Model-free Setting-Independent Detection of Dynamic Objects in 3D Lidar

by

Juny David Yoon

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

Abstract

Model-free Setting-Independent Detection of Dynamic Objects in 3D Lidar

Juny David Yoon
Master of Applied Science
University of Toronto Institute for Aerospace Studies
University of Toronto
2019

This thesis presents a model-free, setting-independent method for online detection of dynamic objects in 3D lidar data. We focus on a common type of 3D lidar, spinning-lidars, for which its moving-while-scanning operation must be compensated. Our detection method uses motion-compensated scan alignment and a novel freespace querying algorithm to classify between dynamic (currently moving) and static (currently stationary) labels at the point level.

A public dataset and benchmark suitable to evaluate our work did not previously exist due to the difficulty of accurately labelling groundtruth. For a quantitative evaluation, we establish a benchmark with motion-distorted lidar data using an open-source 3D simulator for autonomous driving research. We also provide a qualitative evaluation with real data using a commercially available lidar in driving scenarios. Results show that we can do a reasonable job of labelling each lidar point as either static or dynamic without resorting to a model or application-specific assumptions.

Acknowledgements

There are several people I give thanks to for the completion of this thesis, for their support in both my academic and personal life.

A very special thanks to my supervisor, Dr. Tim Barfoot, for his expert guidance.

To all the members of the Autonomous Space Robotics Lab, each of whom helped me in countless ways, I am truly grateful.

Finally, thanks to my family, my parents and older brother, for helping me through all the years of school that built up to the completion of this thesis.

Contents

List of Figures

Notation

a	:	Symbols in this font are real scalars.
$\mathbf{a}$	:	Symbols in this font are real column vectors.
$\mathbf{A}$	:	Symbols in this font are real matrices.
$\mathbf{1}$	:	The identity matrix.
$\mathbf{0}$	:	The zero matrix.
$\underrightarrow{\boldsymbol{\mathcal{F}}}_a$	:	A reference frame in three dimensions.
$\mathbf{T}_{a,b}$	:	The $SE(3)$ transformation that transforms vectors in homogeneous form from $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$ to $\underrightarrow{\boldsymbol{\mathcal{F}}}_a$.

Chapter 1

Introduction

1.1 Object Detection

At the current state of the art for autonomous mobile robotics research, we are able autonomously navigate with ease in stationary, predictable environments. A good example is visual teach and repeat, a navigation system that can reliably repeat long routes with a single on-board stereo camera (Furgale and Barfoot, 2010). However, in order to further enable autonomous navigation, an autonomous vehicle (or robot) must be able to reliably detect the dynamic aspects of its setting. While dynamic detection and perception is not as important in applications such as extraterrestrial rover navigation, the current trend in the industry is enabling autonomous driving, where there is a high abundance of moving objects (e.g., other vehicles and pedestrians).

Fortunately, detection is possible with the same suite of sensors used for egomotion estimation. Cameras are passive sensors that are widely used for their small form factor and being relatively inexpensive. A disadvantage to using cameras are their reliance on ambient lighting and the difficulty in estimating depth. In contrast, lidar (light detection and ranging) actively illuminates the scene and measures distance by observing the time it takes to reflect and return. Lidars are significantly more expensive than cameras, but

provide rich geometric data and are relatively unaffected by ambient lighting.

In this thesis we focus on the spinning-lidar configuration, which is currently the most common type of lidar that produces three-dimensional (3D) measurements. Several lasers are rigidly arranged along the vertical and are rotated continuously to produce 3D data over a limited vertical, but 360° horizontal field of view (FOV). Refer to Figure 2.2 in Section 2.1.1 for an example diagram.

Given data from a spinning-lidar, our goal is to detect which parts of the data belong to dynamic (moving) objects. One approach to dynamic object detection trains learning algorithms, such as deep neural networks (DNN), most commonly to detect objects of pre-defined class categories (e.g., pedestrians, vehicles, cyclists) (Chen et al., 2017). Given an accurate, reliable map, we can alternatively compare new data to the map and segment the discrepancies that are likely to be caused by dynamic objects (e.g., change detection) (Underwood et al., 2013).

We are more interested in a detection method that is independent of prior knowledge on the objects and the setting, including training data, making its use flexible to a larger variety of applications. Such a method does not have to be a direct competitor to methods that use prior knowledge, but can rather be a part of a larger suite of detection methods where it will act as a safety net (e.g., for learning-based detectors). Consider for example application in scenarios where a prior map is not available (e.g., a disaster zone), where methods that only rely on the latest sequences of data will have no issue detecting survivors.

1.2 Contributions

This thesis introduces a model-free, setting-independent dynamic object detection method for 3D lidar data at the point level. Detection involves labeling lidar points as dynamic (moving) or static (not moving) using only the most recent set of lidar scans. Due

to the difficulty in obtaining accurate groundtruth labels, we quantitatively analyze our method using motion-distorted lidar data simulated in CARLA (Dosovitskiy et al., 2017), an open-source simulator for autonomous driving research. We make this dataset publicly available for others to use (visit http://asrl.utias.utoronto.ca/datasets/mdlidar/index.html). We also provide a qualitative analysis using real data collected with a Velodyne HDL-64E in driving scenarios.

Specifically, the novel contributions are as follows:

- Model-free, setting-independent labelling of lidar points as static or dynamic.
- Dataset and benchmark of simulated 3D lidar data in on-road driving scenarios.
- Quantitative analysis of our work using the simulated benchmark.
- Qualitative analysis of our work using real lidar data collected in on-road driving scenarios.

1.3 High-level Overview

The remainder of this thesis is divided into the following chapters. Chapter 2 provides background knowledge on lidar data representations, lidar scan alignment, existing detection methods, and existing lidar datasets. Chapter 3 introduces the detection pipeline and discusses the individual components. Chapter 4 presents our lidar datasets, consisting of real data we collected with a Velodyne HDL-64E sensor, and simulated data using CARLA. Chapter 5 presents the quantitative and qualitative analyses, as well as the discussion of the results. Finally, Chapter 6 concludes the thesis and discusses future work.

Chapter 2

Background

We describe in this chapter background material and relevant literature. We begin with describing lidar data and its different representations, followed by a brief look into lidar scan alignment. We then discuss the existing literature on dynamic object detection methods, and finally end the chapter with a look at existing lidar datasets.

2.1 Lidar and Data Representation

Lidar operates by actively illuminating the scene and uses the time it takes for light to reflect and return to determine range. The amount of reflected light is also available as an intensity measurement.

This thesis focuses on the spinning-lidar configuration, which is currently the most common type of lidar in autonomous application for 3D measurements (see Figure 2.2). These lidars operate by continuously rotating a vertical arrangement of lasers to provide a 360° horizontal FOV. The horizontal resolution depends on how often the lasers take measurements and the rotation speed of the rotating laser hub. The vertical resolution and FOV depends on the number of lasers and the angular spacing along the vertical (i.e., elevation).

In this thesis, we refer to the accumulation of data from a finite amount of rotation

of the lidar (e.g., one revolution) as a *lidar scan*. We carefully distinguish the term, lidar scan, from the different representations of lidar data (e.g., pointcloud, imagespace, and freespace).

The Velodyne HDL-64E, a spinning-lidar with 64 lasers, first found prominent use during the 2007 Defense Advanced Research Projects Agency (DARPA) Urban Challenge, an autonomous driving challenge in an urban area course (Montemerlo et al., 2008; Bohren et al., 2008). To this day, Velodyne lidars are one of the most common 3D lidar sensor used in robotics. Currently, Velodyne offers lidar variants with 16, 32, 64, and 128 lasers. Competitors to Velodyne have emerged over recent years, with companies like Quanergy and Ouster providing similar spinning-lidars that produce 3D measurements with a 360° horizontal FOV.

2.1.1 Pointcloud

A pointcloud, as its name implies, is a collection of points formed from the endpoints of laser measurements of a lidar scan. Figure 2.1 shows an example pointcloud formed from one revolution of a Velodyne HDL-64E S3 at the University of Toronto Institute for Aerospace Studies campus.

The points (i.e., each element) of the pointcloud are computed from the raw bearing (encoder measurement of rotating laser hub) and range measurements of the lidar. Each bearing and range measurement pair corresponds to one laser, where each laser has a known unique position and orientation (i.e., pose) with respect to the base of the sensor. These individual laser pose values are given as calibration parameters.

We can explain this more clearly with reference frames. We introduce the lidar base frame, $\underrightarrow{\mathcal{F}}_b$, the laser hub frame, $\underrightarrow{\mathcal{F}}_h$, and individual laser frames, $\underrightarrow{\mathcal{F}}_\ell$, for each laser ℓ. The laser hub houses all the lasers and rotates with respect to the lidar base. We define $\underrightarrow{\mathcal{F}}_h$ to have a time-varying rotation about a single axis (e.g., z-axis) of $\underrightarrow{\mathcal{F}}_b$. We define each $\underrightarrow{\mathcal{F}}_\ell$ to measure range along a single axis (e.g., x-axis). Figure 2.2 illustrates the

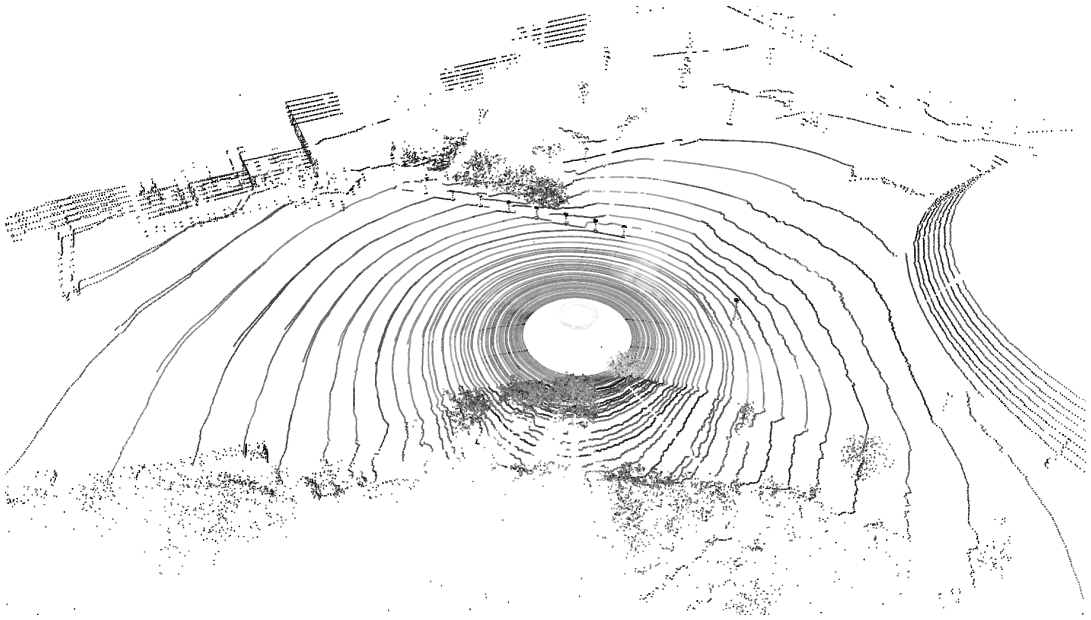


Figure 2.1: Example pointcloud taken from a Velodyne HDL-64E S3. 3D spinning-lidar sensors provide accurate geometric information at far ranges (up to 120 m).

reference frames for an example lidar with n lasers, $\underrightarrow{\boldsymbol{\mathcal{F}}}_{\ell_1}$, $\underrightarrow{\boldsymbol{\mathcal{F}}}_{\ell_2}$, ..., $\underrightarrow{\boldsymbol{\mathcal{F}}}_{\ell_n}$. The blue arrows indicate rigid transformations between the frames.

Atanacio-Jiménez et al. (2011) present a calibration method for Velodyne HDL-64E sensors using pattern planes, which includes accurately determining the unique poses of each laser. Consequently, their work is a good reference for converting raw range and bearing data to a pointcloud. Maddern et al. (2012a) present a calibration method for multiple generic 2D and 3D lidars by optimizing the quality of the resulting composite pointclouds.

The pointcloud is the most commonly used data product of 3D lidars and is often treated as an instantaneous snapshot of the scene. This is not true for spinning-lidars due to their moving-while-scanning mode of operation. Analogous to the rolling-shutter effect in certain cameras, the scene observed by the lidar will be distorted if it is mounted on a moving platform (e.g., robot or vehicle). We refer to this effect as motion distortion.

