provide accurate high-definition maps. Following, we carefully place existing approaches into two categories: classic approach and deep learning approach.

Filter-based and ICP-based methods are the most common classic approaches to address the precise localization problem. Levinson et al. [31] propose an approach to utilize GPS, IMU, wheel odometry and LiDAR to build a high-resolution map, followed by a particle filter algorithm to correlate LiDAR measurements against the map for accurate localization. With a similar idea, Wan et al. [58] use Bird Eye View (BEV) LiDAR intensity images, coupled with a differential GPS and an IMU, to robustly localize against a pre-generated map to achieve centimetre-level accuracy. An error-state EKF

is also employed to fuse measurements from different sensor modalities for uncertainty estimation. In contrast, Yoneda et al. [62] extract distinct LiDAR features and align the online LiDAR scan to the highly precise 3D map using ICP. However, as discussed in an earlier section, ICP is known to be susceptible to poor initial guesses.

With the rising popularity of deep learning, recent literature attempts to retrieve precise localization information through data-driven methods. Barsan et al. [5] propose a real-time calibration agnostic approach that learns deep embedding to align online LiDAR scan to a LiDAR intensity map that has been pre-processed for dynamic object removal. Although the method achieves centimetre-level accuracy, its performance highly relies on a good GPS prior. Using a similar idea, Wei et al. [59] encode LiDAR intensity images into binarized form to achieve scan-to-map alignment. The method emphasizes its contribution to the high map compression rate without loss of localization performance. Furthermore, Lu et al. [35] introduce L^3-Net to learn keypoints and their representations in an unordered point cloud for scan-to-map alignment. The method creates a cost volume for every keypoint in the pre-generated map using the initial pose estimate and conducts keypoint matching using learned representations. Although it achieves strong localization performance on real-world datasets, it cannot run in real-time due to expensive computations.

In this thesis, our mapping and localization solution provides us with accurate pose estimates for robots within a globally consistent map. We rely on the map along with these pose estimates to accomplish our automated annotation pipeline.

2.3 Scene Understanding

Scene understanding refers to analyzing and elaborating an interpretation of a 3D dynamic scene observed through a network of sensors. It provides the capability to further enable autonomous navigation. In this thesis, we focus on scene understanding through LiDAR point clouds and we introduce existing literature in this section. We carefully place existing scene understanding works into two research streams: semantic segmentation for point clouds and dynamic object detection in point clouds. While the former

classifies semantics into more classes, the latter focuses on distinguishing only dynamic points in a scene.

2.3.1 Semantic Segmentation

The goal of semantic segmentation for point clouds is to assign every point in a point cloud a semantic label of what it is represented. Common semantic labels in a self-driving scenario include vegetation, building, person etc. In the current literature, researchers often address point cloud semantic segmentation through supervised deep learning approaches, which require groundtruth semantic labels from human annotations. A good example of a hand-annotated semantic segmentation dataset for self-driving is *SemanticKITTI* [6], which consists of 28 semantic classes distinguishing moving and non-moving objects as shown in Figure 2.7.

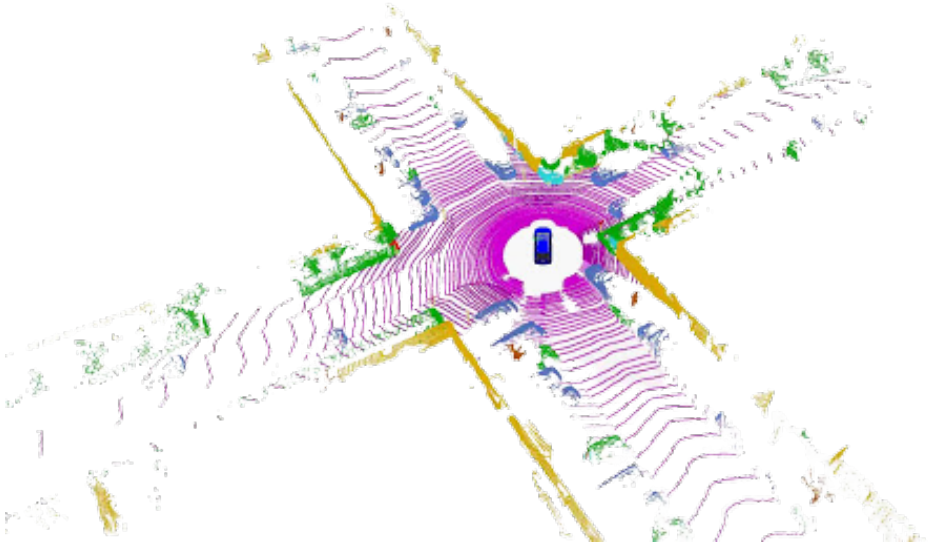


Figure 2.7: Ground truth semantic segmentation for a point cloud in *SemanticKITTI* [6]. The dataset hand-annotates point clouds into classes including ground, structure, vehicle, nature, human, object and outlier. It is commonly used as a benchmark for point cloud semantic segmentation tasks. Image credit: Behley et al. [6].

In this section, we introduce existing literature on supervised semantic segmentation for point clouds. Riegler et al. [46] introduce *OctNet* to learn deep 3D representations at high resolutions for point clouds. The method projects a point cloud onto an intermediate

grid structure and uses every leaf node in an octo-tree to store a feature representation. However, the fallback of this approach is its limited flexibility due to the fixed grid size. To avoid the grid structure, Qi et al. [44] pioneer pointwise operation in point cloud learning and uses a shared Multi-Layer Perceptron (MLP) on every point followed by a max pooling operation. The method validates its effectiveness on both classification and semantic segmentation tasks. However, despite its great innovation in point cloud learning, it fails to encode local geometric structure information into its feature representation. To extend *PointNet*, Qi et al. [45] further introduce *PointNet++* to capture local geometric structure induced by the metric space points live in. This method improves the generalizability of semantic segmentation, especially in more complex scenes.

In addition to pointwise operations on point clouds, some recent literature defines convolutions for points explicitly. Thomas et al. [54] define kernel point convolution for point clouds and introduce KPConv that operates on point clouds without any intermediate representations. The convolutional weights lie in Euclidean space and apply to points that are close to them. The locations of these kernel points are also learnable through the network and thus they can learn to adapt to the local geometry of the point cloud. As a result, this method outperforms state-of-the-art semantic segmentation approaches in multiple datasets.

In this thesis, we are interested in existing semantic segmentation networks for point clouds because we want to use automated annotations as training data to learn the semantic information from a point cloud. To satisfy our need, we make the design choice to build our network based on the existing semantic segmentation network, KPConv [54]. We further elaborate on our implementation details in Chapter 3.

2.3.2 Dynamic Object Detection

Compared to semantic segmentation, dynamic object detection focuses on detecting only dynamic points in a point cloud. In this section, we describe the existing literature.

Recent literature detects dynamic points by exploiting temporal information from a window of LiDAR scans. Yoon et al. [63] propose a model-free approach for online detection of dynamic points in a point cloud. The method explicitly compensates for

motion distortion and compares the latest LiDAR scan to a reference scan for detection. However, the fallback of this method is that it captures only currently moving objects as opposed to all the movable objects. A good example is a parked vehicle, which has the ability to move to another location yet does not move at the capture time. Including points from a movable object in mapping and localization can affect the scan matching performance because these points are not consistently observed.

In contrast, Dewan et al. [17] identify points into three categories: non-movable, movable, and dynamic, and introduce *Fast-Net* to learn a pointwise *objectness* score, where a high score corresponds to the moving class. Coupled with a *dynamicity* score computed using odometry estimates, the method uses a Bayes filter framework for estimating the pointwise semantic classification. Although this method takes into account movable points, the network training requires hand-labelled groundtruth. Our approach avoids this expensive operation via an automated annotation pipeline. Furthermore, our semantic segmentation network makes predictions using the latest LiDAR scan yet this method relies on accurate pose estimates between consecutive frames to compute the *dynamicity* score.

2.3.3 Semantic SLAM

Motivated by the advances of deep learning in scene understanding, there have been many semantic SLAM techniques that exploit semantic information in order to improve SLAM performance.

Dubé et al. [20] introduces *SegMap* which is based on extracting segments from point clouds to form data-driven descriptors. The method learns a simple fully connected network to extract semantic information from descriptors. Using the semantic information, it performs LiDAR localization only against static objects to add robustness against any dynamic changes. In contrast, our approach not only learns semantic information directly from the unordered sets of point clouds as opposed to voxel grid representation but also replaces hand-labelling with an automated annotation pipeline.

In *Suma++*, Chen et al. [12] improve over *SegMap* by integrating semantic information provided by a semantic segmentation network into a surfel-based Mapping pipeline

[7]. This method focuses on generating a rich semantic map with an abundance of semantic classes. To improve both mapping and odometry accuracy, it filters out points that belong to a movable class, such as moving vehicles and pedestrians. Despite performance improvement, *Suma++* still fails to address the expensive operation to retrieve hand-labelled groundtruth for semantic segmentation.

Eventually, we emphasize that the foundation of our approach is the self-supervised segmentation pipeline developed by Thomas et al. [53]. Thomas proposes an automated annotation pipeline for LiDAR point clouds using pose estimates from a multi-session SLAM solution. A point cloud segmentation network is then trained to predict semantic labels using annotations as the groundtruth labels. The method obtains annotations without any hand-labelling and shows its performance improvement from session to session. Despite great benefits in this work, this method is only evaluated in an unchanged simple indoor environment with limited complexities for scene understanding. In this thesis, we improve over the existing pipeline and further extend it to a large-scale environment. Following Chapter 3, we elaborate on our proposed pipeline.

2.4 Summary

In this chapter, we introduced the background and existing literature related to our thesis. We provide in-depth descriptions of existing works on LiDAR mapping and localization along with scene understanding. Finally, we looked at existing semantic SLAM methods that incorporate semantic information to achieve improvements in mapping and odometry accuracy.

Chapter 3

Methodology

This chapter introduces our self-supervised semantic learning pipeline for large-scale scenes and describes the methodology of each component. Our proposed pipeline can be summarized into two steps each comprising multiple components. Step one is to annotate LiDAR scans using an automated approach and step two is to use annotations from step one to train a semantic segmentation network.

From now on, we explicitly define a *session* to be a traversal through an environment. In a mapping and localization setting, it usually involves multiple sessions in the same environment.

In the following sections, we begin by presenting a mapping and localization pipeline that produces a globally consistent map along with associated localization trajectories. We then apply ray-tracing on the map using all localization scans to compute point-level moving probabilities. Next, we map point-level moving probability to a heuristically determined annotation and finally learn a semantic segmentation network from the annotations.

We explicitly state that our proposed pipeline builds on the work by Thomas et al. [53] for autonomous indoor navigation. We extend the approach to large-scale environments, especially autonomous driving scenarios. The main extensions of our approach are

- utilizing a robust LiDAR-only mapping and localization solution provided by Applanix Corporation for large-scale autonomous driving scenarios as opposed to stan-

dard ICP-based alignment

- correcting motion distortion for LiDAR point clouds when aggregating scans to map
- designing a voting strategy to extract ground points for large-scale point cloud map
- accounting for move-and-stop nature of on-road vehicles at traffic intersections
- labelling long-term movable points without carefully crafted geometric constraints

3.1 Pipeline Overview

In this section, we define components in our self-supervised semantic learning pipeline for large-scale scenes. A pipeline diagram is shown in Figure 3.1.

Our pipeline requires a classic mapping and localization setup, in which a vehicle traverses through an environment, collecting multiple sessions of data. Usually, one session is used for mapping and other sessions are used for localization. Following we provide a short summary of every block and operation in the pipeline shown in Figure 3.1.

1. *Mapping/Localization scans*: Scans from a mapping/localization session.
2. *Mapping*: Align scans from a mapping session to produce a globally consistent map.
3. *Localization*: Localize scans from a localization session against map built from a mapping session.
4. *Mapping/Localization trajectory*: Estimated vehicle trajectory for a mapping/localization session.
5. $\mathcal{M}^{(m)}/\mathcal{M}^{(l)}$: Aggregated point cloud map for a mapping/localization session.
6. *PointRay*: A ray-tracing technique that uses mapping and localization scans with both $\mathcal{M}^{(m)}$ and $\mathcal{M}^{(l)}$. We provide more details in Chapter 3.3.1.

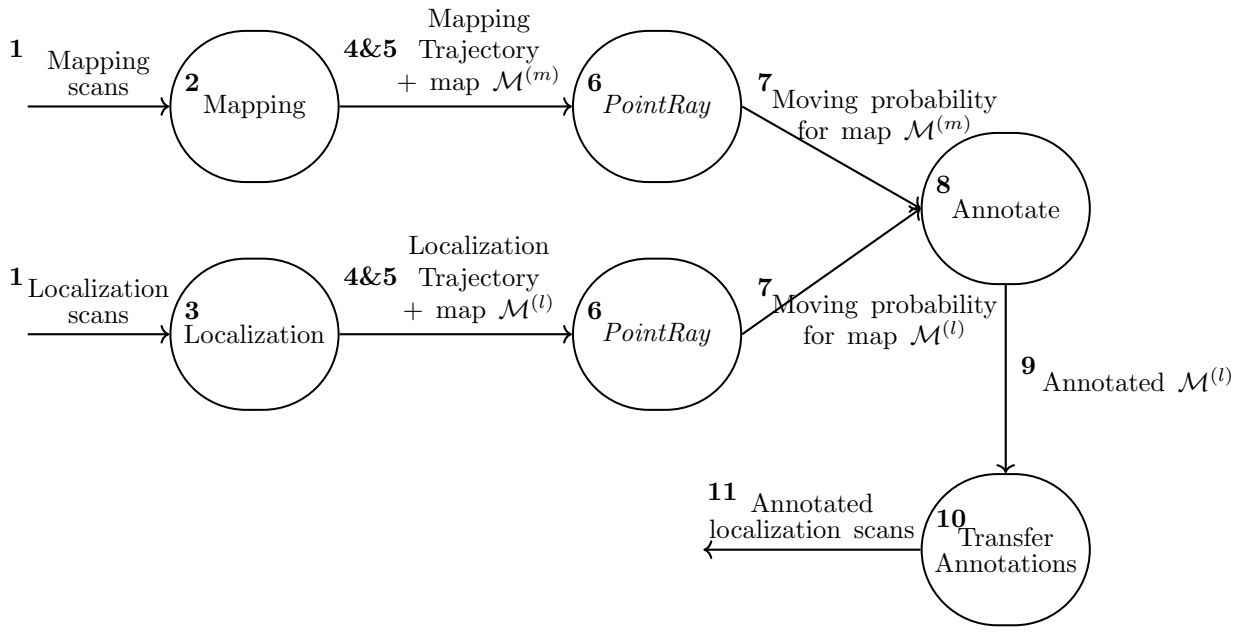


Figure 3.1: Self-supervised semantic learning pipeline for large-scale scene

7. *Moving probability*: Probability that a point belongs to a moving object.

8. *Annotate*: Assign a semantic label to point in a point cloud map.

9. *Annotated map*: A point cloud map in which every point is assigned a semantic label.

10. *Transfer annotations*: Transfer point labels from a point cloud map to points in associated scans.

11. *Annotated scans*: Scans in which every point is assigned a semantic label.

3.2 Mapping and Localization

In Chapter 2, we introduced existing LiDAR mapping and localization methods. While some existing literature focuses on localization within a pre-generated high-definition map obtained through a third-party provider, our proposed pipeline obtains a globally consistent map directly from LiDAR scans. Since an accurate mapping and localization

solution is an important factor in our pipeline, we make our design choice to use an existing LiDAR-only mapping and localization solution provided by Applanix Corporation. Given multiple data sessions in an unknown environment, we follow the convention to use one session as a mapping session and the rest as localization sessions.