Compensation for motion distortion (i.e., accounting for the actual sensor pose for each measurement) is required for accurate construction of pointclouds on moving plat-

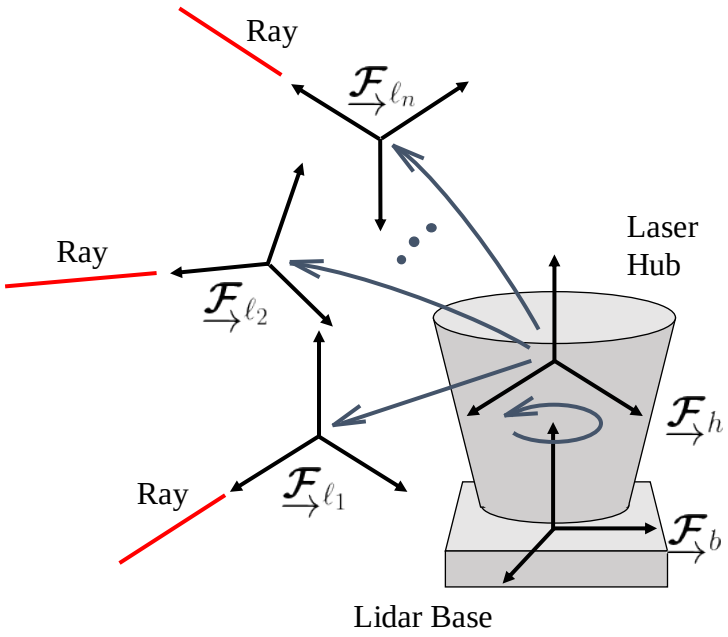


Figure 2.2: An illustration of the lidar reference frames involved in converting range and bearing measurements into 3D points for an example lidar with n lasers, $\underrightarrow{\mathcal{F}}_{\ell_1}$, $\underrightarrow{\mathcal{F}}_{\ell_2}$, ..., $\underrightarrow{\mathcal{F}}_{\ell_n}$. The laser hub frame, $\underrightarrow{\mathcal{F}}_{h}$, rotates with respect to the lidar base frame, $\underrightarrow{\mathcal{F}}_{b}$. The blue arrows indicate rigid transformations. Note the placement of the laser frames are not metrically to scale and should all be within the laser hub (grey).

forms. Past robotics research often experimented with terrestrial robots with movement speeds of at-most a few meters per second, for which a 10 Hz spinrate (common specification) is sufficiently fast enough to assume each revolution is a snapshot of the scene. The snapshot assumption loses validity as applications involve faster moving vehicles, such as vehicles for autonomous driving. Note that it is not correct to judge the amount of distortion by the spinrate of the lidar, rather it is the rate at which the lasers take measurements that is the limiting factor and should be taken into account. The configured spinrate is a result of considering the measurement-rate of the lidar and deciding on a desired horizontal (angular) resolution.

Figure 2.3 shows an example of a simulated lidar scan (one revolution) of a 10 Hz spinning-lidar moving at approximately 70 km/h. The top pointcloud is created from the scan by treating it as an instantaneous snapshot (i.e., all measurements share the same timestamp). The bottom pointcloud is created by compensating for the exact

timestamp of each measurement, each which has a corresponding lidar pose. The jagged discontinuity of the bottom pointcloud at the start and end of the sensor rotation shows the degree of distortion. The dimension of the shown square grid is 1 m, indicating approximately 2 m of distortion between the rotation start and end.

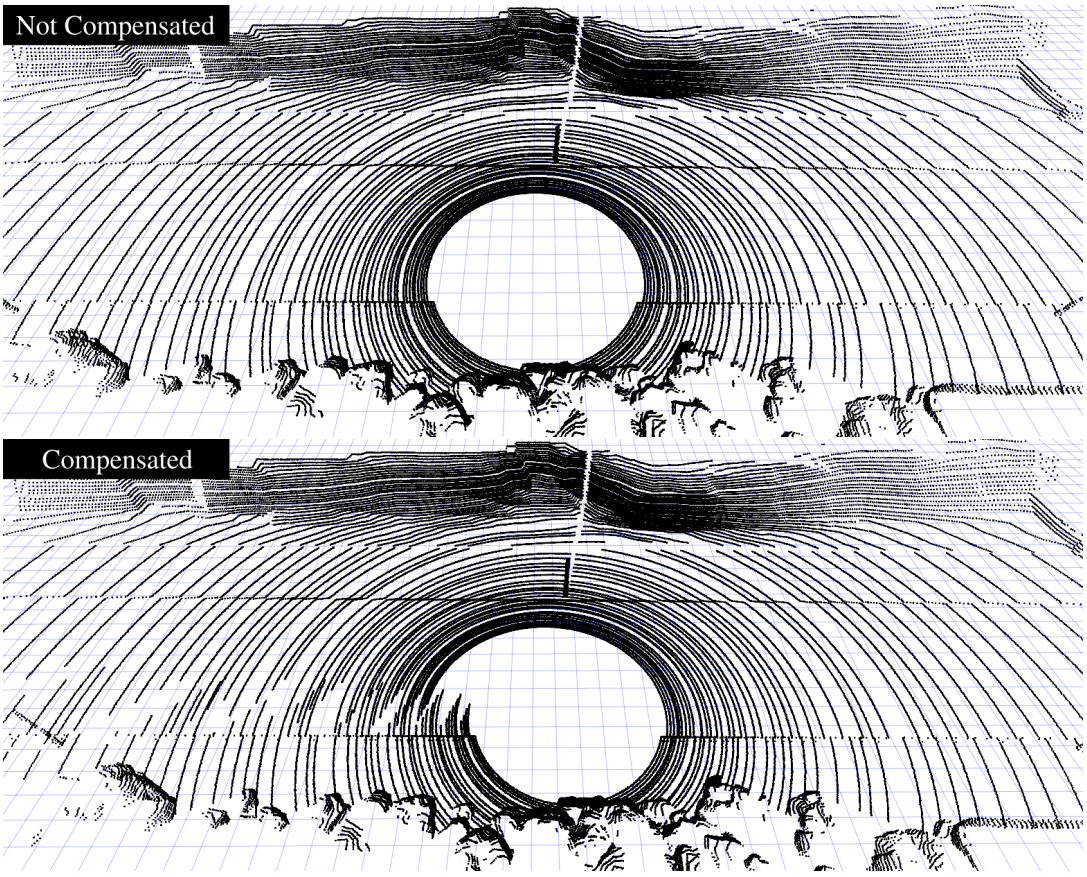


Figure 2.3: An example simulated lidar scan of a 10 Hz lidar moving at approximately 70 km/h. The bottom pointcloud has been compensated for motion, whereas the top has not. The motion distortion is most noticeable at the discontinuity of the start and end of the lidar rotation (9 o'clock position). The dimension of the shown grid (blue) is 1 m, showing approximately 2 m of distortion between the rotation start and end.

2.1.2 Imagespace

The ordering of laser measurements in a spinning-lidar can be used to represent the data in a form similar to a camera image. We refer to this as the imagespace representation. Each laser, ordered by its elevation angle, is an individual row of the image. The consec-

utive measurements of each laser are what becomes the entries along each row (forming columns). The rows can be viewed as the elevation and the columns as the azimuth of the lidar measurements.

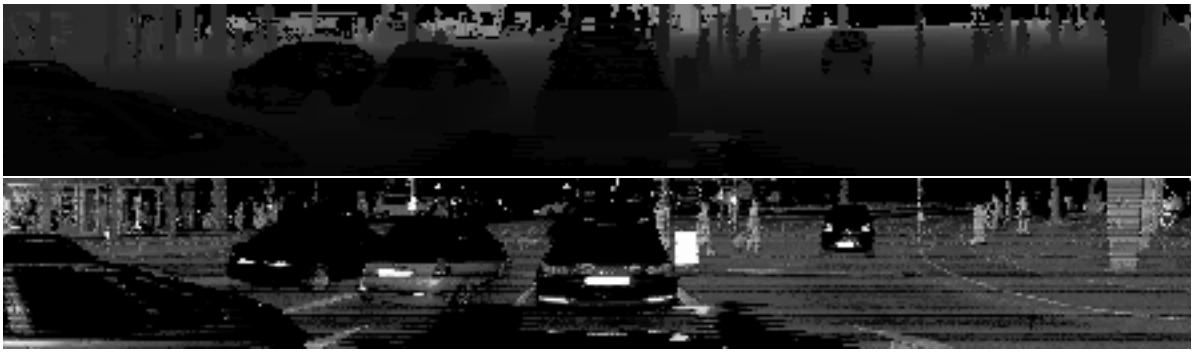


Figure 2.4: Example imagespace representations of Velodyne HDL-64E S2 lidar data from the publicly available dataset of Moosmann and Stiller (2013) in greyscale. The top image is coloured by range measurements and the bottom by intensity. Darker pixels indicate smaller values and lighter pixels indicate larger values. This example only shows a portion of the entire horizontal FOV in order to fit the page.

Depending on the hardware configuration of a lidar sensor, consecutive measurements of the lasers may not align perfectly with one another to form the columns of the imagespace. Uniform downsampling in the horizontal angular dimension may be required, which is what Moosmann and Stiller (2013) did for their Velodyne HDL-64E S2 dataset. They uniformly downsampled their data in the horizontal dimension to produce imagespace representations that are 64×870 in dimension. Greyscale examples are shown in Figure 2.4, where the top image has each pixel coloured by range measurements, and the bottom by intensity measurements. Darker pixels indicate smaller values and lighter pixels indicate larger values.

While pointclouds discard lidar measurement ordering, the imagespace representation takes full advantage of it. One example of benefiting from measurement ordering is how measurement neighbours in imagepsace can be used for efficient searching in Euclidean space for point neighbours, though it is limited to the scope of a single scan. Searching for point neighbours within a single scan is commonly done for surface normal estimation.

2.1.3 Freespace

The paths that measurements of a lidar scan trace in space define freespace. This aspect of lidar data is often ignored as many applications focus on processing pointclouds (i.e., the endpoints of measurement rays).

A common way of representing freespace is by discretizing the world into occupancy grids, or voxel grids in 3D. Raytracing algorithms, such as Bresenham's line algorithm (Bresenham, 1965), can associate lidar measurements into the discretized representation. The occupancy of each grid (or voxel) can then be modelled with uncertainty (Moravec, 1988). This method is advantageous for its efficient spatial query, which is an O(1) operation. Unfortunately, creating an occupancy grid is computationally expensive in both processing and in memory, especially for lidar sensors with far range limits. Octree data structure implementations, such as Octomap (Hornung et al., 2013) improve the memory requirements for 3D application, but the required computation is still considerable, increasing with smaller grid resolutions.

Alternatively, the freespace of a single lidar scan can be queried efficiently by determining the nearest laser ray of the scan for each query point (3D coordinate) of interest. From there it is a matter of checking if the scan ray goes past its corresponding query point, which indicates that the point may lie inside the scan's freespace. This is accomplished efficiently by building a search data structure (e.g., kd-tree) out of the spherical coordinates (azimuth and elevation, excluding range) from the scan's pointcloud (Pomerleau et al., 2014). The freespace can then be queried by converting the query points to spherical coordinates in the local frame of the scan, from which each query point is then matched to its nearest ray via a nearest neighbour search in spherical coordinates. While this method is much more efficient than occupancy voxel grids, it assumes that the pointclouds of lidar scans are instantaneous snapshots of the scene (i.e., no motion distortion). The method also approximates that all measurements of the pointcloud originate from a single point in space, the origin of the local reference frame of the pointcloud, which is

not true because each laser of the lidar can have a unique position and orientation.

2.2 Online Lidar Scan Alignment

A classic technique for aligning lidar scans is using the iterative closest point (ICP) algorithm (Besl and McKay, 1992), a method that works well for rigid pointcloud registration. ICP handles the data association problem between two pointclouds by iteratively re-associating data until convergence. Unfortunately, ICP assumes the pointclouds are rigid, which is not the case for pointclouds produced from spinning-lidar scans. An alternative to ICP is the normal distributions transform (NDT), which instead of points, uses combinations of normal distributions (Magnusson et al., 2007). Likewise to ICP however, NDT does not consider the motion distortion of lidar scans. Another alternative is using an entropy-based method, such as Renyi's Quadratic Entropy (RQE) (Maddern et al., 2012b), but once again, the technique was not designed to address motion distortion.

There has been work on lidar motion estimation that addresses the motion distortion by using continuous-time representations of trajectories. Zlot and Bosse (2014) represent their motion trajectory using splines and, similar to NDT, work with local statistical summaries of pointclouds called *surfels*. The current state of the art for lidar odometry is the work of Zhang and Singh (2014), which they call lidar odometry and mapping (LOAM). LOAM extracts geometric features (e.g., edges and planes) from pointcloud data to optimize a linearly-interpolated trajectory. LOAM currently ranks third in the KITTI odometry benchmark (Geiger et al., 2013). A variation of LOAM that also uses stereo camera data (VLOAM) (Zhang and Singh, 2015) ranks first.

A limitation to these continuous-time trajectory methods is the lack of a physically motivated motion prior, making them susceptible to incorrect scan alignment in settings with degenerate geometry (e.g., tunnels with few distinguishing geometric features). Anderson and Barfoot (2015) formulate a continuous-time trajectory estimation framework

with a physically motivated, constant velocity motion prior in $SE(3)$. We are able to use this framework for scan alignment of motion-distorted lidar data, which we have used in other works we have recently published (McGarey et al., 2018; Tang et al., 2018).

2.3 Existing Detection Methods

We place existing 3D lidar object detection methods into three categories: methods that use class-specific or model-based detectors, methods that use maps, and methods that only use the latest sequence of acquired data (live data).

Class-specific, or model-based, detectors take advantage of prior information of the objects to be detected. Petrovskaya and Thrun (2009) model vehicles as 2D bounding boxes, which they apply to 3D data by processing it into a 2D representation. Rather than manually crafting models, recent work focuses on learning methods. Chen et al. (2017), among many others (Ku et al., 2018; Zeng et al., 2018), take lidar data (some may also use camera data) as input to a deep neural network (DNN) and output class-specific detections at the object level in the form of bounding boxes. While this category is proven to work well, such methods will simply not detect objects for which they have not been trained.

Detection without prior object information is possible, which we refer to as being model-free, by comparing current data to a reliable prior map. Given a reliable map of the stationary world, differences from the comparison are indicative of dynamic objects. Sometimes called *change detection* (Hebel et al., 2011; Underwood et al., 2013), these methods make use of pointcloud comparisons (i.e., end-points of lidar measurements) and freespace comparisons (i.e., paths traced by lidar measurements). Hebel et al. (2011) ray-trace lidar data into occupancy voxel grids for their freespace representation. Occupancy voxel grids are expensive computationally and in memory, so instead, Pomerleau et al. (2014) query lidar freespace by matching measurements with local spherical coordinates.

The method is efficient, but assumes pointclouds are not motion distorted. Note that while existing works that use occupancy voxel grids do not consider motion distortion, compensation is trivial with a continuous-time trajectory.

Our interest is in methods that do not require prior information on the objects or the setting, only making use of live data. Unfortunately, such methods are limited to detecting objects that are moving in the current scene. Objects that are stationary, but may be of interest (e.g., stationary cars in traffic), are not detectable.

Among live data methods are ones that only use pointcloud information. Dewan et al. (2016a) compare subsequent pointclouds and sequentially identify motion through a voting scheme. The first detected motion will always be the relative motion of the stationary environment, followed by the largest dynamic objects. They directly compared their work to Moosmann and Stiller (2013) and showed superior performance at the object level. In another publication, Dewan et al. (2016b) produce scene flow (i.e., point-wise velocity estimation), from which dynamic labels are trivial. However, both of their methods are not setting-independent because they rely on removing ground points as a pre-processing step. Live data methods are challenging because there are significant differences between subsequent pointcloud comparisons due to viewpoint occlusions and spatial data sparsity. In contrast, occlusions and data sparsity are not an issue when comparing live data to a map. Removing ground points is helpful in avoiding false detections, but it is not clear how to handle occlusions with only pointclouds.

Live data methods that use freespace handle viewpoint occlusions well. Azim and Aycard (2012) raytrace over occupancy voxel grids and compare voxels over subsequent scans, but only provide a qualitative analysis of their method. Postica et al. (2016) also make comparisons with occupancy voxel grids. They present quantitative results using short sequences from the KITTI dataset Geiger et al. (2013), for which they have manually annotated with groundtruth labels, but have not made public. Notable limitations of their work include relying on pre-processing ground points and ignoring measurements

further than 30 m, limiting their detection range.

The three categories are complementary to one another, so a combination can be more effective. Moosmann and Stiller (2013) segment pointclouds into object proposals, which they track over time. Consistent ones are labelled dynamic by a learned classifier. Ushani et al. (2017) use freespace and learning to compute scene flow. Occupancy voxel grids coarsely identify dynamic points, which are then refined by a learned classifier. They make a planar motion assumption and limit their method to a 50 m $\times$ 50 m grid. Dewan et al. (2017) combine their prior work on scene flow (Dewan et al., 2016b) with a DNN to produce point labels of dynamic, static, and a third label for objects with the potential to move (e.g., stationary cars).

The work we present in this thesis belongs to the live data category and is model-free, labelling each point as dynamic (currently moving) or static (currently not moving). Our detection method is unique because we compensate for motion distortion for both pointclouds and freespace querying, which existing methods do not consider. We make full use of the sensor range, which is often limited for methods that use freespace. Our detection is setting independent. We do not make prior assumptions of the setting, such as exploiting the knowledge of the gravity vector for the existence of a ground plane (isotropic). If setting independence is not an application importance, our method can be combined with methods from the other categories for greater performance and robustness.

2.4 Existing Datasets and Benchmarks

A lidar dataset labelled with accurate groundtruth is required to quantitatively evaluate the dynamic object detection method presented in this thesis and compare it to existing ones. The dataset must provide the raw motion-distorted data (e.g., not compensated by other sensors). There should be an abundance of dynamic objects in the scene, with ideally, accurate groundtruth labels at the point level. There should also be an abundance

of static obstruction in varying shapes for viewpoint occlusions (i.e., the setting should not be an open field with only moving objects).

Several lidar datasets exist for the purpose of motion estimation (e.g., odometry, localization). Some examples are the Ford Campus (Pandey et al., 2011), University of Michigan North Campus (Carlevaris-Bianco et al., 2016), and Complex Urban (Jeong et al., 2018) datasets. While these datasets are abundant in dynamic objects and static obstruction, they are unfortunately not labelled with detection groundtruth. This is often the case because annotating lidar data is a difficult and tedious manual task.

Most object detection methods evaluate against the KITTI 3D Object Detection Evaluation benchmark (Geiger et al., 2013), where objects of specific classes (vehicles, pedestrians, and cyclists) are annotated at the object level with bounding boxes. In this dataset, lidar data was collected using a Velodyne HDL-64E S2. Apart from the absence of point-level labels, a significant issue with this dataset for our use case is that only pointcloud data is provided. The dataset is missing the raw lidar data (i.e., range and bearing measurements) and laser calibration values (i.e., values related to laser positions and orientations). As described in Section 2.1.1, each laser of spinning-lidar sensors has a unique position and orientation. For computing freespace, simply computing the spherical coordinates of a pointcloud (i.e., azimuth and elevation) in its local reference frame is not equivalent to the true laser ray paths.

More recently, Roynard et al. (2018) made available the Paris-Lille-3D lidar dataset, which has a Velodyne HDL-32E mounted on a vehicle driven through an urban setting. They manually labelled their dataset at the point-level for a variety of objects, dynamic and static. Unfortunately, similar to the KITTI dataset, they only provide processed points and not the raw lidar data. Their pointclouds are motion compensated and measurements with range greater than 20 m were removed, which is significantly lower than the sensor's maximum range of 100 m.

Existing work that is comparable to the work presented in this thesis either omit

quantitative results or use short sequences of manually labelled data, which they do not make public. The lack of a suitable dataset and benchmark is a concern that needs to be resolved. For the purpose of quantitative evaluation, we look to using simulated lidar data. While simulated data is not an ideal replacement for real data, it is suitable as a comparison benchmark between different methods given that there is enough fidelity in the simulation. An example simulator is CARLA, an open-source simulator for autonomous driving research (Dosovitskiy et al., 2017). We detail our efforts in generating lidar data with accurate groundtruth using CARLA, including modifications to suite our interests, in Chapter 4. Also discussed in Chapter 4 is the setup of our data perception vehicle, which has mounted a Velodyne HDL-64E. We use datasets collected with this setup for a qualitative evaluation.

2.5 Summary

In summary, this chapter describes background material related to the dynamic object detection problem in 3D lidar data. We describe the different representations of lidar data with examples. A brief literature review on lidar scan alignment is presented where we highlight our ability to accomplish motion-compensated pointcloud registration with a physically-motivated motion prior. A detailed literature review on existing detection methods is provided, where we also distinguish the work presented in this thesis. The chapter ends with a review on existing datasets, where we establish the lack of a sufficient dataset and benchmark for evaluation and comparison of our work to others. Our resolution to this issue is to work with simulated data for a quantitative evaluation and real data for a qualitative evaluation.

Chapter 3

Methodology

In this chapter, we introduce our detection pipeline and describe the methodology of each component. The general idea behind our detection is to rely on discrepancies between subsequent lidar scans to identify dynamic objects without prior knowledge on object types or the setting.

We begin with a brief overview of our pipeline, then discuss each pipeline component in more detail. We introduce our lidar scan alignment algorithm that compensates for motion distortion. We discuss trade-offs when applying comparisons between subsequent lidar scans (pointclouds). We introduce a novel freespace query algorithm that compensates for the moving-while-scanning operation of spinning-lidars. We describe an imagespace filtering technique to reduce spurious labels. Finally, we outline our approach to region-growing existing object detections, which we will see is a requirement to reliably detect the entirety of dynamic objects.

From this point on, we explicitly refer to a *lidar scan* as the accumulation of measurements over a *single revolution* of a spinning-lidar. In contrast, pointclouds can consist of data from multiple lidar scans.

3.1 Pipeline Formulation

In this section, we define our dynamic object detection pipeline. A pipeline diagram is shown in Figure 3.1, which labels points as dynamic or static on a single scan of interest, the *query scan*. The sequential steps to the pipeline are explained as follows:

1. *Odometry*: Align the latest lidar scan to existing ones.

2. *Pointcloud Comparison*: Comparison of query scan against a reference. Discrepancies are set to dynamic.

3. *Freespace Check*: Check dynamic points of query scan against freespace of other reference scans. Points not in freespace are not dynamic and changed to static.

4. *Box Filter*: Apply a sliding box filter on the image representation of the query scan for outlier rejection.

5. *Region Growth*: Cluster the dynamic query scan points. Add nearby points to clusters if they satisfy conditions that indicate they are part of the same object.

The numbered steps correspond to the pipeline blocks in Figure 3.1. In the top-right of the figure is an example diagram indicating which lidar scans are in use to compute the detection of one query scan. The diagram indicates a gap of 4 scans between the query and reference scans. Figure 3.2 shows a portion of a pointcloud with a moving vehicle as it passes through the detection pipeline. The letters (a) to (d) correspond to the letters in Figure 3.1.

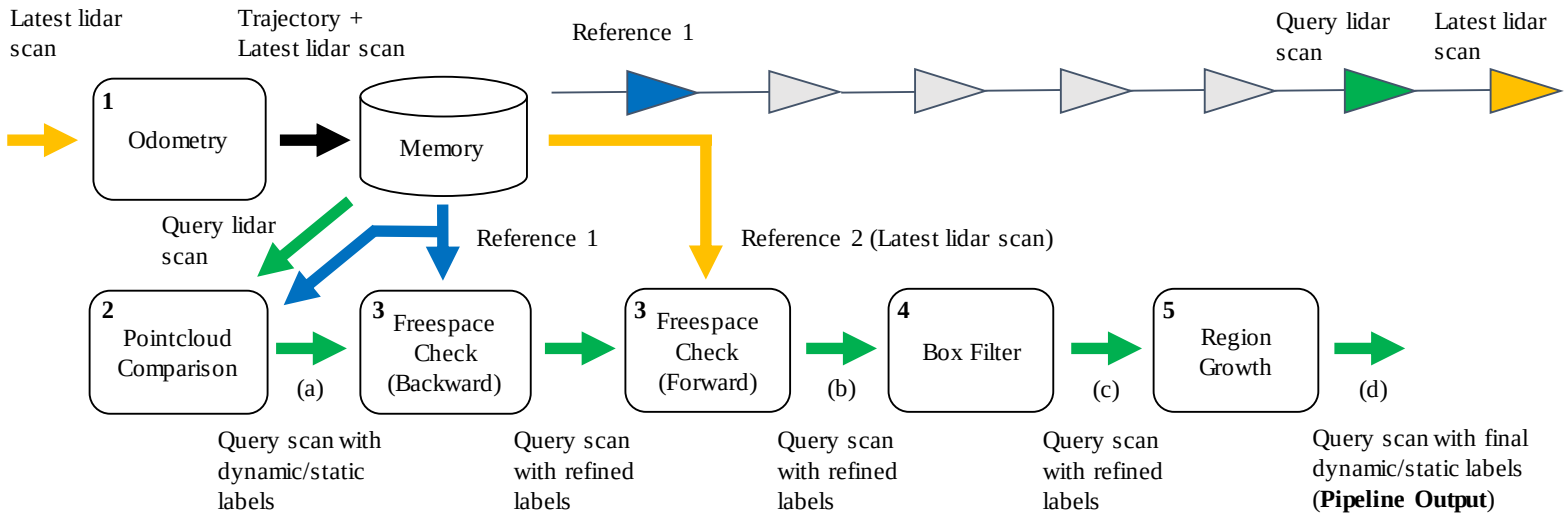
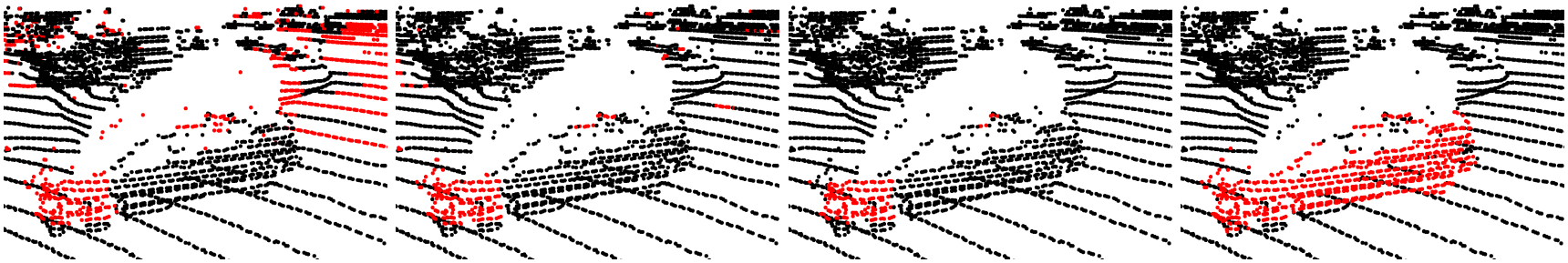


Figure 3.1: The full detection pipeline, describing the sequence of operations on the query lidar scan, outputting the scan with points labelled dynamic or static. A lidar odometry algorithm computes the sensor trajectory, which aligns the latest lidar. The labels (a) to (d) correspond to the images in Figure 3.2. The numbers correspond to the enumerated steps in the text.