3.2.1 Applanix Mapping and Localization Solution

The goal of mapping is to create a globally consistent map for an environment that would later be used for localization. This becomes really challenging for large-scale environments, especially self-driving scenarios that comprise many dynamic elements. To our knowledge, Applanix Corporation has been a top contender in the industry field to provide a robust mapping and localization solution, which perfectly suits our needs.

Here we elaborate our mapping and localization pipeline provided by Applanix even though we cannot provide algorithm details due to the non-disclosure agreement. We define point cloud $P_k^{(m)}$ captured in LiDAR base frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ at timestamp $t_k^{(m)}$, where a superscript m represents a quantity from the mapping session. Using a time series of N point cloud input, $P_1^{(m)}, P_2^{(m)}, ..., P_N^{(m)}$, the mapping pipeline estimates poses $\mathbf{T}^{(m)} = \{\mathbf{T}_1^{(m)}, \mathbf{T}_2^{(m)}, ..., \mathbf{T}_N^{(m)}\}$, in which each entry $\mathbf{T}_k^{(m)}$ represents pose of $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ at time $t_k^{(m)}$ with respect to the inertial frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_i$. We emphasize that in our thesis the inertial frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_i$ is designed to coincide with the initial $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ at time $t_1^{(m)}$. In addition to the estimated poses $\mathbf{T}^{(m)}$, the mapping pipeline produces a globally consistent map $\mathcal{M}^{(m)}$ which we would use to localize our robot.

The goal of localization is to localize the current scans to a globally consistent map we obtain from the mapping session. Again, we rely on Applanix Corporation's mapping and localization solution to provide accurate localization outputs for our multiple localization sessions.

We demonstrate the localization pipeline to localize scans from one session against map $\mathcal{M}^{(m)}$. We repeat this process to retrieve pose estimates for multiple localization sessions. We define point cloud $P_k^{(l_j)}$ captured in LiDAR base frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ at timestamp $t_k^{(l_j)}$, where a superscript l_j represents a quantity from the j^{th} localization session. We abuse the notation and assume N point cloud input from the localization session,

$P_1^{(l_j)}, P_2^{(l_j)}, ..., P_N^{(l_j)}$. Provided the map $\mathcal{M}^{(m)}$ from mapping session, the localization estimates poses $\mathbf{T}^{(l_j)} = \{\mathbf{T}_1^{(l_j)}, \mathbf{T}_2^{(l_j)}, ..., \mathbf{T}_N^{(l_j)}\}$, pose of $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ at time $t_k^{(l_j)}$ with respect to the inertial frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_i$ in map $\mathcal{M}^{(m)}$.

Using the described mapping and localization pipeline provided by Applanix Corporation, we are able to retrieve a globally consistent map $\mathcal{M}^{(m)}$ along with accurate pose estimates $\mathbf{T}^{(m)}$ and $\mathbf{T}^{(l)}$ for both mapping and localization sessions. We remove the subscript j from $\mathbf{T}^{(l_j)}$ so that $\mathbf{T}^{(l)}$ represents a collection of pose estimates for multiple localization sessions. To use $\mathcal{M}^{(m)}$ in our annotation pipeline, we require $\mathcal{M}^{(m)}$ to retain a fine resolution. However, $\mathcal{M}^{(m)}$ is significantly down-sampled in the mapping and localization pipeline to satisfy the computation requirement, and thus can no longer be used. Therefore, we choose to re-generate $\mathcal{M}^{(m)}$ by aggregating LiDAR scans $P^{(m)} = \{P_1^{(m)}, P_2^{(m)}, ..., P_N^{(m)}\}$, followed by a grid down-sampling operation.

Prior to aggregating LiDAR scans, we need to correct the motion distortion. Previously, we have defined point cloud $P_k^{(m)}$ to be the k^{th} scan captured at time $t_k^{(m)}$. We further let point i in $P_k^{(m)}$ where its pointwise timestamp is bounded by $t_k^{(m)} <= t_{k_i}^{(m)} < t_{k+1}^{(m)}$, as illustrated in Figure 3.2. The goal of motion compensation is to retrieve pose estimates $\mathbf{T}_{k_i}^{(m)}$, which can be used to transform point i to $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ at time t_k for each LiDAR scan. The resulting scan at time t_k is motion-compensated and we define it to be $\bar{P}_k^{(m)}$. We explicitly state that motion-compensated scans $\bar{P}_k^{(m)}$ can be obtained from Applanix mapping and localization solution.

There are other alternatives to do mapping and localization other than the solution provided by Applanix. We did not employ *PointMap* from Thomas et al. [53] to address this due to its limitation in large-scale scenes. Compared to [53], the mapping and localization solution from Applanix Corporation takes into account motion compensation, works more robustly towards outliers, and provides high computational efficiency.

3.2.2 Map Aggregation

To aggregate motion compensated scans to a globally consistent map $\mathcal{M}^{(m)}$, we transform LiDAR scans $\bar{P}^{(m)} = \{\bar{P}_1^{(m)}, \bar{P}_2^{(m)}, ..., \bar{P}_N^{(m)}\}$ to the inertial frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_i$, followed by a grid

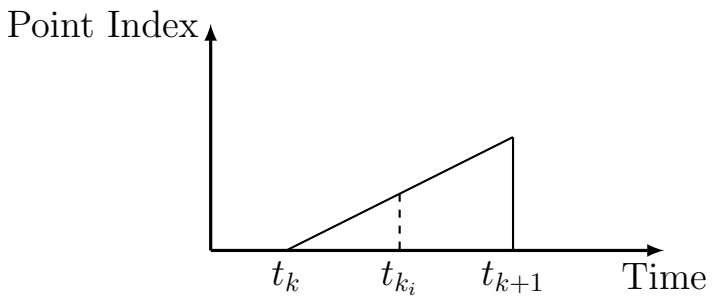


Figure 3.2: Pointwise timestamp information for all points in P_k spanning from t_k to t_{k+1}

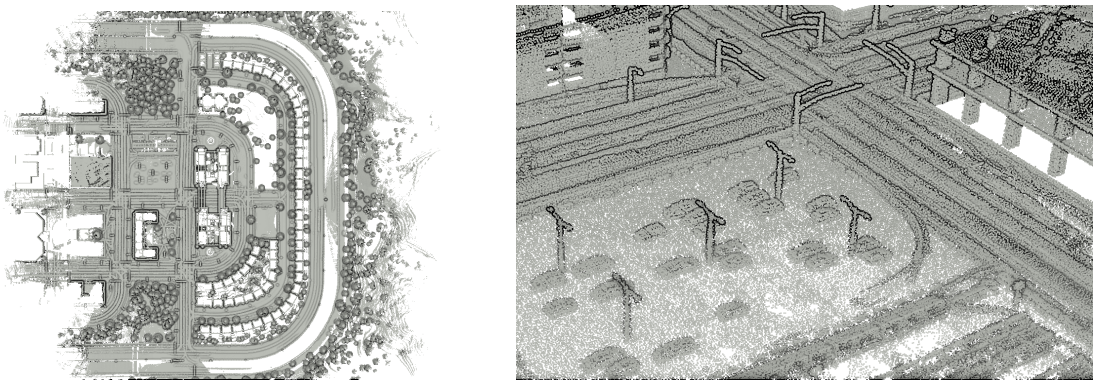
down-subsampling operation, denoted as $g(\cdot)$,

$$\mathcal{M}^{(m)} = g(\sum_{k=0}^{N} \mathbf{T}_k^{(m)} \bar{P}_k^{(m)}) \tag{3.1}$$

For grid down-sampling, we use the first added point in a grid cell to represent a grid instead of using the barycenter. We explicitly choose this strategy to avoid drift in space when more LiDAR scans are added to the map. To retain a fine resolution for our automated annotation pipeline, we heuristically choose a grid size of 0.3 m. This grid size keeps the map compact while not losing much detail. Eventually, we obtain a list of voxels after grid down-sampling and there is exactly one point residing in each voxel. Figure 3.3(a) shows an example of a grid down-sampled map. When we take a closer look at the map in Figure 3.3(b), we can easily observe the fine details and ignore the usual artifacts coming from a grid representation.

Using the exact map aggregation method, we also aggregate LiDAR scans from each localization session in order to produce $\mathcal{M}^{(l_j)}$. The benefit of using pose estimates after localizing against a globally consistent map $\mathcal{M}^{(m)}$ is that we can now easily overlay $\mathcal{M}^{(l_j)}$ on top of $\mathcal{M}^{(m)}$ without worrying about any misalignment issue. To simplify our notation, we use $\mathcal{M}^{(l)}$ to represent a collection of point cloud maps $\mathcal{M}^{(l_j)}$ for all localization sessions.

Finally, we conclude our mapping and localization pipeline by mentioning that it not only provides us with accurate pose estimates $\mathbf{T}^{(m)}$ and $\mathbf{T}^{(l)}$, but also allows us to retrieve aggregated and grid down-sampled maps $\mathcal{M}^{(m)}$ and $\mathcal{M}^{(l)}$. More importantly, it empowers our annotation pipeline which we introduce in the next section.



(a) Grid down-sampled map **(b)** Sub-region in grid down-sampled map

Figure 3.3: **(a)** After map aggregation, we apply grid down-sampling to retrieve a grid representation of our point cloud map. A grid size of 0.3 m allows us to keep the map compact while not losing many details. **(b)** When we zoom into the grid down-sampled map, we can observe traces of the vehicles and clear contours of the buildings.

3.3 Annotation Pipeline

We build our annotation pipeline on the foundation of Thomas et al. [53] to annotate LiDAR scans from localization sessions with four semantic classes: ground, non-movable, long-term movable, and short-term movable. We choose these semantic classes to represent dynamics of the points. As the name suggests, ground refers to points belonging to the ground. In a large-scale self-driving environment, ground points tend to have a lower elevation than the rest of the point cloud and can be easily retrieved. Non-movable includes points in a point cloud that do not have the capability of moving around. Permanent structures such as buildings belong to this category. Scan-to-scan odometry and scan-to-map alignment intuitively achieve better accuracy if there is an abundance of permanent structures in the current scan. Another category is long-term movable. Points belonging to this category do not move in the current session but can be relocated in another session. For example, a vehicle may park at one location in the mapping session, but at a different location in a localization session. The existence of these points affects the localization performance because they are not consistently observed in the map. The last class is the short-term movable that refers to points that are currently moving in the session. For a large-scale self-driving environment, most of these points belong to the on-road vehicles and pedestrians on the sidewalk.

In this section, we use pose estimates $\mathbf{T}^{(m)}$ and $\mathbf{T}^{(l)}$ along with map $\mathcal{M}^{(m)}$ and $\mathcal{M}^{(l)}$ from the mapping and localization pipeline to automate the annotation pipeline. We emphasize that our goal is to annotate LiDAR scans from all localization sessions. We begin with introducing *PointRay* as a ray-tracing strategy [53] to compute moving probability for points in a map. Next, we utilize the moving probability to annotate the map and transfer the annotations to LiDAR scans. Finally, we use automated annotations to train a semantic segmentation network for scene understanding offline. To extend the original idea [53] to large-scale autonomous driving scenarios, we employ a voting strategy to extract ground points for large-scale point cloud maps and account for the move-and-stop nature of on-road vehicles at traffic intersections. Furthermore, our labelling of long-term movable points does not require carefully crafted geometric constraints compared to the original approach.

3.3.1 *PointRay*

Recall that a laser beam in a LiDAR sensor sends out pulsed light waves and returns when it hits an obstacle. This is the foundation of ray-tracing. We use ray-tracing to identify occupied and unoccupied space in a point cloud map in order to further compute the moving probability. In this section, we provide more details about *PointRay* from Thomas et al. [53] and how we utilize it to address our problem.

We begin by introducing the notion of freespace, which is the path that a laser ray travels. Next, we describe the frustum grid we use to encode the freespace. Then we define *the moving probability* for points in a point cloud, and eventually, combine the knowledge to present our complete approach to compute and update moving probability.

3.3.1.1 Free Space

The paths that laser beams emit pulsed light waves define freespace. This aspect of LiDAR data is often ignored as most robotics applications focus on processing endpoints, i.e., point cloud data. However, freespace also provides us with useful information.

For better demonstration, Figure 3.4 shows an example of a 2D LiDAR sensor emitting

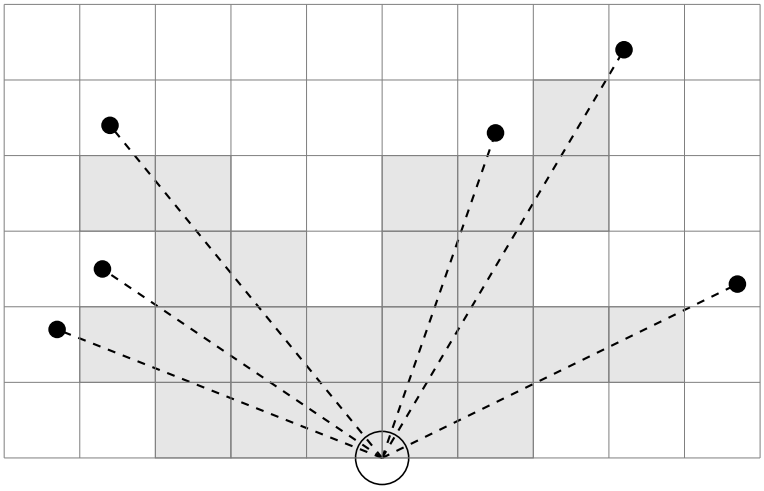


Figure 3.4: Top view of a 2D LiDAR sensor (bottom middle) emitting light waves and receiving returns when hit on obstacles. The dots (black) represent point returns whereas the dashed lines (black) stand for the paths light rays travel. In a grid representation of this scan, gray grids represent the freespace.

light waves and receiving point returns. The dashed lines indicate light rays from the LiDAR. If we choose to divide the scan into grids as how we apply grid down-sampling in Chapter 3.2, the grids in gray represent the freespace of this scan.

Freespace allows us to learn more about the unoccupied space in the environment. When a light ray travels and defines the freespace, it tells us that its freespace is less likely to contain any permanent structures, e.g., buildings. Because otherwise, the light ray is unable to go through. In other words, if a point is found in the freespace of another point, the former point more likely belongs to a moving object. We further use this knowledge about freespace to help us define moving probability. From now on, we define the freespace of a scan to be the joint freespace of all points from this scan.

3.3.1.2 Frustum Grid

Prior to defining moving probability for our automated annotation pipeline, we describe our method to encode freespace for a LiDAR scan.

We employ the common method to discretize the world using spherical coordinates into frustum grids, also known as 2D grids along θ and ϕ dimension. Each pixel in the frustum grid stores the minimum point distance to the origin of LiDAR base frame

$\underrightarrow{\mathcal{F}}_b$. Figure 3.5 shows an example of a frustum grid representation in a self-driving environment. We can easily observe that this representation resembles a LiDAR range image in Figure 2.2(b). However, rather than keeping its row dimension as the number of available laser beams, frustum grids determine it to be parametric to the vertical resolution $d\phi$.