(a) After pointcloud comparison. (b) After freespace check. (c) After box filter. (d) After region growth.

Figure 3.2: Pointcloud examples of a vehicle throughout the detection pipeline from a Velodyne HDL-64E. Dynamic labels are shown as red and static labels as black. Refer to the pipeline in Figure 3.1 for corresponding letters (a) to (d).

3.2 Lidar Trajectory and Scan Alignment

We require a scan alignment algorithm to provide metric-accurate alignment of incoming lidar scans (i.e., the *latest scan*), which in consequence produces a trajectory of the moving platform the sensor is mounted on. In other words, we require an online lidar odometry algorithm. In this section, we provide a brief overview of a lidar odometry algorithm that uses the continuous-time trajectory estimation framework of Anderson and Barfoot (2015), which uses a physically motivated motion prior. We formulate the problem as an optimization to align the latest lidar scan to existing ones.

3.2.1 Trajectory Representation and Motion Prior

Anderson and Barfoot (2015) present a continuous-time trajectory estimation framework they refer to as *simultaneous trajectory estimation and mapping* (STEAM), a variation of *simultaneous localization and mapping* (SLAM) that, rather than estimating for robot states and landmarks at discrete times, outputs a continuous-time Markovian trajectory:

$$\mathbf{x}(t) = \{\mathbf{T}(t), \boldsymbol{\varpi}(t)\} \in SE(3) \times \mathbb{R}^6, \tag{3.1}$$

where $\mathbf{T} \in SE(3)$ is the robot pose and $\boldsymbol{\varpi} \in \mathbb{R}^6$ is the body-centric generalized velocity.

In our use case, we are not concerned about estimating landmarks and only interested in the trajectory. The underlying representation is still a discrete collection of robot states,

$$\mathbf{x} = \{\mathbf{T}_{1,0}, \boldsymbol{\varpi}_1, \mathbf{T}_{2,0}, \boldsymbol{\varpi}_2, \ldots, \mathbf{T}_{k,0}, \boldsymbol{\varpi}_k\}, \quad \underrightarrow{\boldsymbol{\mathcal{F}}}_k = \underrightarrow{\boldsymbol{\mathcal{F}}}_v(t_k), \tag{3.2}$$

where $\underrightarrow{\boldsymbol{\mathcal{F}}}_v(t_k)$ is the reference frame of the robot at time t_k and $\underrightarrow{\boldsymbol{\mathcal{F}}}_0$ can be considered as the world reference frame.

The estimation problem is formulated as a Gaussian Process (GP) regression (Rasmussen and Williams, 2006) with time as the independent variable. The physically moti-

vated GP prior (motion model) that Anderson and Barfoot use is a nonlinear stochastic differential equation (SDE) of the form

$$\dot{\mathbf{T}}(t) = \boldsymbol{\varpi}(t)^{\wedge}\mathbf{T}(t), \dot{\boldsymbol{\varpi}} = \mathbf{w}(t), \quad \mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_C\delta(t-t')), \tag{3.3}$$

where the process noise, $\mathbf{w}(t)$, is a zero-mean GP with power-spectral-density matrix, $\mathbf{Q}_C$, $\delta(t)$ is the Dirac delta function, and the operator $\wedge$ turns a vector in $\mathbb{R}^6$ into a 4×4 member of the Lie algebra, $\mathfrak{se}(3)$ (see Appendix A for the definition). In other words, it is a constant body-centric velocity motion prior.

The nonlinear SDE is approximated using a piecewise, locally linear SDE between the discrete states for an efficient solution. We do not delve into the details of the derivation in this thesis, but the importance of this method in our work is two-fold:

1. Binary smoothing terms between temporally adjacent states.

2. GP interpolation equations for $\mathbf{T}(t)$ and $\boldsymbol{\varpi}(t)$ at any time, t.

The binary smoothing terms between temporally adjacent states at times t_k and t_{k-1} are defined as

$$\mathbf{e}_{u_k} = \begin{bmatrix} \ln\left(\mathbf{T}_{k,0}\mathbf{T}_{k-1,0}^{-1}\right)^{\vee} - (t_k - t_{k-1})\boldsymbol{\varpi}_{k-1} \\ \boldsymbol{\mathcal{J}}\left(\ln\left(\mathbf{T}_{k,0}\mathbf{T}_{k-1,0}^{-1}\right)^{\vee}\right)^{-1}\boldsymbol{\varpi}_k - \boldsymbol{\varpi}_{k-1} \end{bmatrix}, \tag{3.4}$$

where $\ln(\cdot)$ is the inverse of the $SE(3)$ exponential map, $\boldsymbol{\mathcal{J}}$ is the left Jacobian of $SE(3)$ and $\vee$ is the inverse of $\wedge$ (see Appendix A for the definitions). These terms are weighted with the inverse covariance expression

$$\mathbf{Q}_k^{-1} = \begin{bmatrix} 12\triangle t_k^{-3}\mathbf{Q}_C^{-1} & -6\triangle t_k^{-2}\mathbf{Q}_C^{-1} \\ -6\triangle t_k^{-2}\mathbf{Q}_C^{-1} & 4\triangle t_k^{-1}\mathbf{Q}_C^{-1} \end{bmatrix}, \tag{3.5}$$

where $\triangle t_k := t_k - t_{k-1}$.

From the GP linear interpolation equation of Rasmussen and Williams (2006), An-

derson and Barfoot use their piecewise motion prior to form the following interpolation equations:

$$\mathbf{T}(\tau) = \exp\left(\left(\boldsymbol{\Lambda}_1(\tau)\boldsymbol{\gamma}_{t_{k-1}}(t_{k-1}) + \boldsymbol{\Omega}_1(\tau)\boldsymbol{\gamma}_{t_{k-1}}(t_k)\right)^{\wedge}\right)\mathbf{T}_{k-1,0}, \tag{3.6}$$

$$\boldsymbol{\varpi}(\tau) = \boldsymbol{\mathcal{J}}\left(\ln\left(\mathbf{T}(\tau)\mathbf{T}_{k-1,0}^{-1}\right)^{\vee}\right)\left(\boldsymbol{\Lambda}_2(\tau)\boldsymbol{\gamma}_{t_{k-1}}(t_{k-1}) + \boldsymbol{\Omega}_2(\tau)\boldsymbol{\gamma}_{t_{k-1}}(t_k)\right), \tag{3.7}$$

where $t_{k-1} \leq \tau \leq t_k$, $\exp(\cdot)$ is the exponential map, $\boldsymbol{\Lambda}(\tau) = [\boldsymbol{\Lambda}_1(\tau)\,\boldsymbol{\Lambda}_2(\tau)]^T$, $\boldsymbol{\Omega}(\tau) = [\boldsymbol{\Omega}_1(\tau)\,\boldsymbol{\Omega}_2(\tau)]^T$, and

$$\boldsymbol{\gamma}_{t_{k-1}}(t_{k-1}) = \begin{bmatrix} \mathbf{0} \\ \boldsymbol{\varpi}(t_{k-1}) \end{bmatrix}, \quad \boldsymbol{\gamma}_{t_{k-1}}(t_k) = \begin{bmatrix} \ln\left(\mathbf{T}_{k,0}\mathbf{T}_{k-1,0}^{-1}\right)^{\vee} \\ \boldsymbol{\mathcal{J}}\left(\ln\left(\mathbf{T}_{k,0}\mathbf{T}_{k-1,0}^{-1}\right)^{\vee}\right)^{-1}\boldsymbol{\varpi}_k \end{bmatrix}. \tag{3.8}$$

Definitions of the time-dependant coefficient matrices, $\boldsymbol{\Lambda}(\tau)$ and $\boldsymbol{\Omega}(\tau)$, are provided in Appendix A. Note that $\boldsymbol{\gamma}_{t_{k-1}}(t)$ is a vector representation of the trajectory local to $SE(3)$ transformation $\mathbf{T}_{k-1,0}$. Anderson and Barfoot apply their piecewise prior in the local vectorspace of each discrete state pose, which is essentially the sparse vectorspace GP regression estimation of Barfoot et al. (2014).

The significance of the interpolation equations for the pose, $\mathbf{T}(t)$, and velocity, $\boldsymbol{\varpi}(t)$, at any time, t, is the O(1) efficiency of the operation. This is due to how Anderson and Barfoot (2015) take advantage of the Markovian state trajectory to exploit sparsity in the GP regression formulation, resulting in interpolation equations that only involve two temporally adjacent discrete states.

3.2.2 Measurement Term

We define a measurement term for the scan alignment optimization based on the converted 3D points (i.e., pointcloud) of lidar scans (recall that raw lidar measurements are the range and bearing values). Although this method of data handling is ad hoc

compared to working with the raw data, it has proven to work well for scan alignment algorithms and has become a standard (Pomerleau et al., 2015).

Let $\mathbf{q}_{t_i}^i \in \mathbb{R}^3$ be the ith point from the latest scan that we wish to align, acquired at measurement time, t_i, and expressed in the time-varying robot reference frame $\underrightarrow{\mathcal{F}}_v(t_i)$. Assuming the data association is known, let $\mathbf{p}_0^i \in \mathbb{R}^3$ be a point from a known reference pointcloud that is associated to $\mathbf{q}_{t_i}^i$, expressed in the world reference frame, $\underrightarrow{\mathcal{F}}_0$.

A point-to-point error term between $\mathbf{q}_{t_i}^i$ and its reference, $\mathbf{p}_i$, for the scan alignment optimization is defined as the euclidean distance between the two:

$$\mathbf{e}_{m_i} = \mathbf{B}\left(\begin{bmatrix}\mathbf{q}_{t_i}^i \\ 1\end{bmatrix} - \mathbf{T}_{t_i,0}\begin{bmatrix}\mathbf{p}_0^i \\ 1\end{bmatrix}\right), \quad \mathbf{B} = \begin{bmatrix}1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0\end{bmatrix}, \tag{3.9}$$

where $\mathbf{B}$ is a constant projection matrix and $\mathbf{T}_{t_i,0} := \mathbf{T}(t_i)$, which is known from the trajectory interpolation scheme in Equation (3.6).

These error terms are weighted with an inverse covariance expression $\mathbf{M}_i^{-1} \in \mathbb{R}^{3\times3}$, which inevitably is a tuning parameter as it corresponds to a 3D point, not the raw measurement. Alternatively, we can compute an estimate of the normal vector of the surface that $\mathbf{q}_0^i$ resides on, $\mathbf{n}^i$, to formulate a point-to-plane error term. This is done by weighting the error term, $\mathbf{e}_{m_i}$, with $\mathbf{M}_i^{-1} = \mathbf{n}^i\mathbf{n}^{iT}$ (Pomerleau et al., 2015). Surface normals can be computed by evaluating the sample covariance of the interest point and its neighbours (e.g., a constant radius away). The eigenvector corresponding to the smallest eigenvalue of the sample covariance is the surface normal.

3.2.3 Single-scan Lidar Odometry

Here we formulate a single-scan alignment optimization for the latest lidar scan, which is equivalently a lidar odometry algorithm.

Recall the discrete trajectory representation in Equation (3.1). By design choice, we

create a new discrete state (i.e., pose and velocity) at the latest measurement time of the scan we wish to align. With each new lidar scan, we optimize one discrete state, $\mathbf{x}_k = \{\mathbf{T}_{k,0}, \boldsymbol{\varpi}_k\}$. For the motion prior smoothing term in Equation (3.4), we use the posterior estimate of the temporally previous discrete state, $\hat{\mathbf{x}}_{k-1} = \left\{\hat{\mathbf{T}}_{k-1,0}, \hat{\boldsymbol{\varpi}}_{k-1}\right\}$.

The latest scan consists of m points obtained at times $t_i, i = 1 \dots m$, where $t_{k-1} < t_i \leq t_k$. We formulate the scan alignment problem as a nonlinear least squares optimization:

$$\hat{\mathbf{x}}_k = \arg\min_{\mathbf{x}_k} \frac{1}{2} \left(\mathbf{e}_{u_k}(\mathbf{x}_k)^T \mathbf{Q}_k^{-1} \mathbf{e}_{u_k}(\mathbf{x}_k) + \sum_{i=1}^{m} \mathbf{e}_{m_i}(\mathbf{x}_k)^T \mathbf{M}_k^{-1} \mathbf{e}_{m_i}(\mathbf{x}_k) \right). \tag{3.10}$$

Equation (3.10) is optimized using the Gauss-Newton algorithm. The error terms, $\mathbf{e}_{u_k}$ and $\mathbf{e}_{m_i}$, are linearized with respect to $\mathbf{x}_k$ using the constraint-sensitive $SE(3)$ perturbation scheme of Barfoot and Furgale (2014). Optionally, a robust cost function (i.e., M-estimation) can be applied to the measurement terms to eliminate sensitivity to outlier point associations (Barfoot, 2017).