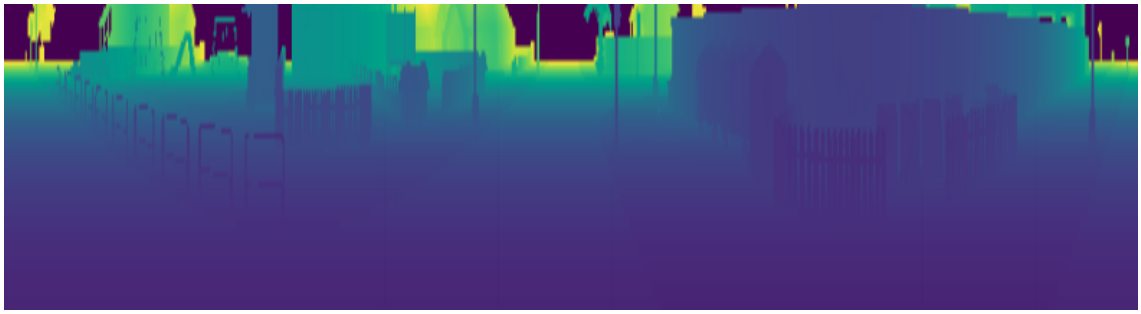


Figure 3.5: Frustum grid representation of freespace for a 3D point cloud. Each pixel contains the minimum point distance to the sensor origin from a spherical angle. Points further away are coloured in yellow and points closer are coloured in blue.

The resolution of a frustum grid representation is defined by the parameter $d\theta$ and $d\phi$. We use $d\theta = 1.2°$ and $d\phi = 0.1°$ in our thesis to encode freespace. Therefore, our frustum grid representation has $\frac{360°}{1.2°} = 300$ columns.

Furthermore, we emphasize that one of the benefits of using a frustum grid representation is its efficient spatial query, which is an O(1) operation. To check if a point belongs to the freespace, we convert the point into frustum grid representation and compare its range value to the one stored in the corresponding frustum grid cell. The point belongs to the freespace if it has a smaller range value.

3.3.1.3 Moving Probability

Moving probability refers to the probability that a point belongs to a moving object. It is an important measure in our annotation pipeline because we carefully design heuristics around it to annotate LiDAR scans. In our pipeline, we employ *PointRay* in [53] to compute moving probability for points in the aforementioned $\mathcal{M}^{(m)}$ and $\mathcal{M}^{(l)}$ using associated motion-compensated scans $\bar{P}^{(m)}$ and $\bar{P}^{(l)}$ and pose estimates $\mathbf{T}^{(m)}$ and $\mathbf{T}^{(l)}$.

PointRay consumes a sequence of LiDAR scans and associated poses to compute

moving probability for all the points in an arbitrary map $\mathcal{M}$. We carefully identify quantities to describe *PointRay*,

1. x_i: a point in map $\mathcal{M}$.
2. $\bar{P}_k$: a motion compensated scan at time t_k that *PointRay* consumes to apply ray-tracing on map $\mathcal{M}$.
3. $\mathbf{T}_k$: pose estimate of scan $\bar{P}_k$ at time t_k with respect to the inertial frame in map $\mathcal{M}$.
4. n_i: number of times x_i is observed in any LiDAR scans.
5. m_i: number of times x_i is observed in the freespace of any LiDAR scans.

From now on, we use mathematical notations to further describe *PointRay*. For every point x_i in $\mathcal{M}$, *PointRay* assigns n_i and m_i to help quantify how frequently x_i is observed in the freespace of any scan. While n_i counts the number of times a point is observed in any scans, m_i counts the number of times a point is observed in the freespace of any scans. To begin *PointRay*, we initialize n_i and m_i to be all zeros.

Next, for every scan $\bar{P}_k$, we encode its freespace and update n_i and m_i for point x_i by checking whether x_i belongs to the freespace of $\bar{P}_k$. For any point $\bar{P}_{k_i}$ in point cloud $\bar{P}_k$, we obtain its occupied voxel in our grid down-sampled map $\mathcal{M}$ using $\mathbf{T}_k$ and the known grid resolution of 0.3 m

$$vox(\bar{P}_{k_i}) = \frac{\mathbf{T}_k \bar{P}_{k_i}}{0.3} \in \mathbb{Z}^3 \tag{3.2}$$

where $\mathbb{Z}^3$ represents the voxel grid coordinate system.

Then, we assume $vox(\bar{P}_{k_i})$ happens to be the i^{th} voxel in map $\mathcal{M}$ in which x_i resides. This means $\bar{P}_{k_i}$ in scan $\bar{P}_k$ shares the same voxel as x_i in map $\mathcal{M}$ after scan-to-map alignment using $\mathbf{T}_k$. We repeat this process for all points in scan $\bar{P}_k$ to retrieve a list of voxels in map $\mathcal{M}$.

For all x_i residing in the list of voxels in the map $\mathcal{M}$, we increment n_i by 1. This incremental procedure ensures that we accurately count the number of times a point x_i is

observed in the occupied space in map $\mathcal{M}$ when traversing through all the scans. Next, we encode the freespace of $\bar{P}_k$ into a frustum grid representation $\mathcal{I}_k$ to further update n_i and m_i. A detailed encoding process is described in Chapter 3.3.1.2. To check whether a point x_i belongs to the freespace of $\bar{P}_k$, map $\mathcal{M}$ is also projected into the same frustum grid. We carefully update n_i again by checking the following condition

$$n_i = \begin{cases} n_i + 1 & \text{if } r(x_i) <= \mathcal{I}_k(\theta_i, \phi_i) \\ n_i & \text{if } r(x_i) > \mathcal{I}_k(\theta_i, \phi_i) \end{cases}. \tag{3.3}$$

Similarly, we update m_i by

$$m_i = \begin{cases} m_i + 1 & \text{if } r(x_i) <= \mathcal{I}_k(\theta_i, \phi_i) \\ m_i & \text{if } r(x_i) > \mathcal{I}_k(\theta_i, \phi_i) \end{cases}, \tag{3.4}$$

where $r(\cdot)$ defines the distance of x_i to the origin of $\bar{P}_k$. When x_i is projected onto the frustum grid $\mathcal{I}_k$, $\mathcal{I}_k(\theta_i, \phi_i)$ refers to the stored minimum point distance along the spherical direction θ_i and ϕ_i. This incremental policy suggests that whenever x_i is in the freespace of a scan, we increment both n_i and m_i by 1.

Finally, after we traverse through all the scans and update n_i and m_i for all x_i in map $\mathcal{M}$, we proceed to compute the moving probability p_i for point x_i in map $\mathcal{M}$

$$p_i = \frac{m_i}{n_i}, \tag{3.5}$$

where $0 <= p_i <= 1$ because m_i is strictly smaller than n_i following our update policy. Also, we can easily observe that moving probability p_i is the probability that a point x_i is observed in the freespace of any scans.

To further use moving probability for the automated annotation pipeline, we save p_i to the existing $\mathcal{M}$ by adding a moving probability entry to each x_i in addition to x, y and z coordinates. Figure 3.6 shows an example moving probability of $\mathcal{M}^{(m)}$ after applying *PointRay* using scans from the same mapping session. It is easy to notice that points belonging to vehicle traces (red) tend to receive a much higher moving probability as

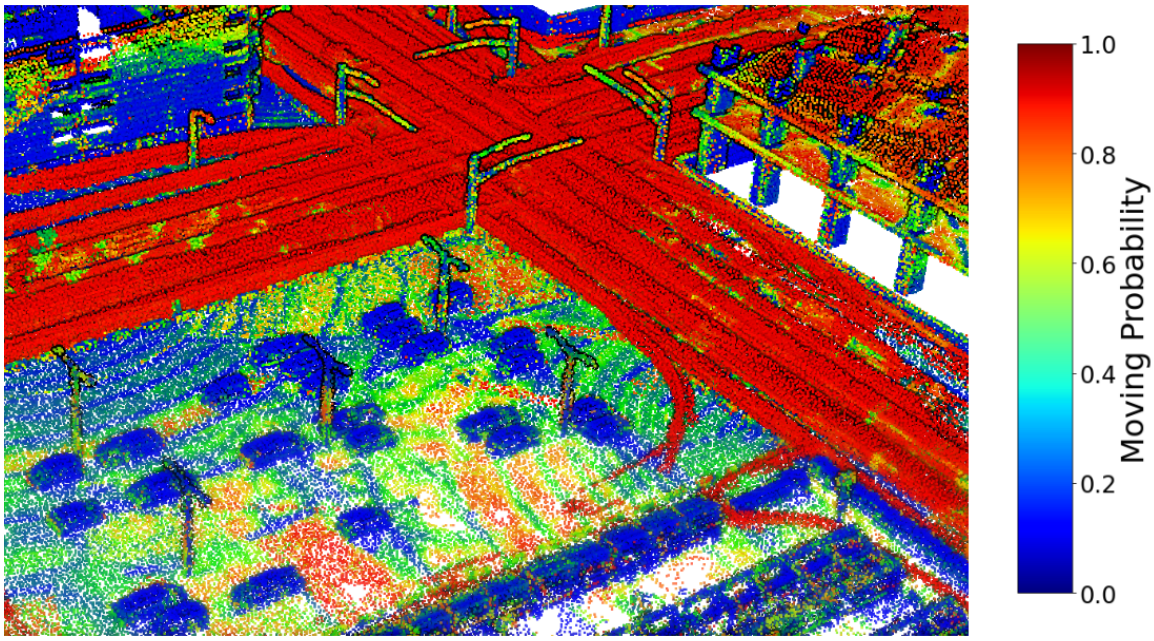


Figure 3.6: An example of moving probability for a sub-region in map $\mathcal{M}$, where the colour depicts the probability that a point belongs to a moving object. Points in red have a high moving probability whereas points in blue have a low probability.

opposed to points belonging to permanent structures (blue). This is because vehicles are observed moving in the scene, and thus points belonging to vehicle traces frequently fall into the freespace of LiDAR scans, resulting in a high moving probability. On the other hand, permanent structures such as buildings do not move and therefore receive a lower moving probability. To further illustrate our complete process to compute moving probability, we provide pseudo-code in Algorithm 1 in the Appendix.

3.3.1.4 *PointRay* Comparison

PointRay computes moving probability for map $\mathcal{M}$ using scans from one session. However, moving probability p_i for point i in map $\mathcal{M}$ differs when applying *PointRay* using scans from another session. This is because different sessions define different freespace, and thus resulting in different moving probabilities.

To illustrate the difference, we first apply *PointRay* on map $\mathcal{M}$ using scans from session A, the same session that is used to aggregate and down-sample map $\mathcal{M}$. Then we apply *PointRay* again on map $\mathcal{M}$ but using scans from session B. Note that these

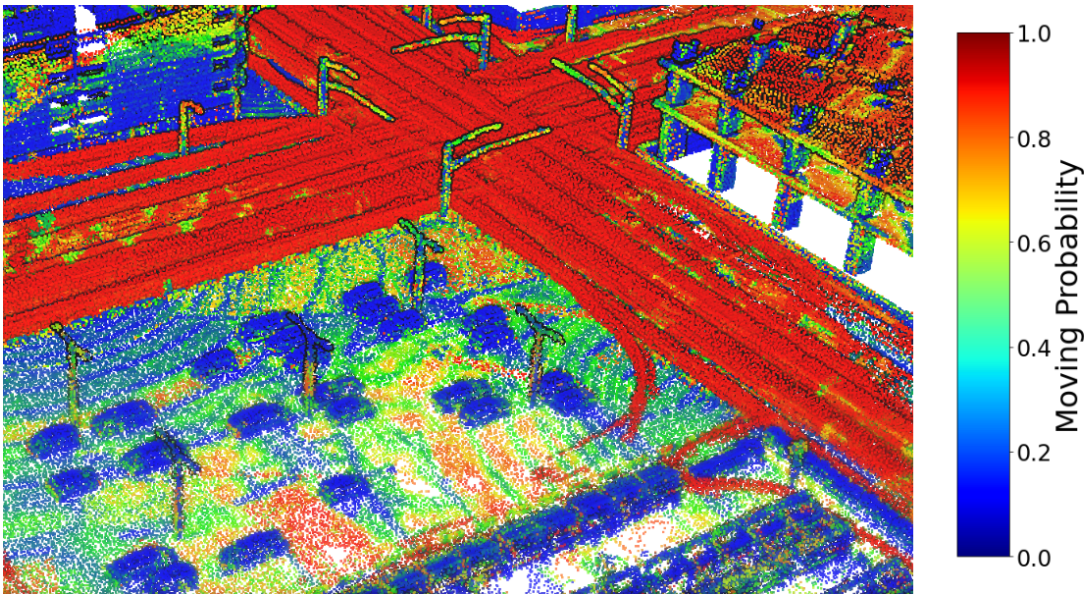
two *PointRay* operations are completely independent so we end up with two moving probabilities for map $\mathcal{M}$. In our thesis, we always use mapping session as session A and localization session as session B.

When applying two independent *PointRay* operations, Figure 3.7(a) and 3.7(b) show the difference in moving probability of a sub-region in map $\mathcal{M}$. We can easily observe that this sub-region contains points from all the four semantic classes we are interested to annotate. The fundamental difference between the two figures is the moving probability for parked vehicles. In Figure 3.7(a), points belonging to parked vehicles receive a low moving probability because these non-moving points from map $\mathcal{M}$ can always be found in the occupied space of session A. However in Figure 3.7(b), points belonging to some parked vehicles receive a high moving probability (in red). This discrepancy is because these points from map $\mathcal{M}$ do not exist in session B, and thus are often found in the freespace, resulting in a high moving probability. Again, we emphasize the importance of freespace as it is fundamental to our capability to compute moving probability. We further take advantage of this discrepancy in moving probability to distinguish a long-term movable point from a short-term movable point.

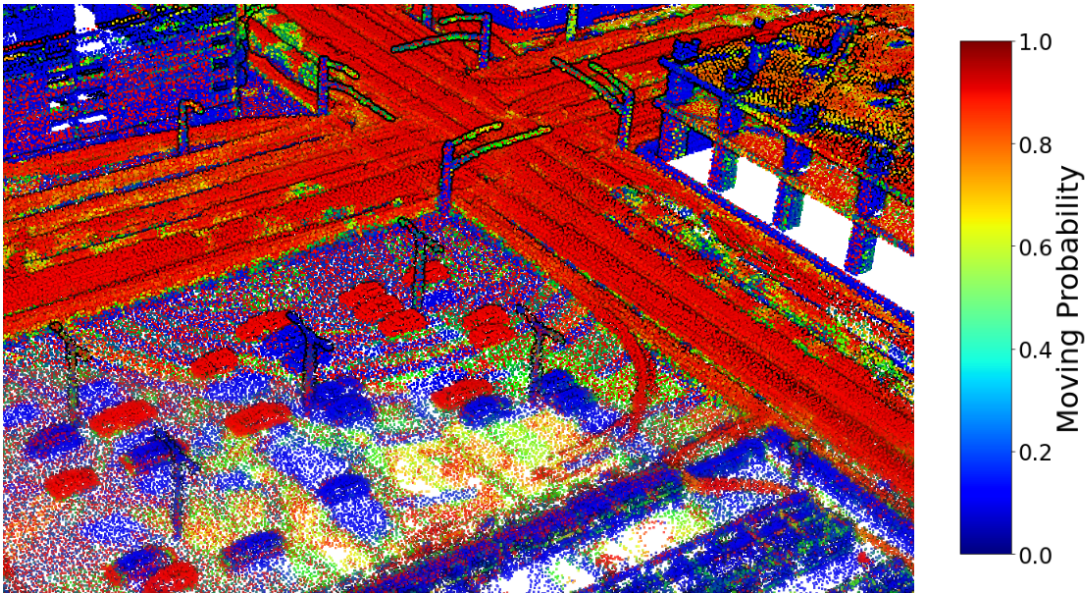
3.3.2 Annotation

Recall that we obtain maps $\mathcal{M}^{(m)}$ and $\mathcal{M}^{(l)}$ along with pose estimates $\mathbf{T}^{(m)}$ and $\mathbf{T}^{(l)}$ from the mapping and localization pipeline. We showed in the last section that we can use *PointRay* to compute the moving probability for any map $\mathcal{M}$ given a session of LiDAR scans. In this section, we describe our proposed pipeline to annotate scans $P^{(l)}$ for all localization sessions. Our method extends the original approach proposed by Thomas et al. [53] to large-scale autonomous driving scenarios. We emphasize our contributions in robustly extracting ground points for large-scale point cloud maps, accounting for the move-and-stop nature of on-road vehicles at traffic intersections, and labelling long-term movable points without carefully crafted geometric constraints.

Intuitively, we choose to annotate map $\mathcal{M}^{(l)}$ prior to annotating associated LiDAR scans $P^{(l)}$. We transfer the annotations from map $\mathcal{M}^{(l)}$ to the associated scans $P^{(l)}$ via nearest neighbor interpolation due to the grid down-sampling operation.



(a) Moving probability of a sub-region in map $\mathcal{M}$ after applying *PointRay* using scans from session A.



(b) Moving probability of a sub-region in map $\mathcal{M}$ after applying *PointRay* using scans from session B.

Figure 3.7: **(a)** Points belonging to parked vehicles in the bottom left receive a low moving probability because these points do not belong to the freespace of scans that are used to aggregate this map. However, **(b)** points belonging to some parked vehicles receive a high moving probability because they do not exist in scans from other sessions, and thus belong to the freespace.

3.3.2.1 Ground Points Extraction

Recall that we are interested in identifying points that belong to the ground. However, these points are difficult to distinguish based on moving probability alone because they have high moving probabilities like other movable objects in the scene. Therefore, we follow a similar idea in [53] to explicitly retrieve ground points from $\mathcal{M}^{(l_j)}$ without relying on any other previously computed quantities. However, while Thomas et al. [53] only need to extract one ground plane from map $\mathcal{M}^{(l_j)}$ to annotate an indoor environment, we cannot simply apply this trick due to the scale of our environment. Instead, we choose to extract ground points for every scan and identify potential ground points in $\mathcal{M}^{(l_j)}$ via a voting strategy.

To extract ground points from a scan, we choose a common approach proposed by Himmelsbach et al. [24]. We emphasize that it is a fast method to segment scans, and thus does not add much computation burden to our pipeline. The method first divides a scan into a number of spherical segments and extracts a line function based on the z coordinate of points in each segment. A point is identified as part of the ground if it is close to the line function by a distance heuristic. In our thesis, we use a distance heuristic of 0.1 m.

Once we extract ground points for every scan, we assign a l_i to every x_i in $\mathcal{M}^{(l_j)}$ and initialize it to be a zero. We carefully update l_i in the same way we update m_i for every x_i. Instead of checking whether a point belongs to the freespace of any LiDAR scans, we increment l_i by 1 if it is classified as a ground point in any scans. Using this voting strategy, points classified more often as ground points receive more votes.

Finally, we identify a point x_i in $\mathcal{M}^{(l_j)}$ as a ground point if it has been classified as a ground point in at least 10 scans, i.e $l_i >= 10$.

3.3.2.2 Map Annotation

As mentioned at the start of Chapter 3.3.2, we choose to annotate map $\mathcal{M}^{(l_j)}$ prior to transferring the annotations to its associated scans $P^{(l_j)}$.

First, we use $\mathcal{M}^{(m)}$ as the starting point of our map annotation process. We define

$\mathcal{P}^{(m,l_j)}$ to be the moving probability for $\mathcal{M}^{(m)}$ using motion-compensated scans $\bar{P}^{(l_j)}$ from the j^{th} localization session. We repeat this *PointRay* operation on map $\mathcal{M}^{(m)}$ for all localization sessions to retrieve $\mathcal{P}^{(m,l)}$

$$\mathcal{P}^{(m,l)} = \max_{j}(\mathcal{P}^{(m,l_j)}) \tag{3.6}$$

where $\mathcal{P}^{(m,l)}$ is the maximum moving probability for points in $\mathcal{M}^{(m)}$. Due to these *cross-session PointRay* operations, all the movable points in $\mathcal{M}^{(m)}$ including both long-term and short-term movable receive a high moving probability, as per the discussion related to Figure 3.7(a) and Figure 3.7(b). We take advantage of this information and use a threshold of $\tau_{\text{refine}} = 0.7$ to refine $\mathcal{M}^{(m)}$ so that it is left with only ground and non-movable points. Since we already know ground points from our ground extraction algorithm, we can easily separate ground from non-movable points. We further denote the refined $\mathcal{M}^{(m)}$ to be $\mathcal{M}^{(m)}_{\text{refine}}$.

In the next step, we move our focus to retrieve short-term movable points in $\mathcal{M}^{(l_j)}$. Similarly, we utilize *PointRay* on map $\mathcal{M}^{(l_j)}$ using scans $\bar{P}^{(l_j)}$ from the same localization session to retrieve $\mathcal{P}^{(l_j,l_j)}$. In contrast to the earlier *cross-session* operation, this provides high moving probability only for most short-term movable points, but not the long-term movable points. Therefore, we use a threshold $\tau_{\text{short}} = 0.5$ to identify these short-term movable points. However, not all short-term movable points can be identified through this approach due to the move-and-stop nature of the on-road vehicles. In a large-scale self-driving environment, some vehicles move and stop at intersections and this behaviour causes points belonging to these vehicles to have a lower moving probability compared to points belonging to a constantly moving vehicle. In our thesis, we choose to annotate points belonging to these vehicles as short-term movable even though they may not move constantly. These points are not easily identified by applying a threshold $\tau_{\text{short}} = 0.5$ to $\mathcal{P}^{(l_j,l_j)}$, and thus it requires us to compute $\mathcal{P}^{(m,m)}$ to help resolve this issue. We emphasize that we only need to compute $\mathcal{P}^{(m,m)}$ once and can re-use it to annotate the map for every localization session. We take advantage of $\mathcal{P}^{(m,m)}$ to retrieve prior information about short-term movable points in map $\mathcal{M}^{(l_j)}$ by overlaying $\mathcal{M}^{(l_j)}$ on top of $\mathcal{M}^{(m)}$.

Step	Target Class	Strategy
1	Non-movable	Moving probability lower than τ_{short} or distance to $\mathcal{M}^{(m)}_{\text{refine}}$ less than 0.2 m
2	Long-term movable	Moving probability lower than τ_{short} and distance to $\mathcal{M}^{(m)}_{\text{refine}}$ greater than 0.2 m
3	Short-term movable	Moving probability greater than τ_{short} or its nearest neighbor in $\mathcal{M}^{(m)}$ has a moving probability greater than τ_{short}
4	Ground	l_i greater than 10

Table 3.1: Steps to annotate point cloud map $\mathcal{M}^{(l_j)}$. We use the computed moving probability and ground extraction results to annotate map $\mathcal{M}^{(l_j)}$ so that every point belongs to one of the semantic classes: ground, non-movable, long-term movable and short-term movable.

We classify a point in $\mathcal{M}^{(l_j)}$ to be a short-term movable point if distance to its nearest neighbor in $\mathcal{M}^{(m)}$ is less than 0.2 m and its nearest neighbor also has a moving probability greater than $\tau_{\text{short}} = 0.5$.

Moving forward, we proceed to annotate the non-movable and long-term movable points in $\mathcal{M}^{(l_j)}$. Thanks to $\mathcal{M}^{(m)}_{\text{refine}}$, we can compare $\mathcal{M}^{(l_j)}$ to it and identify the non-movable points in $\mathcal{M}^{(l_j)}$ based on a distance metric. We heuristically choose $d_{\text{non-movable}} =$ 0.2 m so that any points in $\mathcal{M}^{(l_j)}$ that are within $d_{\text{non-movable}}$ to $\mathcal{M}^{(m)}_{\text{refine}}$ are classified as non-movable points. To retrieve long-term movable points in $\mathcal{M}^{(l_j)}$, we extend the original approach [53] to accommodate our large-scale autonomous driving scenarios. The original approach relies on surface normal directions to identify long-term movable points in an indoor scenario, which mostly have flat surfaces such as tables and chairs. In contrast, our approach does not require any carefully crafted geometric constraints. We carefully identify points in $\mathcal{M}^{(l_j)}$ that do not have a high moving probability but are at least $d_{\text{non-movable}}$ away from any points in $\mathcal{M}^{(m)}_{\text{refine}}$ as long-term movable points. In Table 3.1, we list out our annotation operations in strict order to annotate $\mathcal{M}^{(l_j)}$. We emphasize that if a point receives multiple annotations during these operations, we use its last annotation as the final annotation.

Also, an intermediate annotation $\mathcal{M}^{(m)}$ is crucial to lift our computation burden. Instead of applying *PointRay* on $\mathcal{M}^{(l_j)}$ using scans from all the other localization sessions,

we condense the joint freespace information into $\mathcal{P}^{(m,l)}$ to produce $\mathcal{M}^{(m)}_{\text{refine}}$ and reuse it to annotate every $\mathcal{M}^{(l_j)}$.

3.3.2.3 Scan Annotation

After annotating map $\mathcal{M}^{(l_j)}$ for every localization session, we transfer annotations in $\mathcal{M}^{(l_j)}$ to the associated motion compensated scans $\bar{P}^{(l_j)}$. Because we applied grid downsampling on $\mathcal{M}^{(l_j)}$, points from scan $\bar{P}^{(l_j)}$ may not find a point x_i in $\mathcal{M}^{(l_j)}$ that it perfectly coincides with. Therefore, we assign every point in scan $\bar{P}^{(l_j)}$ with the annotation of its nearest neighbor in $\mathcal{M}^{(l_j)}$. We accomplish this nearest neighbor search by taking advantage of a KD-tree.

Finally, given the annotations to $\bar{P}^{(l_j)}$, it becomes trivial to transfer annotations from motion-compensated scans $\bar{P}^{(l_j)}$ to motion distorted scans $P^{(l_j)}$ due to the identical point order. We hereby complete scan annotations for all the localization sessions using our automated annotation pipeline.

3.4 Semantic Segmentation

Our proposed pipeline aims to learn the semantic information from a point cloud in a self-supervised manner. After we retrieve automated annotations through *PointRay* in the last section, we use these annotations as *groundtruth* labels for training our semantic segmentation network.

We employ the most recent 3D point cloud semantic segmentation network KPConv [54], as it has shown state-of-the-art performance in completing semantic segmentation tasks in a large-scale environment. Also, its open-source implementation in PyTorch on GitHub allows us to freely make changes to the original code. However, we emphasize that our semantic segmentation network can be replaced with any existing segmentation network implementations.

Because KPConv has demonstrated its semantic segmentation performance in the SemanticKitti dataset, we use mostly the same parameters as the original KPConv implementation. Figure 3.8 shows our proposed semantic segmentation network architecture

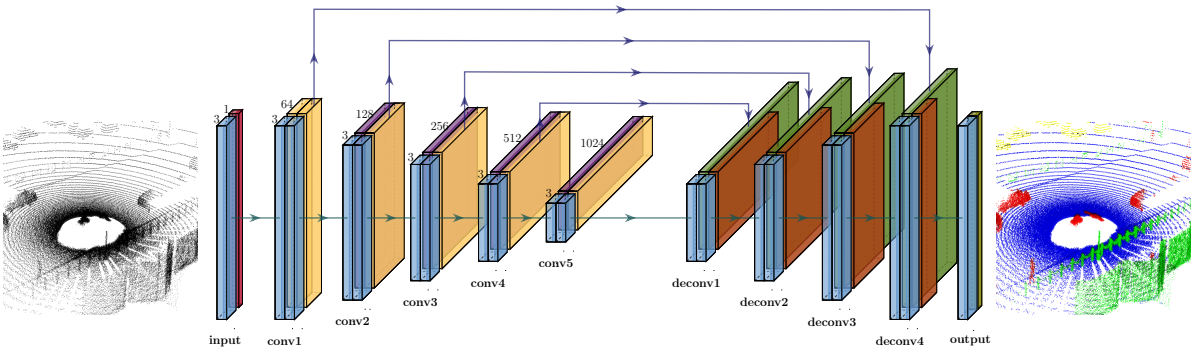


Figure 3.8: Our semantic segmentation network based on KPConv [54]. For simulation dataset, we use point cloud x, y and z coordinates as input whereas we add the intensity channel for the real-world dataset. We use KPConv (yellow) to convolve points with 3D kernels and Strided KPConv (purple) to effective reduce the number of points by half. Skip connections allow information to flow between encoder and decoder layers at the same resolution. Nearest up-samping and concatenation (green) deconvolute the learned decoder representations and output a class label for every point in point cloud.

based on KPConv. Following we describe details about how KPConv works and our design choice of crucial parameters.

KPConv applies direct point convolution through spherical kernels. Each local neighbourhood in the point cloud is convolved by applying the weights of the nearest distance kernel point in the neighbourhood. By default, these kernel points have a uniform distribution in Euclidean space. However, KPConv also offers a deformable operator that learns the kernel point positions in the network, which allows the kernel points to adapt to the topology of the local neighbourhood. For our segmentation network, we simply choose to use the default kernel points.

We have four crucial parameters to set for our network training and inference: first down-sampling grid size dl_{in}, convolution radius R_{in}, first feature dimension f_{in} and class weight w. We use dl_{in} and R_{in} to determine the size of our receptive field. We choose the value $dl_{\text{in}} = 0.2$ and $R_{\text{in}} = 2.5$ m to cover a non-trivial local region of our LiDAR scan while learning enough details about the surrounding 3D geometry to make accurate predictions. To fit more scans into a single batch, we choose $f_{\text{in}} = 64$ instead of default 128 to reduce feature dimensions while not sacrificing network prediction accuracy. Another important parameter is the class weight w. It is a manual scaling weight given to each class during training. We determine w based on the distribution of classes in our LiDAR

scans. In our large-scale self-driving environment, we tend to have more non-movable and ground points than long-term and short-term movable points. Therefore, we choose a class weight $w = [0.1, 0.2, 0.5, 0.5]$ for ground, non-movable, long-term movable and short-term movable points. The network gets penalized more when making an incorrect prediction to points belonging to long-term and short-term movable compared to ground and non-movable.

During inference, our network takes the latest LiDAR scan input and predicts a semantic label for every point in the scan. In Chapter 4, we further introduce measures to evaluate our network performance and make qualitative and quantitative analyses on our network prediction accuracy.

3.5 Summary

In this chapter, we showed our self-supervised semantic segmentation pipeline for a large-scale self-driving environment. Mapping and localization pipeline provides accurate pose estimates for LiDAR scans within a globally consistent map. Automated annotation pipeline assigns four semantic labels to annotate LiDAR scans thanks to the power of *PointRay*. Finally, our semantic segmentation network takes advantage of automated annotations and learns the LiDAR segmentation for a large-scale self-driving environment. We explicitly highlight our contributions in robustly extracting ground points for large-scale point cloud maps, accounting for the move-and-stop nature of on-road vehicles at traffic intersections, and labelling long-term movable points without carefully crafted geometric constraints.