Equation (3.10) was formulated with the reference points, $\mathbf{q}^i$, corresponding to each scan point, $\mathbf{p}^i_{t_k}$, as known. A simple method known to work well to solve the data association problem is to employ the iterative nearest neighbour approach, similar to ICP. This means we identify the nearest neighbour of each scan point, $\mathbf{p}^i_{t_k}$, in the reference pointcloud as our reference points, $\mathbf{q}^i$, and optimize Equation (3.10), repeating the two steps until convergence.

The scan alignment algorithm we present here is for a single lidar scan, optimizing a single discrete time state, $\mathbf{x}_k = \{\mathbf{T}_{k,0}, \boldsymbol{\varpi}_k\}$. The discrete state is a single element of a larger set of states that is the underlying representation of a continuous-time trajectory. It is trivial to expand the method to more states and lidar scans, where the batch estimation variation is what Anderson and Barfoot (2015) originally formulated. We applied our scan alignment method in published works (McGarey et al., 2018; Tang et al., 2018).

3.3 Pointcloud Comparison

We compute a pointcloud comparison using the motion-compensated alignment of lidar scans from the previous section. The pointcloud of the lidar scan of interest, the *query scan* (in this case, equivalently called the *query pointcloud*), is compared to a *reference pointcloud*. The reference pointcloud can consist of one or more lidar scans. Refer to the lidar scan diagram in the top-right corner of Figure 3.1, where it depicts the query scan (green triangle) and reference pointcloud consisting of one scan (blue triangle) in the past. The reference pointcloud consists of scans from slightly different periods of time compared to the query, so dynamic objects will cause discrepancies in the comparison. This is the basis of how we achieve a model-free and setting-independant detection method, where the pointcloud comparison will give the initial classification between the static and dynamic labels. Note that the query scan is not necessarily the latest scan (i.e., the most recent aligned scan).

Given the metrically aligned query pointcloud and reference pointcloud, the pointcloud comparison algorithm is as follows:

1. Step through all query points.
2. Determine the nearest neighbour of the current query point in the reference pointcloud.
3. Compute the *error metric* between the query and reference points.
    - If the error metric is greater than a threshold, λ_{error}, label the query point as dynamic.
    - Otherwise, label the query point as static.

The rest of this section defines how we compute the error metric and discusses trade-offs that must be taken into account. These trade-offs involve choosing the appropriate *scan gap* between the query pointcloud and reference, selecting the number of scans to

include in the reference pointcloud, deciding on the error metric type, and pointcloud comparisons to previous and/or later scans. We end this section with a discussion on the limitation of pointcloud comparisons for detecting dynamic objects.

3.3.1 Error Metric

Identifying discrepancies when comparing pointclouds is accomplished by using similar measurement error metrics (i.e., point-to-point and point-to-plane) used for the scan alignment problem in Section 3.2.2. If the query pointcloud matches well with its reference pointcloud, the residual error after alignment should be low (i.e., near zero). Portions of the pointcloud with high error are indicative of a discrepancy, which may have been caused by a dynamic object. The error metric is computed for all points of the query pointcloud.

We use a point-to-plane metric when points have sufficient neighbours to compute surface normals. Otherwise, we use a point-to-point metric. All points are transformed to $\underrightarrow{\mathcal{F}}_0$ using our continuous-time trajectory, allowing us to work with motion-compensated pointclouds. Given a query point, $\mathbf{q}_0$, its unit surface normal, $\mathbf{n}^q$, and its nearest reference scan neighbour, $\mathbf{p}_0$, the point-to-plane metric is $|\mathbf{n}^q \cdot (\mathbf{p}_0 - \mathbf{q}_0)|$. The point-to-point metric is $\|\mathbf{p}_0 - \mathbf{q}_0\|_2$.

As mentioned previously, we expect static points to have low error because there should be a corresponding reference point of the same surface observation. We expect high error from dynamic points since they are observations of moving surfaces (i.e., discrepancies). We take a constant scalar *error threshold*, λ_{error}. Error metrics greater than the λ_{error} are labelled dynamic, otherwise they are labelled static. Figure 3.3 shows an example outcome of a pointcloud comparison before taking the error threshold. The shown pointcloud is a query pointcloud where each point is coloured from black (low error) to red (high error) according to its scalar error metric against a reference pointcloud (not shown).

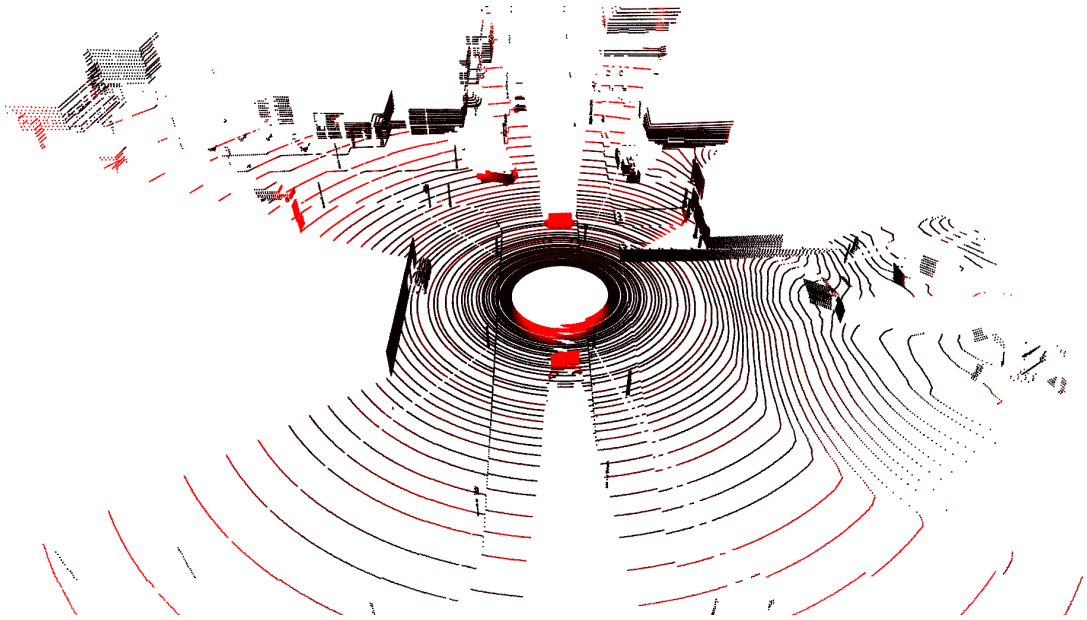


Figure 3.3: Example query pointcloud from simulated data, where each point is coloured by its pointcloud comparison error metric. Black indicates low error and red indicates high error. Notice the difficulty in discerning the dynamic objects as many parts of the static scene also have high error.

3.3.2 Comparison Trade-offs

There are various trade-offs to consider, such as parameter selection, when implementing the pointcloud comparison method. We discuss these trade-offs in this subsection.

Error Threshold

The error threshold, λ_{error}, is the parameter that determines whether a point will be labelled as static or dynamic during the pointcloud comparison. Fortunately, this parameter has a realworld meaning, as the error metric is the metric distance of a given point to its reference point or surface.

In an ideal scenario, λ_{error} should simply have a value of 0 so that all points with non-zero error are considered dynamic (recall that errors greater than λ_{error} are labelled dynamic). Realistically, we have to account for sensor resolution (i.e., the discretization of the realworld by the sensor) and noise. The source of noise for lidars is primarily from range measurements. The encoder measurements of commercially available spinning-

lidars, which determine the orientation of the lasers, are accurate enough to be negligible. Therefore we must account for sensor resolution and noise by setting λ_{error} to be a non-zero, small value.

As the basis of our detection method is identifying discrepancies between subsequent lidar scans, the pointcloud comparison step is relatively significant compared to other parts of the detection pipeline (not yet discussed). For quantitatively evaluating our detection method, we found varying the value of λ_{error} to generate precision-recall curves against groundtruth labels to be an effective way to evaluate our overall performance. For realworld application, we can then refer to the precision-recall curves to select a desirable λ_{error}. In practice, a lower λ_{error} has more false detections. In contrast, a higher λ_{error} has more missed detections (i.e., dynamic points mislabelled as static). For more detail, see Chapter 5 for the experimental results and discussion.

There are also external sources that affect measurements, such as adverse weather conditions (e.g., precipitation or dust storms), but are much more challenging to handle than sensor noise. We choose to treat these cases as outliers and handle them during a later step of the pipeline for outlier filtering.

Scan Gap

The *scan gap*, n_{gap}, is the number of lidar scans between the query scan and the temporally closest scan in the reference pointcloud. Recall the example lidar scan diagram in the top-right of Figure 3.1, where n_{gap} is 4. Ideally, n_{gap} is chosen such that the entirety of a dynamic object will be detected as high error points. In other words, we require an appropriate value for n_{gap} to ensure dynamic objects sufficiently displace between the reference and query pointclouds.

However, consider how the object geometry (shape and size) and speed are not known quantities. Figure 3.4 shows a top-down 1D scenario of two objects (object 1 and 2) passing by. The size of n_{gap} is proportional to the displacement between each object's

new and previous positions. If n_{gap} is chosen that is too small (middle row of figure), the result is a situation where the objects spatially overlap themselves from a previous scan. The overlap is indicated by the red double-sided arrows. This overlap will result in only the front portions of the two objects labelled as dynamic from the comparison.

The obvious solution is to increase n_{gap}, shown in the bottom row of Figure 3.4. Unfortunately, setting a scan gap that is too large may cause overlap between different objects. In the figure we see now that object 2 in the query pointcloud is overlapping object 1 in the reference pointcloud.

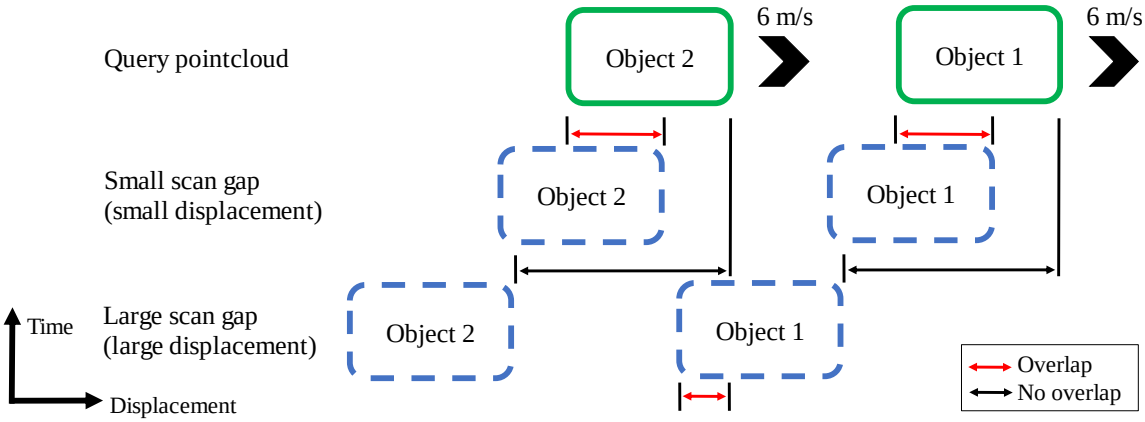


Figure 3.4: 1D scenario of two objects moving toward the right. A pointcloud comparison with a small scan gap will not yield the entirety of the objects because they spatially overlap themselves. Contrarily, a scan gap that is too large is also not desirable. In this case, object 2 in the query pointcloud overlaps with object 1 in the reference pointcloud.

As the geometry and speed of objects are not known quantities, it is impossible to guarantee that the entirety of all objects will be detected. We instead focus on selecting n_{gap} by defining a minimum detection speed to at least partially label dynamic objects. The portions of objects that are not labelled dynamic due to spatial overlap will be recovered in a later part of the detection pipeline. Referring back to Figure 3.4, we prefer the small scan gap scenario (middle row) over the large scan gap scenario (bottom row). The minimum detection speed is taken into account by considering the combination of lidar spinrate, n_{gap}, and λ_{error}. For example, a lidar spinrate of 10 Hz, n_{gap} of 0, and λ_{error} of 0.5 m will not be able to detect objects with a speed less than 5 m/s.

Past Scans vs. Later Scans

We consider the options of comparing the query pointcloud to a reference pointcloud consisting of past scans, later scans, or both. Ideally, we want the latest lidar scan to be the query pointcloud and compare it to past ones since this will be the configuration with the least amount of detection latency. We must verify that comparing to later scans has no additional benefit in detection performance.

Figure 3.5 shows a top-down 1D scenario of an object moving to the right. There are three consecutive scans of the object, where we treat the second scan as the query. Comparing to only the past scan, the detection product (i.e., the object portion labelled dynamic) is the front portion of the object and just the back side. Comparing to only the later scan, the detection product is the back portion of the object and just the front side. Comparing to the composition of the past and later scans results in just the front and back sides, with a middle portion depending on the scan gap.