Chapter 4

Datasets

In this chapter, we present the LiDAR datasets we have collected to evaluate our self-supervised semantic learning pipeline. We begin by describing the simulation dataset collected using the Carla simulator, an open-source software for autonomous driving development and validation. We provide details about the modifications and hyperparameters employed in generating simulated LiDAR scans. The benefit of validating our algorithm using the Carla simulator is that we can obtain groundtruth semantic labels to quantitatively evaluate our algorithm. Furthermore, to evaluate how well our algorithm works in the real world, we describe the LiDAR dataset collected using a Velodyne VLS-128 LiDAR sensor. However, we point out that the real-world dataset does not have groundtruth semantic labels.

4.1 Simulation Dataset

To quantitatively evaluate our pipeline, we collect the LiDAR dataset using the Carla simulator [19]. In this section, we elaborate on the modifications and hyperparameters we choose for the simulation environment.

4.1.1 Simulator

Carla simulator is a software built from the ground up to support the development, training and validation of the autonomous driving system. In addition to the open-source code, Carla provides open digital assets, e.g. vehicles, buildings, to facilitate that purpose. As it is built using the Unreal Engine, the Carla simulator supports flexible specification of the sensor suites, environmental conditions, full control of static and dynamic elements in the map and much more. These factors grant us the capability to customize our own assets to facilitate our autonomous driving research.

Although Carla simulator provides a simulation platform to conduct our research, it is released for general purpose use and has certain limitations that require custom modifications.

Even though Carla simulator provides various generic vehicle models to simulate real-world driving scenarios, most existing vehicle models in Carla are sedan cars such as Tesla Model 3. This limits us from simulating LiDAR scans that suffer from greater occlusions because we are also interested in evaluating our network prediction accuracy with the only partial observation of the objects. To overcome this limitation, we customize some truck models in *Blender* [16] and import our custom truck models into the Carla simulator. The truck models not only have greater dimensions compared to sedan cars but also enrich the variety of vehicle models in the simulation environment.

Another issue with the Carla simulator is that it is difficult to simulate dynamic objects across different sessions. Carla simulator provides static maps in which elements do not change their locations as they should in a real-world environment. For example, the trash on the ground in the Carla simulator stays at the same location when we collect a new session of data. In reality, they should change their locations across sessions because they are considered as dynamics. This uncommon behaviour would result in our pipeline mistakenly identifying points from the trash as non-movable points as opposed to long-term movable points. To overcome this issue, we first remove all the dynamic objects from the Carla simulator, then manually record potential locations for the dynamic objects, and finally re-generate the dynamic objects by adding some noise to the recorded

locations. Using this approach, we are able to reliably generate dynamic objects in our simulation environment.

The last issue with the Carla simulator is its traffic manager, which Carla uses to efficiently manage simulated traffic to mimic a real-world environment. When a vehicle is at an intersection, the traffic manager randomly assigns a direction for the vehicle to turn. When our vehicle is under the control of the traffic manager, it freely explores the map. However, because the map is a large and complex environment, freely navigating inside the map would take forever for our vehicle to fully explore the map. To resolve this issue, we assign waypoints to our vehicle and use the *Behavioral agent* in Carla to navigate from one waypoint to another. To avoid the vehicle from repeating exactly the same route in different sessions, we add random noise to the waypoint locations.

Recall that LiDAR laser beams emit pulsed light waves and return when hit an obstacle. The returned intensity information can also be used to improve the scene understanding of the environment, and it depends on many factors, including the physical properties of the obstacle. Although the Carla simulator is powerful in simulating the traffic in an environment, it does not have the capability of simulating the LiDAR intensity information. In addition to that, the LiDAR sensor is treated as a global shutter as opposed to a rolling shutter in the Carla simulator. Therefore, LiDAR scans from the Carla simulator do not require motion compensation.

The simulated LiDAR from the Carla simulator collects scans at a rate of 10 Hz. It resembles a Velodyne HDL-64 with 64 laser beams and a vertical field of view of 30° ranging from 5° to −25° from the horizontal level. The LiDAR collects 12,000 points in each scan with an added Gaussian noise on the endpoint positions using a standard deviation of 0.1 m.

Using the Carla simulator, we collect datasets in three different maps: Town 00, Town 01 and Town 02. Table 4.1 provides a summary of characteristics for each map. We explicitly collect datasets in three different maps because we want to enable our pipeline to generalize to complex large-scale environments as opposed to an unchanged environment in [53].

Furthermore, we generate different levels of traffic in each session. We define three

Town	Summary
Town 00	Figure-8 infinite loop with a highway and a small town; A bridge and a tunnel at the cross junction.
Town 01	Squared-grid highway encircling a small town; Fence on both sides of the road for sidewalks; Multiple lanes easy for any lane changes.
Town 02	Small town including buildings, residential houses, a parking lot, recreational parks, outdoor patios, sidewalks etc; Multiple lanes easy for any lane changes.

Table 4.1: A summary for three different maps in our simulation dataset. Each map consists of various elements and has completely different layouts.

Traffic Level	No. Vehicles	No. Pedestrians
Light	50	100
Medium	100	100
Extreme	200	400

Table 4.2: Number of vehicles and pedestrians used for each traffic level. Depending on the desired traffic level, we generate different number of vehicles and pedestrians. Unlike parked vehicles, these vehicles and pedestrians constantly move in the map.

(a) Light traffic **(b)** Medium traffic **(c)** Extreme traffic

Figure 4.1: Simulated traffic in Carla simulator with different traffic levels: light, medium and extreme. Different traffic levels simulate different numbers of moving vehicles and pedestrians around our vehicle.

different traffic levels: light, medium and extreme, and we generate a different number of moving vehicles and pedestrians around our vehicle in different traffic levels. Figure 4.1(a), Figure 4.1(b) and Figure 4.1(c) show an example of simulated traffic in the Carla simulator with different traffic levels. For each map, we collect one session with light traffic and use it as the mapping session. We collect three other sessions with light traffic, three sessions with medium traffic and two sessions with extreme traffic as localization sessions. Our vehicle drives approximately 4 km in each session, collecting 6,000 to 12,000 LiDAR scans depending on different traffic levels.

4.2 Real-World Dataset

To validate our proposed pipeline on real-world data, we use a real-world dataset primarily collected by Burnett et al. [10] with a data-taking platform *Boreas*, as shown in Figure 4.2, from Autonomous Space Robotics Laboratory (ASRL).

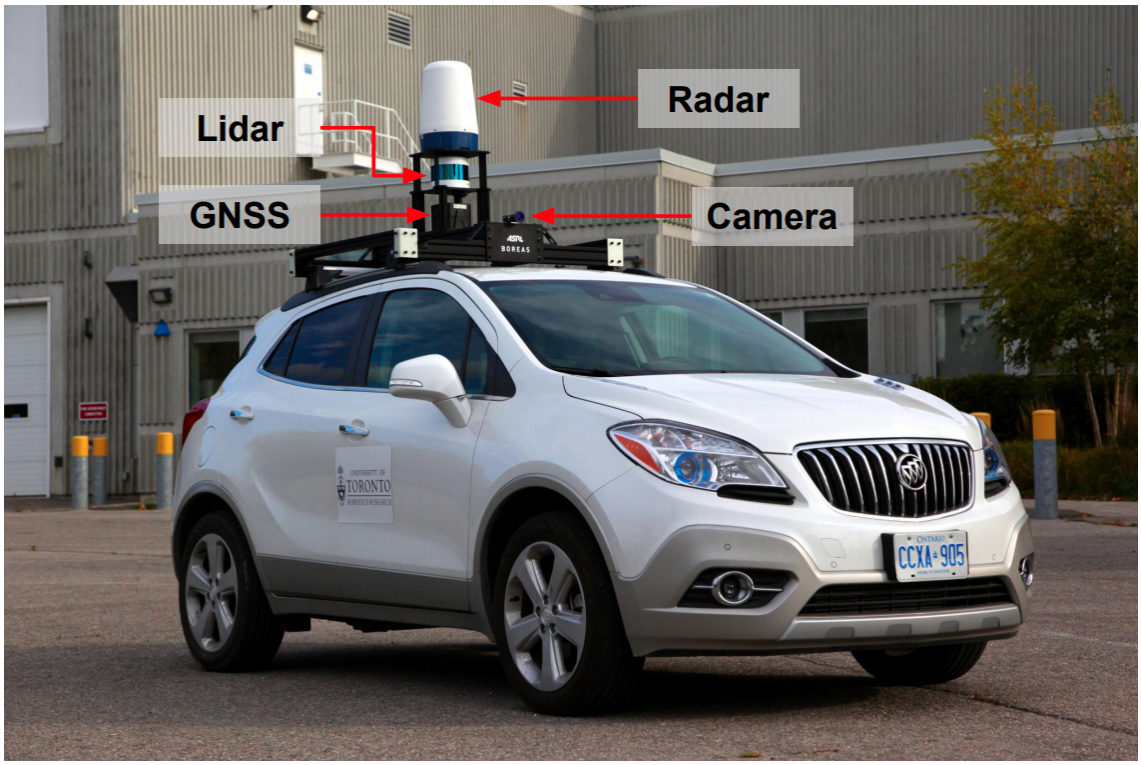


Figure 4.2: Our data-taking platform *Boreas* from ASRL. It includes a Velodyne VLS-128 LiDAR sensor and an Applanix POS-LV inertial navigation system.

The onboard sensor suite includes a Velodyne VLS-128 LiDAR sensor and an Applanix POS-LV inertial navigation system. The LiDAR sensor is time synchronized with the Applanix POS-LV to obtain accurate timing information. Applanix POS-LV inertial navigation system includes an inertial measurement unit (IMU), two global positioning system (GPS) antennas and a wheel encoder. We specifically acquire two GPS antennas to obtain a more accurate baseline and better RTX performance.

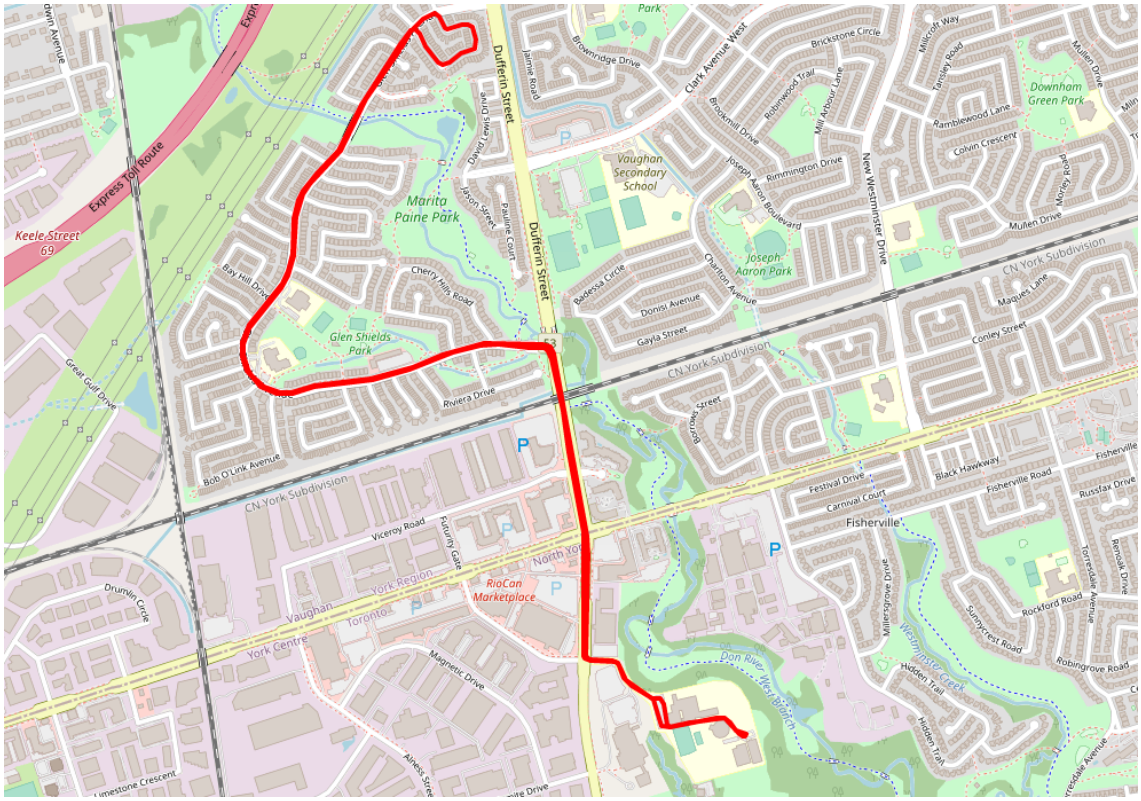


Figure 4.3: Our data collection route using *Boreas* from ASRL. The route starts from UTIAS, turns right onto Dufferin Street, North York, Ontario, and turns left into a local residential neighbourhood. The vehicle takes a detour and returns to its start position.

We operate *Boreas* and log LiDAR data at a rate of 10 Hz by repeatedly driving along a designated route on different days to satisfy the requirements from a mapping and localization pipeline. We especially drive on Dufferin Street, North York, Ontario and in a local residential neighbourhood near the University of Toronto Institute for Aerospace Studies. Figure 4.3 shows an example of our vehicle trajectory.

To the date this thesis is completed, we have collected more than twenty sessions on the designated route. In this thesis, we use one session as the mapping session and

another five sessions as the localization sessions to generate automated annotations. The rest of the sessions can be used as unseen data to validate our approach. We further point out that each session is around 10 km travel, containing nearly 10,000 frames.

4.3 Summary

In this chapter, we present our data collection strategy to evaluate our self-supervised semantic learning pipeline. To make a quantitative analysis, we collect a simulation dataset using the Carla simulator. We make modifications to the open-source software to suit our needs, e.g. greater occlusions with new truck models and customized dynamics spawning strategy. To validate our pipeline in a real-world environment, we use a real-world dataset from ASRL. Its onboard sensor suite includes a Velodyne VLS-128 LiDAR sensor and an Applanix POS-LV inertial navigation system that can be used to retrieve groundtruth poses. Unfortunately, unlike the Carla simulator, we are unable to retrieve groundtruth semantic labels from the real-world dataset.

Chapter 5

Experimental Results

In this chapter, we provide a qualitative and quantitative evaluation of the individual components introduced in Chapter 3. Using the simulation and real-world dataset, we carefully make an analysis of the results from our semantics-aware localization pipeline.

We begin with introducing our experimental setup followed by the methods to evaluate each individual component. Then we present our qualitative and quantitative evaluation on each individual component using the simulation dataset. Next, we show our qualitative results from the real-world dataset. Finally, we evaluate our semantics-aware localization pipeline.

5.1 Experimental Setup

Our self-supervised semantic learning pipeline for large-scale scene understanding can be divided into smaller individual components, including the mapping and localization pipeline provided by Applanix, the automated annotation pipeline and the semantic segmentation network. In this section, we describe our experimental setup in using simulation and real-world datasets to validate our pipeline.

5.1.1 Simulation Dataset

In Chapter 4.1 we collected a simulation dataset using the Carla simulator. The dataset includes a number of sessions collected in three different maps. For each map, we use our mapping and localization pipeline to retrieve a globally consistent map and pose estimates for localization sessions. We use one session with light traffic as the mapping session and the rest sessions with light and medium traffic as localization sessions. Finally, we obtain pose estimates for these five localization sessions.

Using the pose estimates from the mapping and localization pipeline, we feed them into our automated annotation pipeline for scan annotation. For each map, we annotate scans from five localization sessions, producing a total of approximately 40,000 annotated scans.

However, we do not use all the annotated scans in our semantic segmentation network for training. Instead, annotated scans from Town 00, Town 01 and 02 are used for training, validation, and testing, respectively. This ensures us that these sets are completely independent of each other.