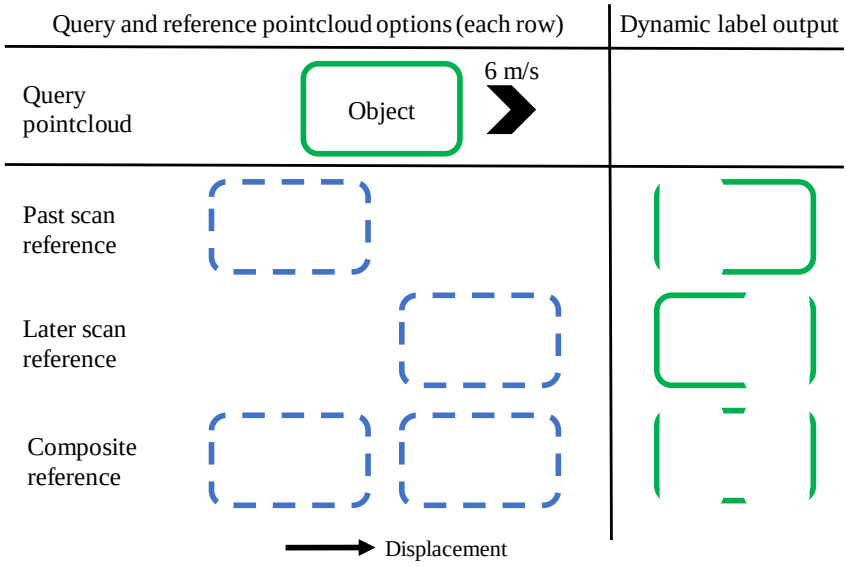


Figure 3.5: 1D scenario showing the options of comparing the query scan to a reference pointcloud consisting of a past scan, later scan, or both. Comparing to a past scan and later scan individually is desirable as they identify different parts of the object, but it is not clear how to take advantage of it. Comparing to the composite reference yields no benefit.

Comparing to the composite pointcloud clearly provides no benefit. We see that

comparing to both past and later scans individually may be desirable as they can identify different portions of the dynamic object. However, it is unclear how to associate the different portions identified to belong to the same object as they can be disconnected with a larger scan gap. A factor this simple example does not take into account is the large amount of the static scene that will be mislabelled dynamic due to spatial sparsity of the points and viewpoint occlusion (recall Figure 3.3, which is a comparison to a single reference scan). Additionally computing the comparison to another scan will equally introduce even more mislabelled points.

Therefore we choose to only compare the query scan to a reference pointcloud consisting of past lidar scans. Additionally comparing to later scans will be more detrimental (more mislabelled points) than beneficial (identify missing object portions), and introduce more latency to the overall detection.

Number of Reference Scans

We established in the previous trade-off discussion that it is desirable to construct a reference pointcloud consisting of past lidar scans. Here we discuss the trade-off associated with selecting the number of scans to construct the reference pointcloud. We set the number of scans as a tunable parameter, n_{scans}.

Due to the lidar sensor resolution, the produced data is inevitably a discretization of the realworld. Lidar points on surfaces further away or ones that make contact at an angle will be spread-out more from their respective neighbour points, compared to points on closer perpendicular surfaces. Notice in the shown pointcloud examples that points on the ground form concentric circles around the lidar. The gap between each consecutive circle is larger at further distances.

The top image of Figure 3.6 shows an example query pointcloud coloured by the error metric in the same way as the previous Figure 3.3 (black is low error, red is high error), using a reference pointcloud consisting of a single past scan. Notice the large

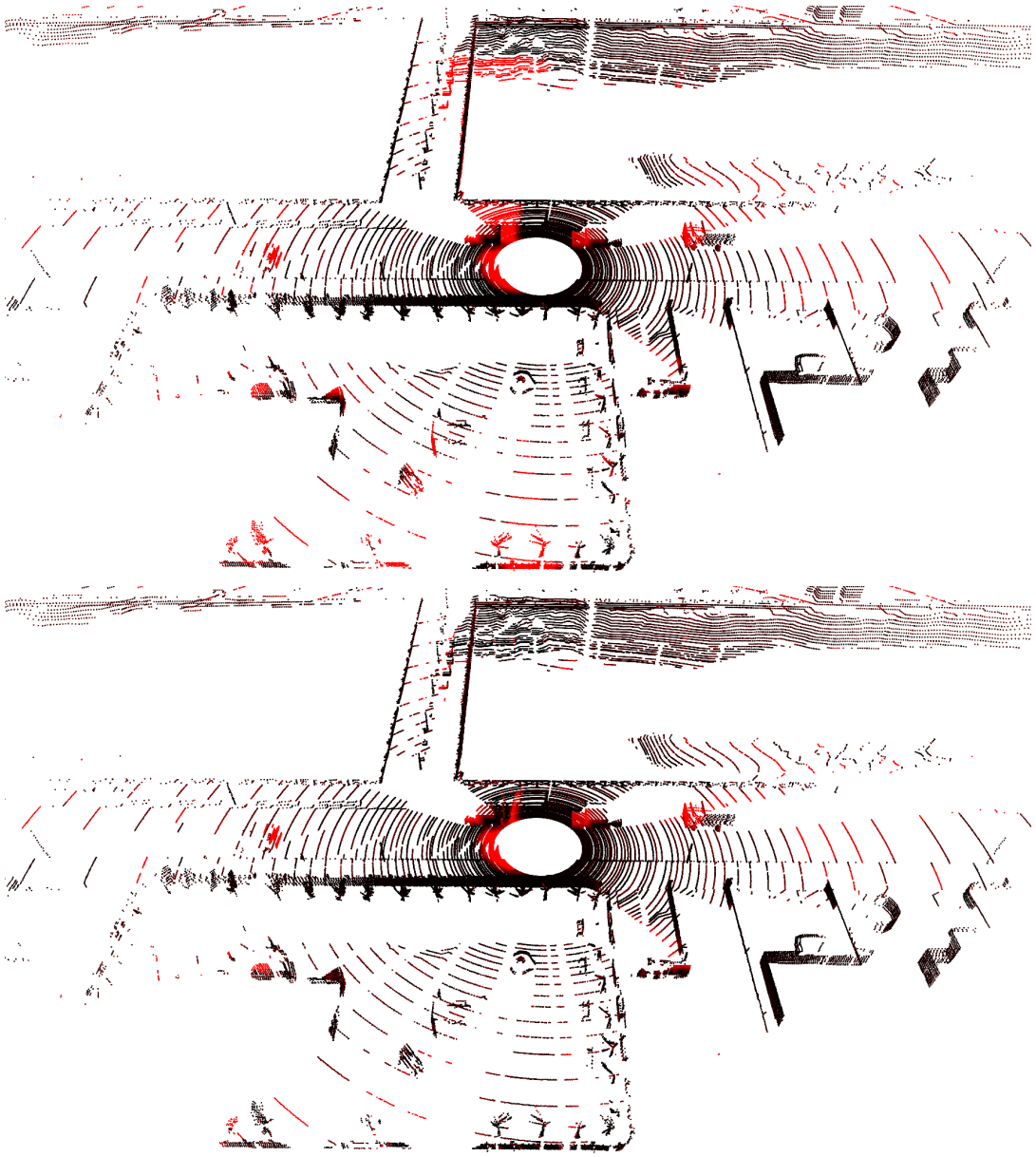


Figure 3.6: The top image is of a pointcloud coloured by the error metric (similar to Figure 3.3) using a reference pointcloud consisting of a single scan. Black indicates low error and red indicates high error. The bottom image is the same pointcloud, but compared to a reference pointcloud consisting of three scans.

number of high error points on static surfaces. This is the limitation of a single lidar scan. The bottom image of Figure 3.6 shows the same query pointcloud, but with a reference pointcloud consisting of three past scans. At the cost of computation by using more scans, there is a noticeable reduction in the number of static points with high error.

There is also a limit to consider when increasing n_{scans}. Similar to how increasing the scan gap, n_{gap}, by too much is detrimental, we run the risk of dynamic objects in the query pointcloud spatially overlapping with objects in the past if scans too far back in the past are included. This can easily occur in driving scenarios due to high traffic. We must consider the maximum speed of objects to ensure these overlaps do not occur.

Point-to-Point vs. Point-to-Plane

As mentioned in the discussion on the number of reference scans, a pointcloud of a single scan will be sparse on oblique and far-away surfaces. In these cases, the point-to-point error metric will yield high values even if the query point and its reference are truly of the same static surface. If there are enough neighbouring points to compute a surface normal estimate, the point-to-plane error will be more robust to this issue.

Using the point-to-plane error metric, which is the preferable metric of the two, comes at the cost of computing surface normals. Surface normals can be computed efficiently with a fast nearest neighbour search and eigendecomposition method, but the computation adds up when each lidar scan consists of a large number of points (e.g., more than 100000).

Surface normals of pointclouds are in general useful for pointcloud-based applications. So far, we described their use in scan alignment (Section 3.2) and pointcloud comparison (Section 3.3). We will later describe their purpose in our region growth method. Multiple methods making use of surface normals makes them more worthwhile to compute. Therefore we choose to use the point-to-plane metric whenever possible (i.e., there are sufficient neighbours to compute a surface normal).

3.3.3 Comparison Limitations

The most apparent limitation for live data pointcloud comparisons as a tool for dynamic object detection is how the objects must be moving in the query scan for it to be detected.

If a dynamic object comes to a stop (i.e., it momentarily becomes static), it will no longer be detected. However, we believe this to be a necessary compromise for model-free, setting-independent detection.

Another limitation is the inherent latency from working with only a window of latest lidar scans, meaning objects must sufficiently displace before they can be detected. Consider the time period where objects transition from a static to dynamic state. There will be latency in detecting the discrepancy, proportional to the scan gap, n_{gap}. Transitions in the opposite way (i.e., dynamic to static) will also have latency, but mislabelling points as dynamic for a couple extra time instances is not as severe of a mistake compared to a delayed detection of dynamic points.

Finally, we see through qualitative examples (Figure 3.3 and 3.6) that pointcloud comparisons alone are not enough for good detection performance. The lidar moving on a vehicle or robot causes new surface observations and viewpoint occlusions, which causes high error for many points on static surfaces. The number of points mislabelled dynamic is therefore, unfortunately, significant. It is not obvious how these mislabels can be reduced from pointcloud operations alone. Instead, we will next introduce freespace checking which is able to correct these mislabels.

3.4 Freespace Check

We define freespacing checking as querying whether all points of a query lidar scan are inside, on the border of, or outside the freespace of a reference lidar scan. Recall that a lidar sensor gives freespace information from the paths traced by its laser measurements. The previous section on pointcloud comparisons shows a good initial step in identifying discrepancies between subsequent lidar scans, which are indicative of dynamic objects. Unfortunately, pointcloud comparisons will label a large number of points on static surfaces as dynamic (high error) due to limited sensor resolution, new surface observations,

and viewpoint occlusions (recall Figure 3.3 and 3.6). We rely on checking against the freespace of lidar scans to correct these mislabels.

Figure 3.7 shows a 2D example scenario depicting data from two positions of a lidar (circles) and two positions of a dynamic object (rectangles). The block structure in grey is static. In this example we treat data from position 1 as the query, where the stars are a coarse representation of the query pointcloud. The space shaded in green indicates the freespace of the scan from position 2. Computing a pointcloud comparison by setting the lidar position 2 scan as the reference results in dynamic (high error red stars) and static (low error black stars) labels. Dynamic points of the static structure, which are mislabels, are not within the shown freespace, but rather are on the freespace boundary. However, dynamic points belonging to the dynamic object (object position 1), which are correct labels, are within the freespace.

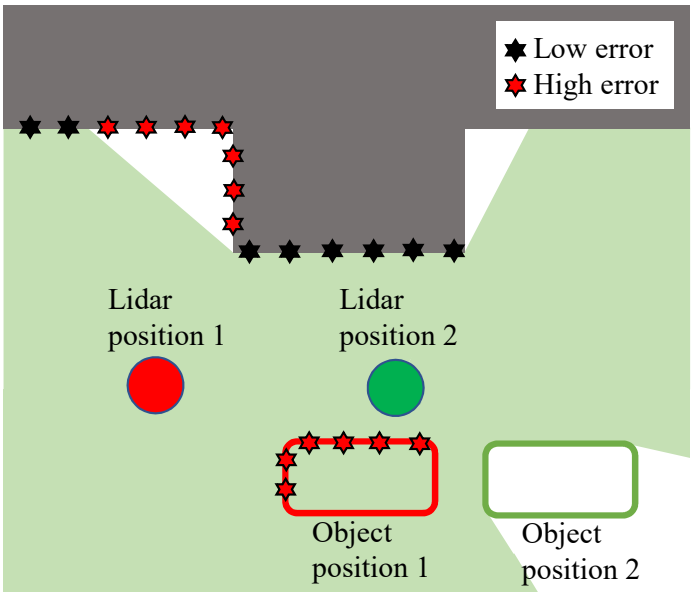


Figure 3.7: 2D lidar example showing data from two positions of a lidar (circles) and two positions of a dynamic object (rectangles). The stars are a coarse representation of the lidar points from lidar position 1. The shaded area in green indicates freespace of the scan from lidar position 2. A pointcloud comparison labels high error points as dynamic (red) and low error points as static (black). We see that the mislabelled dynamic points, those that are on the static (grey) structure, are not within the depicted freespace. Correctly labelled dynamic points, those on the dynamic object, are within the freespace.