Finally, to show our improvement to the localization pipeline, we choose to semantically filter out points in point clouds using our network predictions. We only keep non-movable points in the scan to perform localization. This ensures that only points invariant to the environment, i.e., non-movable points, are localized against the map. To demonstrate our semantics-aware localization pipeline, we use Town 02 as our test set.

5.1.2 Real-world Dataset

In Chapter 4.2 we mentioned that a real-world dataset is collected using a Velodyne VLS-128 LiDAR sensor. Although the dataset has more than twenty sessions, we use one session as a mapping session and another five sessions as localization sessions for our mapping and localization pipeline to retrieve accurate pose estimates.

Using these pose estimates, we feed scans into our automated annotation pipeline for scan annotation. Unlike simulated LiDAR scans from the Carla simulator, these scans require motion compensation to achieve accurate annotation results. Finally, we retrieve

approximately 40,000 annotated scans from five localization sessions.

As opposed to the simulation dataset, we use all the annotated scans for training and validation. For testing, we choose another three independent sessions from the twenty sessions we have collected. Even though there are no overlapping scans between training, validation and test sets, these sets are collected in the same environment, and thus potentially introducing unknown bias to the network prediction.

Finally, to show our improvement to the localization pipeline, we apply our semantics-aware localization pipeline on testing data.

5.2 Evaluation Methods

We break down our pipeline into the mapping and localization pipeline provided by Applanix, the automated annotation pipeline and the semantic segmentation network. In this section, we introduce methods we employ to quantitatively evaluate the performance of each individual component.

Our mapping and localization pipeline can be sub-divided into two components: the mapping component and the localization component. While the mapping component finds the optimal pose estimates to produce a globally consistent map, the localization component finds the optimal pose estimates to match scans within a globally consistent map. Here we only evaluate the localization performance by comparing the estimated poses to the groundtruth provided by Applanix POS-LV. We introduce multiple metrics to evaluate localization, where $\hat{(\cdot)}$ denotes a quantity from our estimation.

1. Horizontal RMS: Root Mean Square (RMS) error along the x and y horizontal directions

$$\text{Horizontal RMS} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x_i})^2 + (y_i - \hat{y_i}^2)}{N}} \tag{5.1}$$

2. Horizontal Max: Maximum L_2 norm between the estimate and groundtruth trajec-

tory along the horizontal directions

$$\text{Horizontal Max} = \max_{i \in N} \sqrt{(x_i - \hat{x_i})^2 + (y_i - \hat{y_i}^2)} \tag{5.2}$$

3. < 0.1 m Pct: Percentage of scans that has horizontal RMS less than 0.1 m
4. < 0.3 m Pct: Percentage of scans that has horizontal RMS less than 0.3 m
5. < 0.6 m Pct: Percentage of scans that has horizontal RMS less than 0.6 m
6. Yaw RMS: RMS error about the yaw direction, i.e., vehicle heading

$$\text{Yaw RMS} = \sqrt{\frac{\sum_{i=1}^{N} (\phi_i - \hat{\phi_i})^2}{N}} \tag{5.3}$$

7. Yaw Max: Maximum RMS error along the yaw direction

$$\text{Horizontal Max} = \max_{i \in N} \sqrt{(\phi_i - \hat{\phi_i})^2} \tag{5.4}$$

8. $< 0.1^\circ$ Pct: Percentage of scans that has yaw RMS less than 0.1°
9. $< 0.3^\circ$ Pct: Percentage of scans that has yaw RMS less than 0.3°
10. $< 0.6^\circ$ Pct: Percentage of scans that has yaw RMS less than 0.6°

To evaluate our annotation pipeline quantitatively, we compare our annotations to the groundtruth for the simulation dataset. We introduce the confusion matrix to evaluate this multi-class classification problem. In Table 5.1 we use an example to demonstrate the confusion matrix by predicting four classes: A, B, C and D. We define true positives (TP) as correctly predicting a class whereas false positives (FP) as falsely predicting a class. We also define false negatives (FN) as falsely predicting other classes. We take a closer look at the prediction for class A especially. Its true positives (TP_A) can be quantified by the number correctly predicting class A. Its false positives (FP_A) are numbers falsely predicting a class A when the actual class are either B, C or D. Similarly, its false

negatives (FN_A) are numbers falsely predicting a class B, C or D when the actual class is A. In our problem, we are more interested in the true positives, i.e., diagonal elements in Table 5.1, as they reflect our correct prediction of a class.

Predicted Class / Actual Class	A	B	C	D
A	TP_A	FN_A		
B	FP_A	TP_B		
C			TP_C	
D				TP_D

Table 5.1: Confusion matrix for evaluating a 4-class classification problem. The goal is to correctly predict classes A, B, C and D. While the diagonal elements are the true positives, the columns and rows comprise false positives and false negatives of a class. We illustrate this by taking class A as an example.

Since the number of points belonging to a class is different, we normalize the confusion matrix by rows and make a quantitative analysis on the percentage of correct predictions for each class. Unfortunately for the real-world dataset, because the groundtruth class labels do not exist, we cannot build a confusion matrix or make any quantitative analysis of our annotations.

For the simulation dataset, we evaluate our network prediction accuracy by comparing our network predictions with the groundtruth. Using exactly the same approach, we build the confusion matrix to quantitatively evaluate our network predictions. However, for the real-world dataset, we choose to only present our qualitative result due to the missing groundtruth.

To evaluate our semantics-aware localization pipeline, we re-use the same evaluation metrics for the aforementioned mapping and localization pipeline. Finally, we compare our semantics-aware localization pipeline to the former generic localization pipeline.

5.3 Mapping and Localization Evaluation

In this section, we evaluate our localization pipeline using the metrics we described in Chapter 5.2. Localization performance is important to our algorithm because it pro-

vides accurate pose estimates that are the foundation of our automated annotation pipeline. We avoid an inaccurate or failed localization solution as it would cause our algorithm to falsely annotate points. We explicitly distinguish this localization solution from semantics-aware localization as it does not rely on any semantic information.

We evaluate our localization performance quantitatively using both simulation dataset and real-world dataset. For the simulation dataset, we only use sessions with light and medium traffic for localization because they usually produce accurate pose estimates. We then use these pose estimates to help us annotate LiDAR scans. We purposely avoid sessions with extreme traffic because we tend to find them with poor pose estimates. In Table 5.2, we evaluate localization by comparing our pose estimates to groundtruth from the Carla simulator. Also, we average the evaluation metrics in Table 5.2 for similar sessions.

Town	Traffic Level	Hori. RMS	Hori. Max	< 0.1m Pct.	< 0.2m Pct.	< 0.3m Pct.	Yaw RMS	Yaw Max	$< 0.1°$ Pct.	$< 0.3°$ Pct.	$< 0.6°$ Pct.
00	Light	0.094	0.453	82.46	96.77	99.13	0.043	0.377	99.11	99.93	99.98
	Medium	0.088	0.374	85.81	97.44	99.76	0.024	0.265	99.64	99.95	99.99
01	Light	0.071	0.347	87.26	98.82	99.89	0.029	0.311	99.68	99.98	100.0
	Medium	0.086	0.401	79.45	96.92	99.33	0.030	0.327	99.59	99.97	99.97
02	Light	0.068	0.325	89.16	98.25	99.82	0.020	0.194	99.71	99.98	99.99
	Medium	0.074	0.317	88.92	95.36	99.11	0.032	0.331	99.10	99.96	99.98

Table 5.2: Localization evaluation on sessions with light and medium traffic levels. The evaluation metrics are averaged among sessions with the same traffic level in the same map.

We can see from Table 5.2 that our mapping and localization pipeline provides accurate pose estimates in both light and medium traffic compared to the groundtruth. Low horizontal RMS and yaw RMS guarantee us decent scan-to-map alignment when we apply *PointRay* in the map annotation process. Our localization performance is also consistent throughout the entire trajectory as suggested by the high percentage of pose estimates below error bounds. Furthermore, it shows that our localization pipeline performs equally well in scenes with different characteristics.

In addition to the simulation dataset, we also evaluate our localization performance on the real-world dataset using post-processed GPS poses from Applanix LV. Table 5.3

Session	Hori. RMS	Hori. Max	< 0.1m Pct.	< 0.2m Pct.	< 0.3m Pct.	Yaw RMS	Yaw Max	$< 0.1°$ Pct.	$< 0.3°$ Pct.	$< 0.6°$ Pct.
1	0.098	0.444	73.09	96.15	99.54	0.054	0.604	99.02	99.82	99.95
2	0.112	0.487	57.81	92.14	98.11	0.051	0.589	99.23	99.91	99.97
3	0.093	0.451	72.48	97.02	99.24	0.047	0.366	99.68	99.93	99.98
4	0.124	0.515	44.91	89.77	96.12	0.081	0.681	98.13	99.13	99.86
5	0.118	0.418	57.82	90.11	97.73	0.067	0.512	99.29	99.27	99.90

Table 5.3: Localization evaluation on real-world datasets.

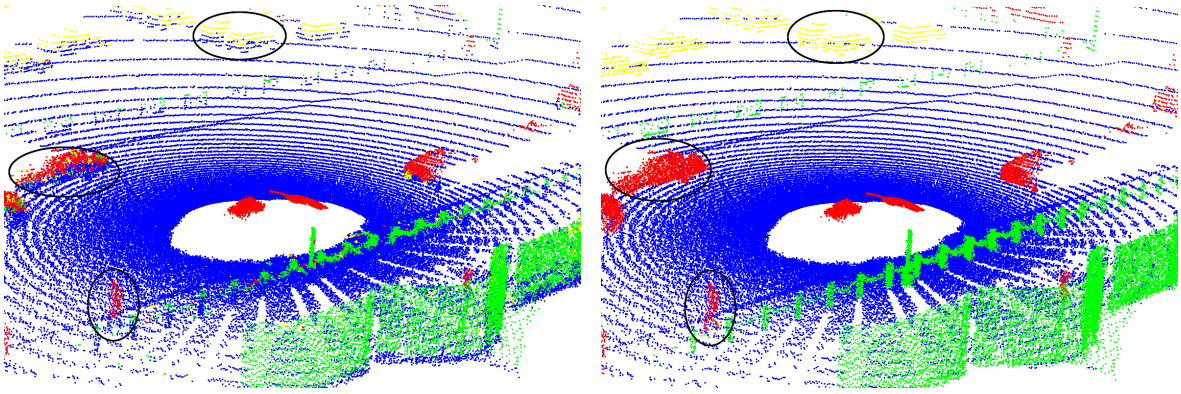
presents our localization performance when localizing multiple sessions in a pre-generated map. Although localization sessions are collected on different days, our localization pipeline continues to produce accurate pose estimates. From a quantitative perspective, localizing in the real world has greater errors compared to localizing in a simulated environment. Despite that, the pose estimates are well-bounded, and thus can still be used to help annotate LiDAR scans.

5.4 Annotation Evaluation

In this section, we compare our annotation on the simulation dataset with the groundtruth from the Carla simulator and provide a qualitative and quantitative evaluation of our annotation accuracy. Recall that our automated annotation pipeline annotates LiDAR scans using four class labels: ground, non-movable, long-term movable and short-term movable. As discussed in Chapter 5.2, we employ the confusion matrix to evaluate this multi-class classification problem.

Figure 5.1(a) and Figure 5.2(b) present an example of our annotated scan in comparison with the groundtruth. Ground, non-movable, long-term movable and short-term movable points are annotated with blue, green, yellow and red. From a qualitative perspective, our ground extraction algorithm manages to capture most of the ground points. However, our annotations in the black circled areas suggest that our ground extraction is quite aggressive in that it also captures non-ground points that are close to the ground plane. The non-movable points in the scan, i.e., walls and fence, are successfully identi-

fied through our annotation pipeline despite some false annotations close to the ground. The main feature that distinguishes our annotation pipeline from other dynamic object detection algorithms is our detection of long-term movable points in the scan. The top circled yellow points are parked vehicles in a parking lot which are successfully identified as long-term movable points. Through *PointRay*, these points receive low moving probability and thus are unlikely to move compared to points from on-road vehicles.



(a) Annotated scan **(b)** Groundtruth scan

Figure 5.1: Comparison of annotated and groundtruth scan examples from the testing set of our simulation dataset using the automated annotation pipeline. The scan is annotated with four classes: ground (blue), non-movable (green), long-term movable (yellow) and short-term movable (red).

In Table 5.4 we present the confusion matrix to provide a quantitative evaluation of our annotation. We emphasize that this confusion matrix is computed from all our annotated scans. As suggested by the diagonal elements, we achieve close to perfect extraction of ground points. We do decently well in extracting non-movable points and short-term movable points, yet fail to extract half of the long-term movable points. From the first column of the confusion matrix, we can easily observe that a lot of false annotations happen in cases where we incorrectly annotate points to be ground when they belong to other classes. This observation aligns with our previous hypothesis that the ground extraction algorithm is overaggressive. Also, nearly 25% of long-term movable points are identified as non-movable and short-term movable points. We identify it as a trade-off to address the move-and-stop nature of on-road vehicles because we purposely choose a relatively low τ_{short} to capture more short-term movable points from on-road vehicles.

This inevitably includes more long-term movable points as short-term movable points and causes our long-term movable and short-term movable points to intermix. This can be well noticed by 12.2% of short-term movable points being misclassified as long-term movable points.

Groundtruth \ Annotation	ground	non-movable	long-term movable	short-term movable
ground	99.49%	0.33%	0.11%	0.06%
non-movable	21.82%	73.97%	2.66%	1.54%
long-term movable	26.64%	15.06%	48.49%	9.82%
short-term movable	8.46%	3.99%	12.21%	75.35%

Table 5.4: Confusion matrix between groundtruth and annotations, normalized by rows. Diagonal elements suggest the correct prediction (TP) of each class. While the rest of the rows suggest the percentage of misclassification to other classes, the column suggests false annotations to a class.

Finally, we conclude our annotation performance on the simulation dataset. We rely on an aggressive ground extraction algorithm to achieve nearly perfect ground annotations, yet sacrifice annotation accuracy on other classes. Our long-term movable annotation especially fails to detect half of the points due to our attempt to address the move-and-stop issue with on-road vehicles. Despite the imperfections in our scan annotations, we hope our semantic segmentation network can robustly learn the true positives from a number of outliers.

5.5 Network Prediction Evaluation

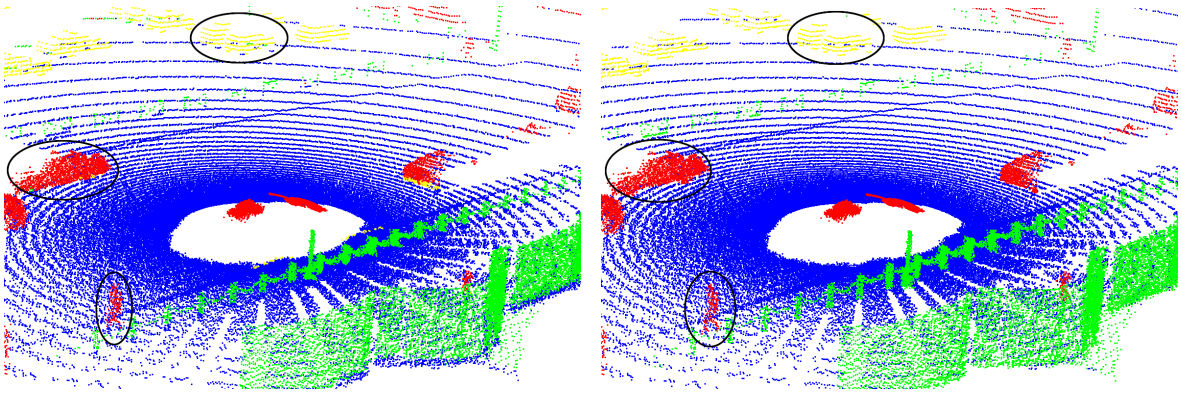
In this section, we train our semantic segmentation network using annotated scans from the simulation dataset and compare our network predictions to the groundtruth from the Carla simulator to provide a qualitative and quantitative analysis. Our network consumes the latest LiDAR scan input and does not rely on any temporal information as other dynamic object detection approaches. [63] For the network training, we use annotated scans from Town 00 and Town 01 as training and validation sets, respectively. We use one Tesla V100 GPU and reach convergence after about 12 hours. Table 5.5 lists our network training parameters.