Our simple example demonstrates the use case of freespace in labelling points. If a query point is determined to be inside the reference freespace, it is of a dynamic object (i.e., some surface exists in a particular location at the time of the query scan, but not the same location in the reference scan). Points determined to be on the border of freespace are likely of static objects. For points determined to be outside the reference freespace, we cannot establish whether they are static or dynamic. In this event, we may have to query the freespace of another reference scan.

An important observation is that there is an overlap between pointcloud comparisons and checking against freespace. A pointcloud comparison can be considered as an efficient way of completing a freespace border check. Therefore we can reduce computation by pipelining freespace queries after the pointcloud comparison, only checking points labelled dynamic against freespace.

Figure 3.8 shows the same query pointcloud of Figure 3.6, but after completing the freespace check. Note that instead of a colour gradient between black and red, we now show a binary label of static (black) or dynamic (red). The result is a significant improvement over what was shown earlier from just a pointcloud comparison. Indicated with circles are areas that need improvement, but are beyond the scope of the freespace check. The rest of this section describes the freespace query algorithm. We end this section with a discussion on the need for using two freespace references for each query scan, where one is of a past scan, and the other, unfortunately, has to be of a later scan.

3.4.1 Freespace Query Algorithm

Given a query point and reference scan that defines the freespace of interest, we wish to determine if the query point is inside, on the border of, or outside freespace. Recall that the spinning lidar continuously sweeps lasers about an axis for a 360° FOV. The laser ray paths, from the sensor to their endpoints, define freespace. We design our freespace query algorithm specifically for a single lidar configuration that has its lasers approximately

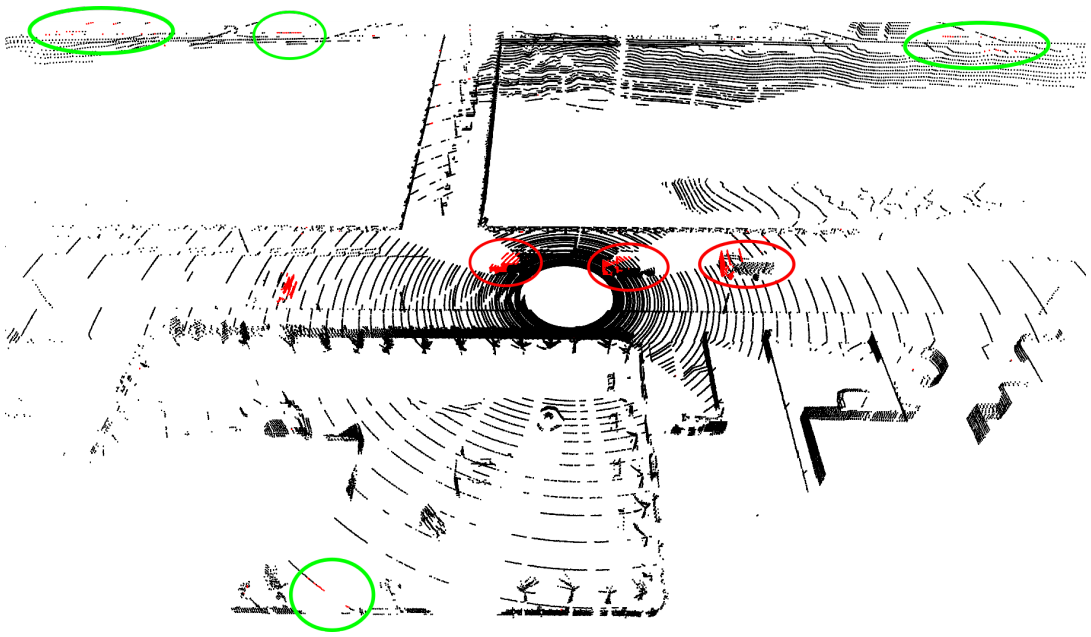


Figure 3.8: An example query pointcloud (the same as in Figure 3.6) after completing the freespace check. Here the points are labelled static (black) or dynamic (red). The labelling of this pointcloud is a significant improvement over just relying on the pointcloud comparison. However, there is still room for improvement beyond the scope of the freespace check. Circled in green are noticeable areas with points mislabelled dynamic. Circled in red are dynamic objects that are not completed labelled dynamic.

radiating outward vertically (i.e., along the sweeping axis). This is important because we exploit the elevation order of the lasers to speed up our freespace query.

Representing freespace in its entirety is possible, but is expensive in computation and memory (e.g., occupancy voxel grids). Instead, we only determine the reference scan measurement ray that has a direction that passes nearest to the query point in question (i.e., smallest point-to-line distance). Consider the query point surface plane and the identified reference scan ray – there are three cases (refer to illustration in Figure 3.9):

- Case 1: Ray intersects the surface plane.
- Case 2: Ray lies on the surface plane.
- Case 3: Ray does not reach the surface plane.

Case 1 means the surface plane is absent during the time period of the reference scan,

which is possible if the query point is an observation of a moving surface and is inside freespace. Case 2 means the measurement is likely an observation of the same surface plane (freespace border). Finally, Case 3 means another surface obstructed the measurement (outside freespace).

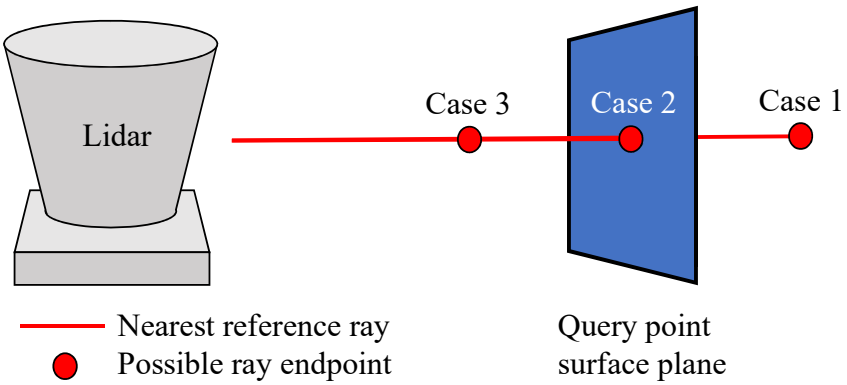


Figure 3.9: An illustration depicting the three cases for freespace querying. The blue rectangle represents the query point surface plane. Case 1 has the nearest reference ray intersect the plane, indicating the query point is inside the reference freespace. Case 2 has the reference point lie on the surface plane (to some margin), indicating the query point is on the boundary of the reference freespace. Case 3 has the reference ray not reach the surface plane, indicating the query point is outside the reference freespace.

Pomerleau et al. (2014) also use a nearest-ray strategy by constructing a kd-tree of spherical coordinates of the reference scan, which they can efficiently search for each query point after converting it as well to spherical coordinates. However, this method requires the assumption that their pointclouds are instantaneous snapshots (i.e., ideal pointcloud assumption) of the scene and that all measurement rays originate from a single point in space (i.e., local frame origin). Instead, we compensate for the sensor platform motion by using our continuous-time trajectory of the moving sensor, $\mathbf{T}_{v,0}(t)$, where we define a time-varying vehicle reference frame, $\underrightarrow{\mathcal{F}}_v$. $\mathbf{T}_{v,0}(t)$ is evaluated using the GP interpolation in Equation (3.6).

We assume a total of L lasers rotate together at constant speed, ω. Each laser, ℓ, is indexed in order of increasing elevation and has a unique pose with respect to the sensor hub (i.e., the continuously rotating base), $\underrightarrow{\mathcal{F}}_h$, defined by the transformation

$\mathbf{T}_{\ell,h} \in SE(3)$.

Given a query point, $\mathbf{q}_0$, we formulate for each laser, ℓ, the point-to-line distance as a continuous function of time:

$$\|\mathbf{e}_\ell(t)\|_2 = \left\| \mathbf{D}\mathbf{T}_{\ell,h}\mathbf{T}_{h,v}(t)\,\mathbf{T}_{v,0}(t) \begin{bmatrix} \mathbf{q}_0 \\ 1 \end{bmatrix} \right\|_2 . \tag{3.11}$$

We introduce

$$\mathbf{T}_{h,v}(t) = \begin{bmatrix} \mathbf{R}^z(\omega t) & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \in SE(3), \quad \mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

where we assume the rotation of $\underrightarrow{\mathcal{F}}_h$ with respect to $\underrightarrow{\mathcal{F}}_v$ as a rotation at constant angular speed, $\mathbf{R}^z(\omega t) \in SO(3)$, at rotation speed ω. Note that we purposely define $\underrightarrow{\mathcal{F}}_v$ to be equivalent to the sensor base frame, $\underrightarrow{\mathcal{F}}_b$ (recall Section 2.1.1). We also define $\underrightarrow{\mathcal{F}}_v$ such that there is no translation between $\underrightarrow{\mathcal{F}}_h$ and $\underrightarrow{\mathcal{F}}_v$, and $\underrightarrow{\mathcal{F}}_h$ rotates about the z-axis of $\underrightarrow{\mathcal{F}}_v$. We define $\underrightarrow{\mathcal{F}}_\ell$ such that the laser points along the x-axis. See Figure 3.10 for a visualization.

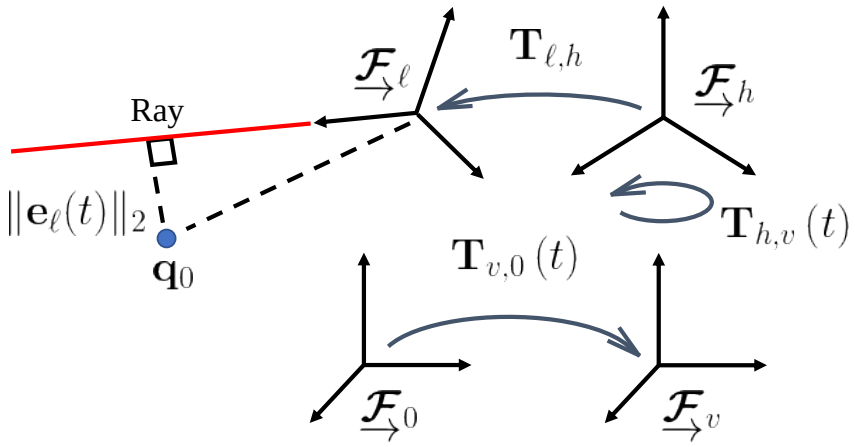


Figure 3.10: A visualization of the point-to-line distance in Equation (3.11). $\mathbf{T}_{v,0}$ is the sensor trajectory, $\mathbf{T}_{h,v}$ is the spinning lidar rotation, and $\mathbf{T}_{\ell,h}$ is the unique pose for laser ℓ. We require the time, t, and laser, ℓ, combination that minimizes $\|\mathbf{e}_\ell(t)\|_2$.

We require t^* and ℓ^* that minimize $\|\mathbf{e}_\ell(t)\|_2$. Unfortunately, ℓ is discrete. We minimize $\|\mathbf{e}_\ell(t)\|_2$ iteratively by selecting ℓ (i.e., our best guess) and solving for t using nonlinear least squares optimization:

$$t = \arg\min_t \frac{1}{2}\mathbf{e}_\ell(t)^T\mathbf{e}_\ell(t). \tag{3.12}$$

We iterate by exploiting laser elevation order. We first solve Equation (3.12) using initial guesses for t and ℓ, the laser neighbour above it, $\ell+1$, and below it, $\ell-1$. If a neighbour optimizes to a smaller $\|\mathbf{e}^\ell(t)\|_2$, we iteratively search along that direction by single laser increments, re-solving Equation (3.12) and comparing optimized $\|\mathbf{e}^\ell(t)\|_2$ values. The iteration stops once $\|\mathbf{e}_\ell(t)\|_2$ no longer decreases or we run out of neighbours. Our iterative method relies on a good initial condition for t and ℓ, which we select by using the ideal pointcloud method of Pomerleau et al. (2014). We experimentally verified this initialization choice always converges.

As the optimization in Equation (3.12) is nonlinear, we also require iteration to solve for t. We solve Equation (3.12) using Gauss-Newton by linearizing $\mathbf{e}_\ell(t)$ with respect to t:

$$\mathbf{e}_\ell(t) \approx \bar{\mathbf{e}}_\ell(t) + \mathbf{E}_\ell\delta t, \tag{3.13}$$

where δt is the time perturbation, $\bar{\mathbf{e}}_\ell(t)$ is the nominal term, and

$$\mathbf{E}_\ell = \frac{d\mathbf{e}_\ell(t)}{dt} = \mathbf{D}\mathbf{T}_{\ell,h}\left(\dot{\mathbf{T}}_{h,v}\mathbf{T}_{v,0} + \mathbf{T}_{h,v}\dot{\mathbf{T}}_{v,0}\right)\begin{bmatrix}\mathbf{q}_0\\ 1\end{bmatrix}. \tag{3.14}$$

This time derivative, $\mathbf{E}_\ell$, has an exact expression since we know the time derivative of an $SE(3)$ transformation (Barfoot, 2017) to be

$$\dot{\mathbf{T}} = \boldsymbol{\varpi}^\wedge\mathbf{T}, \tag{3.15}$$

where $\boldsymbol{\varpi}$ is the generalized velocity of $\mathbf{T}$.