Hyperparameters	**Value**
Optimizer	Stochastic Gradient Descent (SGD)
Learning Rate	0.01
Learning Rate Decay	Yes
Momentum	0.98
Loss	Cross Entropy Loss
Class Weight	ground: 0.1, non-movable: 0.2 long-term movable: 0.5, short-term movable: 0.5
Batch Size	8
Data Augmentation	Yes

Table 5.5: List of network training parameters for learning semantic segmentation of LiDAR scans.

Figure 5.2(a) and Figure 5.2(b) present an example of our predicted scan from Town 02 test set in comparison with the groundtruth. We use the same colour code as in Chapter 5.4. Our network makes correct predictions to most of the ground points in this example scan except for being overaggressive in the bottom right corner. As opposed to its aggressive pattern in the annotating process in Figure 5.1(a), our network prediction becomes more conservative and no longer falsely assigns ground labels to points close to the ground plane. Furthermore, an important observation is that our network learns to distinguish the parked vehicle in the top circle from the on-road vehicle on the left even though they have a very similar geometry. Our hypothesis is that our network has learned to identify whether points from a vehicle belong to long-term movable or short-term movable by interpreting 3D geometry from not only the vehicle but also the surrounding environment, thanks to a large receptive field. Also, for small moving objects, such as the pedestrian in the bottom circle, our network has successfully identified them as short-term movable points. It is likely that our network has learned to interpret a pedestrian model and it auto-assigns any points from it to be short-term movable.

Table 5.6 presents the confusion matrix computed from our predictions on Town 02 test set to quantitatively evaluate our network prediction accuracy. As discussed earlier, we observe that our ground point detection is becoming more conservative compared to our annotation. This is reflected in the true positives and false positives for ground points. Even though the true positives drop from 99.49% to 93.54%, the false positives



(a) Predicted scan **(b)** Groundtruth scan

Figure 5.2: Comparison of predicted and groundtruth scan examples from the testing set of our simulation dataset using best-trained network parameters. Our network learns to assign four class labels to understand the scene.

also reduce significantly. This indicates that fewer points are incorrectly classified as ground points, and it explains why there is a significant improvement in the prediction accuracy of non-movable points, from 73.97% to 92.69%. Also, we are seeing slight improvement on predicting long-term movable points from 48.49% to 55.90%. Even though the improvement is trivial, we are seeing a shift in the distribution of its false negatives. In Table 5.4 most misclassification for long-term movable points comes from either ground or non-movable points, yet our network predictions suggest that it now comes from short-term movable instead. This indicates that our network becomes worse in identifying short-term movable from long-term movable compared to our annotations. Even though we have qualitatively shown the network's ability to identify a parked vehicle from an on-road vehicle, the quantitative result suggests an unsatisfying capability of our network to identify all the long-term movable points in the scene.

Finally, we conclude our network prediction performance on the simulation dataset. Our network learns to predict class labels from the latest LiDAR scan input. Compared to our annotations, the network is more conservative in predicting ground points, which in return improves the prediction accuracy on other classes. However, as suggested by the quantitative results in Table 5.6, predicting and identifying long-term movable and short-term movable from each other remain challenging despite slight improvements.

Groundtruth \ Prediction	ground	non-movable	long-term movable	short-term movable
ground	93.54%	5.98%	0.15%	0.34%
non-movable	4.32%	92.69%	1.45%	1.45%
long-term movable	16.12%	3.88%	55.90%	24.09%
short-term movable	0.48%	0.82%	23.49%	75.20%

Table 5.6: Confusion matrix between groundtruth and network predictions, normalized by rows. Diagonal elements suggest the correct prediction (TP) of each class. While we improve on nearly all classes, we still face challenges when long-term movable and short-term movable intermix.

5.6 Semantics-Aware Localization Evaluation

In this section, we present a quantitative evaluation of our semantics-aware localization pipeline. Our pipeline uses network predictions as visual cues to semantically filter out points from LiDAR scans to further improve localization. We only keep non-movable points in our scans as they are permanent to the environment. We pass semantically filtered scans into our localization pipeline in order to localize against the pre-generated map and compare our performance to the generic localization pipeline without any semantic filtering.

For the simulation dataset, we conduct localization experiments on all sessions with different traffic levels. Table 5.7 presents our localization performance in comparison with the generic localization. We refer to setting 1 as the generic localization pipeline without semantic filtering and setting 2 as our proposed semantics-aware localization pipeline. We emphasize that sessions 1 and 2, 3 and 4, 5 and 6 are simulated with extreme, medium and light traffic to test the robustness of localization. Intuitively, it is more challenging for a vehicle to localize within a pre-generated map with more traffic as it introduces greater occlusions. Our experiments show that in challenging scenarios with extreme traffic (session 1 and 2), semantically filtering out the ground and movable points in the scans helps retain an accurate pose estimate when generic localization fails due to the abundance of moving traffic, causing localization solution to drift. Figure 5.3(a) and Figure 5.3(b) show an example of a scan in extreme traffic before and after semantic filtering. By keeping only non-movable points in the scan, it becomes easier for

localization to match permanent structures to the pre-generated map. Furthermore, our experiments on sessions with decent traffic (session 3-6) suggest that our semantics-aware localization pipeline performs equally or better than the generic localization approach, even though not by a large margin.

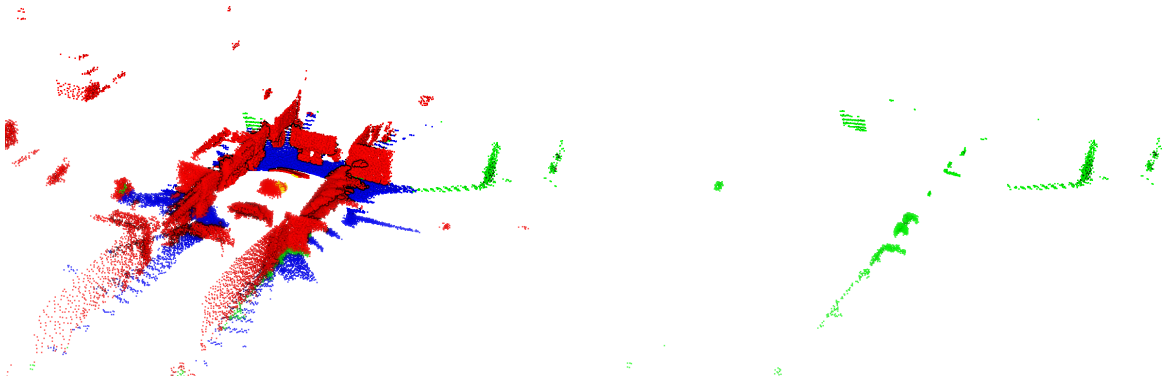
Session	Setting	Hori. RMS	Hori. Max	< 0.1m Pct.	< 0.2m Pct.	< 0.3m Pct.	Yaw RMS	Yaw Max	$< 0.1°$ Pct.	$< 0.3°$ Pct.	$< 0.6°$ Pct.
1	1	9.757	198.8	68.43	91.56	95.17	11.55	174.19	84.63	97.14	97.59
	2 (ours)	**0.086**	**0.416**	**78.83**	**97.09**	**99.87**	**0.026**	**0.453**	**99.39**	**99.97**	**99.98**
2	1	54.49	378.3	64.70	80.74	86.42	3.21	168.72	94.95	97.11	97.39
	2 (ours)	**0.788**	**19.60**	**69.69**	**84.10**	**88.13**	**0.050**	**0.41**	**94.32**	**99.90**	**100.0**
3	1	0.073	**0.308**	**85.83**	98.49	99.94	0.025	0.348	**99.74**	**99.94**	**99.97**
	2 (ours)	**0.071**	0.348	85.62	**98.83**	**99.94**	**0.024**	**0.237**	99.13	**99.97**	**99.97**
4	1	0.069	0.355	87.04	98.55	99.89	**0.020**	0.179	99.67	**99.98**	**99.98**
	2 (ours)	**0.066**	**0.328**	**88.31**	**98.62**	**99.98**	**0.020**	**0.178**	**99.68**	**99.98**	**99.98**
5	1	0.100	**0.503**	**73.75**	93.41	99.18	**0.029**	**0.620**	**99.29**	**99.89**	**99.96**
	2 (ours)	**0.091**	0.539	74.81	**96.83**	**99.33**	0.031	0.627	99.10	99.85	99.95
6	1	0.097	**0.481**	**77.24**	95.88	99.31	0.024	**0.359**	**99.45**	99.92	**99.99**
	2 (ours)	**0.089**	0.487	78.10	**96.03**	**99.69**	**0.022**	0.387	99.44	**99.93**	**99.99**

Table 5.7: Localization evaluation on the test set with various traffic levels. Intuitively, with more traffic in the environment, it is more difficult for our vehicle to localize against the pre-generated map.

Finally, we conclude our semantics-aware localization pipeline. By learning semantics using a semantic segmentation network, our semantics-aware localization pipeline filters out ground and movable points from scans and use the rest of the points for localization. Experiments show that it successfully resolves failures cases when the robot attempts to localize in extremely busy traffic. Furthermore, when evaluated on sessions with common traffic, it performs equally or better than the generic localization approach.

5.7 Real-world Dataset Evaluation

From Chapter 5.4 to Chapter 5.6 we provide a qualitative and quantitative evaluation of our method on the simulation dataset. In this section, we provide a qualitative evaluation of our semantic segmentation network on the real-world dataset and make a quantitative



(a) Scan before semantic filtering **(b)** Scan after semantic filtering

Figure 5.3: Comparison of a LiDAR scan captured in extreme traffic before and after semantic filtering. While generic localization estimates pose using all points in **(a)**, our semantics-aware localization filters out ground (blue), long-term movable (yellow) and short-term movable (red) points from the scan **(b)** before localizing against the pre-generated map. Under extreme traffic, generic localization fails due to the abundance of dynamics in the environment, i.e., on-road vehicles (red) in **(a)**, whereas our semantics-aware localization successfully localizes using non-movable points in **(b)**.

analysis of our semantics-aware localization. We train a semantic segmentation network using real-world scan annotations and present an example of our network predictions on a test set scan from the real-world dataset in Figure 5.4. From a qualitative perspective, we easily notice that the network captures ground points well thanks to their simple 3D geometry. Non-movable points such as walls, poles and traffic signs are also well segmented. The segmentation on billboards in the bottom right circle suggests that our network has also learned the capability to segment small objects in a large-scale scene. Furthermore, our approach manages to distinguish long-term movable from short-term movable, especially parked vehicles from on-road vehicles. The vehicles parked in the doorway (right circle) are successfully identified as long-term movable. Compared with the on-road vehicles, these parked vehicles have different surroundings (trees) that the network learns to distinguish from. Despite all this, the network still incorrectly identifies part of an on-road vehicle as long-term movable as circled on the left. This misclassification suggests that our network still needs improvement in robustly distinguishing long-term movable and short-term movable points.

Table 5.8 presents our semantics-aware localization performance on the test set. As

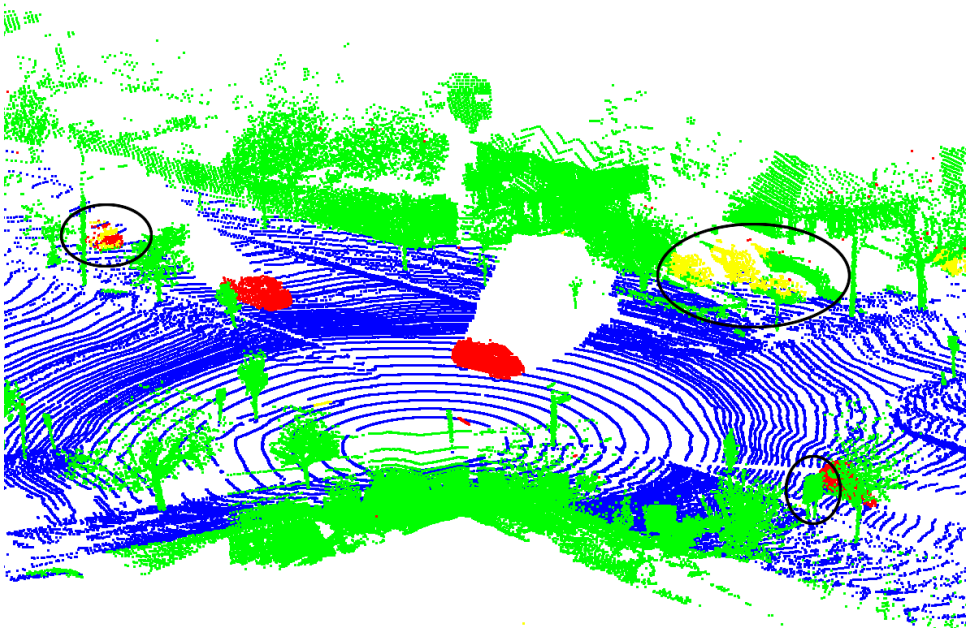


Figure 5.4: Network predictions on the test set from the real-world dataset.

mentioned earlier, we retrieve test set from more than twenty sessions of collected data that are neither used for mapping nor localization. We can easily observe that by semantically filtering out the ground and movable points in the scans, our semantics-aware localization pipeline performs on par with the generic localization pipeline. The reason we do not see a large improvement in the localization performance is that our data is collected in a local residential neighbourhood, which does not contain an abundance of dynamic objects. Therefore, semantically filtering out the ground and movable points do not significantly help localization as it already has enough permanent structures for scan-to-map alignment.

Session	Setting	Hori. RMS	Hori. Max	< 0.1m Pct.	< 0.2m Pct.	< 0.3m Pct.	Yaw RMS	Yaw Max	$< 0.1°$ Pct.	$< 0.3°$ Pct.	$< 0.6°$ Pct.
1	1	0.105	0.423	73.79	**96.35**	**99.53**	0.050	0.592	99.14	99.78	99.94
	2 (ours)	**0.098**	**0.416**	**75.63**	96.10	99.42	**0.049**	**0.584**	**99.28**	**99.83**	**99.97**
2	1	**0.104**	0.531	70.91	90.60	98.89	**0.044**	**0.493**	**99.32**	**99.88**	**100.0**
	2 (ours)	0.106	**0.489**	**71.74**	**92.47**	**99.18**	0.045	0.497	**99.32**	99.86	**100.0**
3	1	0.115	**0.502**	68.97	88.98	99.08	0.049	**0.459**	99.12	99.82	**99.99**
	2 (ours)	**0.110**	0.534	**70.59**	**91.35**	**99.12**	**0.047**	0.517	**99.20**	**99.84**	**99.99**

Table 5.8: Localization evaluation on the test set for the real-world dataset.

Finally, we conclude our semantics-aware localization pipeline on a real-world dataset. Our semantic segmentation network learns decent class labels from our qualitative evaluation. However, when we semantically filter out points in the scans, the quantitative improvement on localization is trivial due to the lack of dynamics in the environment.

5.8 Summary

In this chapter, we presented the results of our self-supervised semantic learning pipeline. Using the metrics discussed earlier in the chapter, we break down our pipeline into individual components and make qualitative and quantitative analyses on results from the simulation dataset. Furthermore, we evaluate our network predictions qualitatively on the real-world dataset and provide quantitative analysis to our semantics-aware localization pipeline. From our experimental results on both datasets, we conclude that our proposed method performs well in annotating LiDAR scans and making accurate predictions of semantic classes. The proposed semantics-aware localization pipeline demonstrates improvement to localization in busy traffic. However, due to the lack of dynamic elements, our real-world dataset proves to be insufficient to highlight the benefit of our approach.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

This thesis presents a self-supervised semantic learning approach for large-scale scene understanding. Our goal is to identify movable points in a LiDAR scan. Instead of segmenting the scene into static and dynamic points, we learn to segment point clouds into more descriptive semantic classes: ground, non-movable, long-term movable and short-term movable. We accomplish that by training a semantic segmentation network using groundtruth annotations. However, acquiring hand-labelled groundtruth annotations is expensive. We address this issue via an automated annotation pipeline that takes advantage of ray-tracing and pose estimates from a mapping and localization solution. Compared to earlier work in [53], our method extends its application to a large-scale environment. We utilize a more robust ground extraction technique, address the move-and-stop nature of on-road vehicles, label long-term movable points without carefully crafted geometric constraints, and evaluate our approach in a semantics-aware localization pipeline.

Unfortunately, a public dataset that suits our needs does not exist due to the difficulty in acquiring the groundtruth labels. To evaluate our approach, we collect a simulation dataset using the Carla simulator and a real-world dataset using the data collection vehicle from ASRL. Our simulation dataset provides us with a quantitative benchmark to evaluate individual components in our pipeline. Furthermore, we validate our approach

and make qualitative analyses on results using a real-world dataset.

In both simulation and real-world datasets, our mapping and localization solution provides accurate pose estimates. However, our automated pipeline finds it difficult to annotate long-term movable points due to the limit on map resolution. Our network prediction on the semantic classes shows that we can make accurate predictions on the ground and non-movable points yet misclassify some short-term and long-term movable points. We deem this as acceptable because we still manage to correctly annotate most of the points. Finally, we evaluate our entire pipeline by performing localization on semantically filtered LiDAR scans and compare its performance against the unfiltered approach.

In summary, the novel contribution of this thesis is a self-supervised semantic learning approach for large-scale scene understanding. We explicitly emphasize that we extend the original approach [53] to large-scale autonomous driving scenarios by:

- utilizing a robust LiDAR-only mapping and localization solution provided by Applanix Corporation
- correcting motion distortion for LiDAR point clouds when aggregating scans to map
- designing a voting strategy to extract ground points for large-scale point cloud map
- accounting for move-and-stop nature of on-road vehicles at traffic intersections
- labelling long-term movable points without carefully crafted geometric constraints

Furthermore, we provide a qualitative and quantitative evaluation of our approach on a simulation dataset using the Carla simulator and a real-world dataset using a Velodyne VLS-128 LiDAR sensor.

6.2 Future Work

We conclude this thesis with a discussion on future work to address multiple topics that arise during and after evaluating our self-supervised semantic learning pipeline.

6.2.1 Object-level Annotation

Recall our automated annotation pipeline. We annotate the map prior to transferring the annotations to the associated scans. We annotate points in the scans based on the annotation of their nearest neighbour in the map. Even though this is computationally appealing, points belonging to the same object may not get the same annotation as illustrated in Figure 5.1(a). A good example is a pedestrian on the sidewalk. Our annotation pipeline may annotate most of the points from the pedestrian as short-term movable, yet suggest his feet as ground points. Therefore, our future work should look into developing a self-supervised object-level annotation pipeline as opposed to the current point-level approach.

The object-level annotation pipeline would annotate points in LiDAR scans by ensuring points from the same object always receive the same annotation. To accomplish that, an intuitive approach is to segment scans into different objects, quantify the most frequent annotation in each object, and hard-assign this annotation to points from each object.

Finally, this object-level annotation pipeline would produce more accurate annotations, and thus have our network learn a more accurate semantic segmentation.

6.2.2 Runtime for Semantic Segmentation

Recall that our semantic segmentation network can be replaced with any existing segmentation network implementations. We choose to develop our network based on KPConv because of its state-of-the-art performance in completing semantic segmentation tasks in a large-scale environment. However, the runtime of KPConv makes it inapplicable for any real-time use. The current runtime is 2 Hz in contrast to our LiDAR logging rate at 10 Hz. Therefore, future work should look into replacing KPConv with a real-time semantic segmentation network to achieve fast inference.

An alternative semantic segmentation network is *RangeNet++* introduced by Milioto et al. [38]. The method reduces computational burden by using LiDAR range images as opposed to point clouds and achieves more than 20 times faster inference time than

a point cloud semantic segmentation network. It also achieves near to state-of-the-art semantic segmentation performance, which makes it more applicable for real-time use.

6.2.3 Annotation Update

Recall that we use an annotated map $\mathcal{M}^{(m)}$ as an intermediate representation to annotate scans in our annotation pipeline. The map is used to update annotations of all the localization scans. In other words, if there is any update to the map annotations, we need to update accordingly to all the localization scans. This is not an issue in our thesis because we only compute annotation for $\mathcal{M}^{(m)}$ once. However, when we look into improving our annotations frequently using newly collected localization sessions, it requires us to constantly update map annotations for $\mathcal{M}^{(m)}$. As a result, we have to update annotations for all the localization scans in the current pipeline regardless if it is necessary. This becomes a computationally expensive operation as it is linear to our map update frequency and the number of localization scans. Therefore, future work should look into improving the current pipeline to efficiently update scan annotations. An intuitive modification would be to update the scan annotation only if its associated map annotation has changed to reduce a lot of computation burden.

Appendix A

Computing Moving Probability for Map $\mathcal{M}$

Algorithm 1: Compute moving probability for map $\mathcal{M}$

Result: Map $\mathcal{M}$ with moving probability p_i for each x_i

initialize n, m s.t. each entry is zero;

for $\bar{P}_k$ *in all motion compensated scans* **do**

  for $\bar{P}_{k_i}$ *in* $\bar{P}_k$ **do**

    transform $\bar{P}_{k_i}$ to $\underrightarrow{\boldsymbol{\mathcal{F}}}_i$ using $\mathbf{T}_k$;

    $i = vox(\bar{P}_{k_i})$;

    $n_i = n_i + 1$;

    encode free space of $\bar{P}_k$ into frustum grid $\mathcal{I}_k$;

    project x_i into $\mathcal{I}_k$;

    if $r(x_i) <= \mathcal{I}_k(\theta_i, \phi_i)$ **then**

      $m_i = m_i + 1$

      $n_i = n_i + 1$

    else

      $m_i = m_i$

      $n_i = n_i$

    end

  end

end

for x_i *in* $\mathcal{M}$ **do**

  $p_i = \frac{m_i}{n_i}$

end

Bibliography

[1] Sean Anderson and Timothy D Barfoot. Full STEAM ahead: Exactly sparse gaussian process regression for batch continuous-time trajectory estimation on se (3). In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 157–164. IEEE, 2015.

[2] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2010.

[3] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5):698–700, 1987.

[4] Timothy D Barfoot. *State estimation for robotics*. Cambridge University Press, 2017.

[5] Ioan Andrei Barsan, Shenlong Wang, Andrei Pokrovsky, and Raquel Urtasun. Learning to localize using a lidar intensity map. *arXiv preprint arXiv:2012.10902*, 2020.

[6] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9297–9307, 2019.

[7] Jens Behley and Cyrill Stachniss. Efficient surfel-based slam using 3d laser range data in urban environments. In *Robotics: Science and Systems*, volume 2018, 2018.

[8] Paul J Besl and Neil D McKay. Method for registration of 3-D shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. International Society for Optics and Photonics, 1992.

[9] 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)*, volume 3, pages 2743–2748. IEEE, 2003.

[10] Keenan Burnett, Angela P Schoellig, and Timothy D Barfoot. Do we need to compensate for motion distortion and doppler effects in spinning radar navigation? *IEEE Robotics and Automation Letters*, 6(2):771–778, 2021.

[11] Xieyuanli Chen, Thomas Läbe, Andres Milioto, Timo Röhling, Olga Vysotska, Alexandre Haag, Jens Behley, and Cyrill Stachniss. Overlapnet: Loop closing for lidar-based slam. *arXiv preprint arXiv:2105.11344*, 2021.

[12] Xieyuanli Chen, Andres Milioto, Emanuele Palazzolo, Philippe Giguere, Jens Behley, and Cyrill Stachniss. Suma++: Efficient lidar-based semantic SLAM. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4530–4537. IEEE, 2019.

[13] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[14] Younggun Cho, Giseop Kim, and Ayoung Kim. Deeplo: Geometry-aware deep lidar odometry. *arXiv preprint arXiv:1902.10562*, 2019.

[15] Younggun Cho, Giseop Kim, and Ayoung Kim. Unsupervised geometry-aware deep lidar odometry. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2145–2152. IEEE, 2020.

[16] Blender Online Community. Blender-a 3d modelling and rendering package, 2017.

[17] Ayush Dewan, Gabriel L Oliveira, and Wolfram Burgard. Deep semantic classification for 3d lidar data. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3544–3549. IEEE, 2017.

[18] Hang Dong and Timothy D Barfoot. Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation. In *Field and Service Robotics*, pages 327–342. Springer, 2014.

[19] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[20] Renaud Dubé, Andrei Cramariuc, Daniel Dugas, Juan Nieto, Roland Siegwart, and Cesar Cadena. Segmap: 3d segment mapping using data-driven descriptors. *arXiv preprint arXiv:1804.09557*, 2018.

[21] Udo Frese, Per Larsson, and Tom Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, 2005.

[22] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[23] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2D lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[24] Michael Himmelsbach, Felix V Hundelshausen, and H-J Wuensche. Fast segmentation of 3d point clouds for ground vehicles. In *2010 IEEE Intelligent Vehicles Symposium*, pages 560–565. IEEE, 2010.

[25] Kin Leong Ho and Paul Newman. Loop closure detection in slam by combining visual and spatial appearance. *Robotics and Autonomous Systems*, 54(9):740–749, 2006.

[26] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[27] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4802–4809. IEEE, 2018.

[28] Jacek Komorowski. Minkloc3d: Point cloud based large-scale place recognition. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1790–1799, 2021.

[29] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[30] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[31] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-based precision vehicle localization in urban environments. In *Robotics: science and systems*, volume 4, page 1. Citeseer, 2007.

[32] Qing Li, Shaoyang Chen, Cheng Wang, Xin Li, Chenglu Wen, Ming Cheng, and Jonathan Li. Lo-net: Deep real-time lidar odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8473–8482, 2019.

[33] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4(10):1–3, 2004.

[34] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[35] Weixin Lu, Yao Zhou, Guowei Wan, Shenhua Hou, and Shiyu Song. L3-net: Towards learning based lidar localization for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6389–6398, 2019.

[36] Martin Magnusson, Andreas Nuchter, Christopher Lorken, Achim J Lilienthal, and Joachim Hertzberg. Evaluation of 3d registration reliability and speed-a comparison of icp and ndt. In *2009 IEEE International Conference on Robotics and Automation*, pages 3907–3912. IEEE, 2009.

[37] RS Merali, CH Tong, J Gammell, J Bakambu, E Dupuis, and TD Barfoot. Three-dimensional surface mapping using a semi-autonomous rover: A planetary analogue field experiment. In *Proc. of the 11th Int. Symp. on Artificial Intell., Robotics and Automation in Space (iSAIRAS)*, 2012.

[38] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4213–4220. IEEE, 2019.

[39] Mohammadreza Mostajabi, Payman Yadollahpour, and Gregory Shakhnarovich. Feedforward semantic segmentation with zoom-out features. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3376–3385, 2015.

[40] Kai Ni, Drew Steedly, and Frank Dellaert. Tectonic sam: Exact, out-of-core, submap-based slam. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1678–1685. IEEE, 2007.

[41] Edwin Olson, John Leonard, and Seth Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2262–2269. IEEE, 2006.

[42] Shishir Pagad, Divya Agarwal, Sathya Narayanan, Kasturi Rangan, Hyungjin Kim, and Ganesh Yalla. Robust method for removing dynamic objects from point clouds.

In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10765–10771. IEEE, 2020.

[43] Xuran Pan, Zhuofan Xia, Shiji Song, Li Erran Li, and Gao Huang. 3d object detection with pointformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7463–7472, 2021.

[44] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[45] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

[46] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586, 2017.

[47] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA, 2009.

[48] 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)*, pages 4758–4765. IEEE, 2018.

[49] Mark Sheehan, Alastair Harrison, and Paul Newman. Continuous vehicle localisation using sparse 3d sensing, kernelised rényi distance and fast gauss transforms. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 398–405. IEEE, 2013.

[50] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International journal of computer vision*, 81(1):2–23, 2009.

[51] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *2010 IEEE International Conference on Robotics and Automation*, pages 2657–2664. IEEE, 2010.

[52] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[53] Hugues Thomas, Ben Agro, Mona Gridseth, Jian Zhang, and Timothy D Barfoot. Self-supervised learning of lidar segmentation for autonomous indoor navigation. *arXiv preprint arXiv:2012.05897*, 2020.

[54] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

[55] Chi Hay Tong, Timothy D Barfoot, and Erick Dupuis. 3d slam for planetary worksite mapping. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 631–638. IEEE, 2011.

[56] Mikaela Angelina Uy and Gim Hee Lee. PointNetVLAD: Deep point cloud based retrieval for large-scale place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4470–4479, 2018.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[58] Guowei Wan, Xiaolong Yang, Renlan Cai, Hao Li, Yao Zhou, Hao Wang, and Shiyu Song. Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4670–4677. IEEE, 2018.

[59] Xinkai Wei, Ioan Andrei Bârsan, Shenlong Wang, Julieta Martinez, and Raquel Urtasun. Learning to localize through compressed binary maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10316–10324, 2019.

[60] Daniel Wilbers, Christian Merfels, and Cyrill Stachniss. Localization with sliding window factor graphs on third-party maps for automated driving. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5951–5957. IEEE, 2019.

[61] Huan Yin, Yue Wang, Xiaqing Ding, Li Tang, Shoudong Huang, and Rong Xiong. 3d lidar-based global localization using siamese neural network. *IEEE Transactions on Intelligent Transportation Systems*, 21(4):1380–1392, 2019.

[62] Keisuke Yoneda, Hossein Tehrani, Takashi Ogawa, Naohisa Hukuyama, and Seiichi Mita. Lidar scan feature for localization with highly precise 3-d map. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1345–1350. IEEE, 2014.

[63] David Yoon, Tim Tang, and Timothy Barfoot. Mapless online detection of dynamic objects in 3d lidar. In *2019 16th Conference on Computer and Robot Vision (CRV)*, pages 113–120. IEEE, 2019.

[64] David J Yoon, Haowei Zhang, Mona Gridseth, Hugues Thomas, and Timothy D Barfoot. Unsupervised learning of lidar features for use in a probabilistic trajectory estimator. *IEEE Robotics and Automation Letters*, 6(2):2130–2138, 2021.

[65] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737. IEEE, 2018.

[66] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.