We substitute Equation (3.14) into Equation (3.12), differentiate the cost function with respect to δt, set the derivative expression to 0, and solve for δt. This procedure results in the ith time update

$$t^{(i)} \leftarrow t^{(i-1)} - \left[\left(\mathbf{E}_\ell^T \mathbf{E}_\ell \right)^{-1} \mathbf{E}_\ell^T \mathbf{e}_\ell(t) \right] \Big|_{t=t^{(i-1)}}, \tag{3.16}$$

which we iterate until convergence.

Given t^* and ℓ^*, the reference scan ray (i.e., the ray closest to the query point) is the measurement of laser ℓ^* with the closest measurement time to t^*. Adhering to the three possible cases, we check for an intersection between the ray and surface plane of the query point. This is easily done by computing the point-to-plane error metric of Section 3.3 with the ray endpoint for a freespace border test. Otherwise, the query point is inside or outside freespace depending on which side of its surface the endpoint resides in.

3.4.2 Freespace Reference Scans

Just as in the pointcloud comparison, we ideally only require a freespace check against a single past lidar scan. Unfortunately, dynamic objects moving away from the sensor will never be within the freespace of a previous scan. In order to detect such dynamic objects, we require an additional freespace check against a later lidar scan, where the points of objects moving away will clearly be within the later scan's freespace. This is a common occurrence in driving scenarios, consider for example vehicles driving ahead of the sensor in the same direction. Unfortunately, by using a later scan as a freespace reference, we introduce latency to the overall detection.

Recall the discussion regarding the scan gap, n_{gap}, in Section 3.3.2 on pointcloud comparisons. The same trade-off applies here, where we require a sufficient gap between the query and reference scans. Therefore it is logical to use the same n_{gap} parameter

value for determining the freespace reference scans. However, dynamic objects moving away from the sensor will likely have surface geometry perpendicular to the movement direction. Surface geometry perpendicular to the movement direction is not susceptible to the spatial overlapping issue we saw in Section 3.3.2. From this observation, we understand that a scan gap is not required between the query scan and the later reference scan, and only for the past reference scan, minimizing the added latency.

Another important aspect is how only query points identified as outside the freespace of the past reference scan need to be queried against the later reference scan. This significantly reduces the required computation.

3.5 Box Filter

Our freespace check eliminates the majority of the mislabelled dynamic points from the pointcloud comparison, but is susceptible to mistakes because of finite lidar resolution. Consider how adjacent laser rays diverge from one another as they extend further away from the sensor. Determining freespace accurately at further ranges is more difficult, causing our freespace check method to leave sparse traces of mislabelled dynamic points.

Now recall the imagespace representation of lidar data described in Section 2.1.2. We can arrange the query scan measurements into its imagespace representation, but with values according to their detection label. We set cell values of 1 for dynamic labels and 0 for static labels. The left image in Figure 3.11 shows a portion of the horizontal FOV of a query scan (full vertical FOV) with detection labels after the pointcloud comparison and freespace check. This was completed using simulated data with known ground truth, so we are able to label correct dynamic labels as green and the incorrect dynamic labels as red. White cells indicate static labels.

Notice the sparse traces of incorrect labels (red) and how they form horizontal lines. The occurrence of the horizontal pattern is due to how this specific lidar has a lower

vertical resolution compared to the horizontal. This is a trait of most, if not all, spinning-lidars with 360° horizontal FOV, causing points to be more spread out spatially along the vertical imagespace axis than horizontal (i.e., higher horizontal resolution).

We filter the points mislabelled dynamic by sliding a box filter throughout the image. We use a filter with a horizontal pattern, for example:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

We apply our filter with a pixelwise XNOR (exclusive logical NOR) operation. The sum of all XNOR operations is a numerical score. Scores greater than a constant *score threshold*, λ_{filter}, are considered outliers. λ_{filter} and the dimensions of the filter are tunable parameters that depend on the lidar's specifications (e.g. measurement resolution). The result of applying the filter to the example image on the left of Figure 3.11 is shown in the image on the right.

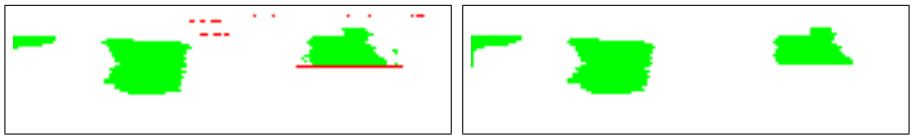


Figure 3.11: We apply a box filter with a horizontal pattern to a query scan's image representation to remove mislabelled dynamic points (red) while maintaining correct ones (green). We show a before (left) and after (right) example.

The box filter relies on the sparsity pattern of the mislabelled dynamic points. This means it will not be as effective if applied directly after the pointcloud comparison because there are too many points mislabelled dynamic (refer back to Figure 3.3 and 3.6).

3.6 Clustering and Region Growth

Recall Section 3.3.2 where we discussed the necessary scan gap, n_{gap}, between the query scan and its reference. We established that it is not possible to guarantee that the pointcloud comparison, and consequently the freespace check, will identify the entirety of dynamic objects due to spatial overlap. This is because the speed and geometry of dynamic objects are not known quantities, making it impossible to select the necessary n_{gap}. We see this issue in Figure 3.8, where only the front portions of dynamic objects are labelled dynamic (red).

In this section, we describe a region growing method to regain dynamic points that were incorrectly labelled as static during the pointcloud comparison and freespace check. We first cluster the current dynamic query points into object clusters, to which we then apply region growth.

For clustering, we use the 3D pointcloud clustering method presented by Klassing et al. (2008). A radially bounded nearest neighbour strategy incrementally groups dynamic points into clusters. Points neighbours, as we are only concerned with points within a single scan, are efficiently found by exploiting the point ordering of the scan's imagespace representation.

The clustering algorithm is only applied to points labelled dynamic, and is briefly described by the following steps:

1. Step through all dynamic points.

2. If the current point is assigned to a cluster, continue to the next point.

3. For the current point that is not part of an existing cluster:

    - Find all neighbours within a set radial bound, r_{neigh}.
    - If any neighbours belong to a cluster, assign the current point and all other neighbours without a cluster to the same cluster.

- If there exists neighbours belonging to different clusters, merge all the clusters.

After clustering, we are left with a set of clusters that are dynamic objects. These objects may only have a portion of its entirety labelled dynamic and thus require region growth to correct the neighbouring points that are mislabelled static.

We make the assumption that all objects are rigid and are made up of locally convex surfaces (i.e., smooth surfaces curve outward or adjacent flat surfaces form obtuse outer angles). We use a criterion presented by Moosmann et al. (2009) called the local convexity criterion, for which they applied to a pointcloud segmentation algorithm. Given two points $\mathbf{p}_1$ and $\mathbf{p}_2$, with unit surface normals $\mathbf{n}_1$ and $\mathbf{n}_2$, the two points are of convex surfaces if the following two conditions are both true:

$$\mathbf{n}_1 \cdot (\mathbf{p}_2 - \mathbf{p}_1) \leq 0, \quad \mathbf{n}_2 \cdot (\mathbf{p}_1 - \mathbf{p}_2) \leq 0.$$

We grow each cluster by iteratively testing neighbouring measurements for parallelism *or* convexity, indicating they are part of the same surface, until none are found. Neighbours are identified using the same method as the clustering algorithm (i.e., points within a radial bound, r_{neigh}). Parallelism is tested by simply taking the dot product of neighbouring surface normal vectors and taking a constant threshold. The two surface normals, $\mathbf{n}_1$ and $\mathbf{n}_2$, are parallel if the following condition is true:

$$\mathbf{n}_1 \cdot \mathbf{n}_2 > \lambda_{\text{parallel}}, \quad 0 \leq \lambda_{\text{parallel}} \leq 1.$$

The region growing algorithm is similar to the clustering algorithm with the addition of a first-in-first-out queue. We also now consider all points, not just the ones labelled dynamic. The algorithm is described by the following steps for each cluster:

1. Queue all points of the cluster.

2. For the next point in the queue:

- Find all neighbours that do not belong to a cluster within a set radial bound, r_{neigh}.
- Check each neighbour for parallelism or convexity. Add the neighbour to the cluster if conditions are met.
- Add to the queue all new cluster points .

Figure 3.12 shows the same query pointcloud example of Figures 3.6 and 3.8 after applying region growth. Points previously dynamic are red, and dynamic points from the result of region growth are green. Where previously we had only the front portions of the objects labelled dynamic, our region growth method is able to recover the mislabeled static portions as dynamic.

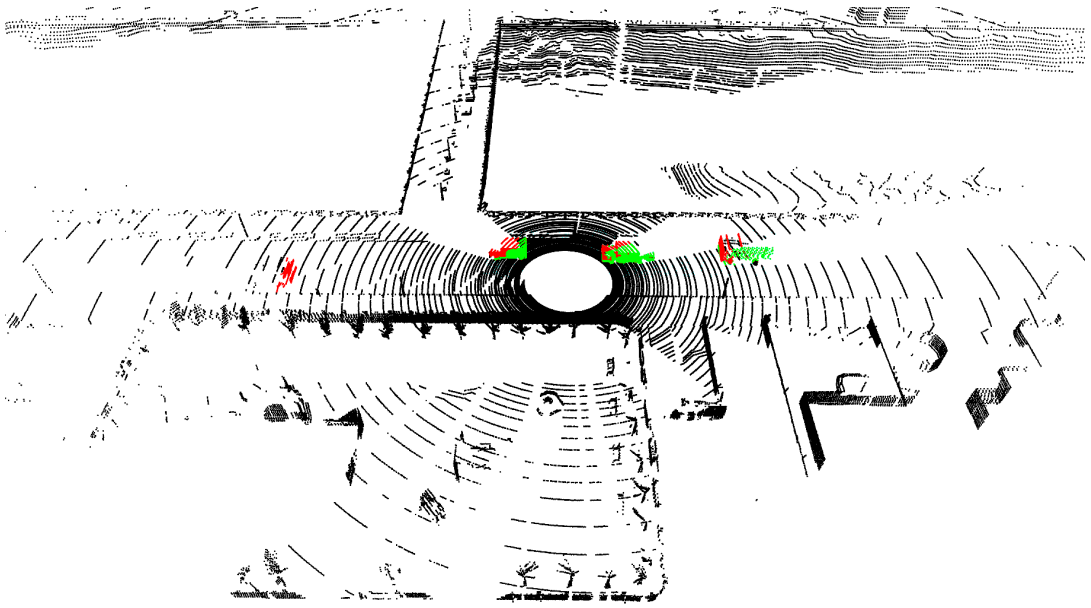


Figure 3.12: An example query pointcloud (the same as in Figures 3.6 and 3.8) after completing region growth. As before, the points are labelled black for static or red for dynamic. The green points are the result of region growth. Portions of dynamic objects previously mislabelled as static are successfully recovered.

3.7 Summary

In summary, this chapter was an extensive look into the methodology behind the detection pipeline and its individual components. We presented a scan alignment algorithm that compensates for the exact measurement times of lidar data to align the latest lidar scan and also produces a continuous-time trajectory. We compute an initial label for all query points by completing a pointcloud comparison with error metrics familiar to the scan alignment problem. We correct for dynamic mislabels by checking them against the freespace of other scans, which we accomplish with a novel freespace querying algorithm that uses the continuous-time trajectory output from the scan alignment. We additionally correct for dynamic mislabels using a box filter on the imagespace representation of the query scan, which we designed by exploiting the sensor scanning characteristics. Finally, we describe a region growth method that can improve detection in cases where the previous components do not label dynamic objects in their entirety. Where this chapter only presents qualitative examples, we show in Chapter 5 quantitative analyses of the different components and the full detection pipeline.

Chapter 4

Datasets and Benchmark

In this chapter we present the lidar datasets we have collected to evaluate our dynamic object detection method. We begin with describing the realworld dataset collected using a Velodyne HDL-64E lidar sensor, which unfortunately does not have groundtruth labels. In order to produce a quantitative analysis, we collect lidar data using CARLA (Dosovitskiy et al., 2017), an open-source simulator for autonomous driving research. We present the details behind the simulated dataset, including modifications to the simulator and explain the benchmark that we establish with it.

4.1 Realworld Dataset

To collect a realworld dataset, we operated a perception vehicle belonging to Autonomous Space Robotics Lab (ASRL). A suite of sensors is mounted onto the vehicle, which includes a Velodyne HDL-64E, a front-facing Bumblebee2 stereo camera, and an Applanix POS-LV inertial navigation system. The POS-LV system includes an inertial measurement unit (IMU), a global positioning system (GPS) receiver antenna, and a wheel encoder on the left-side rear wheel. For the work presented in this thesis, we only make use of the Velodyne HDL-64E lidar sensor. Figure 4.1 shows an image of the vehicle, with a close-up of the roof rack in the top-left corner.

Figure 4.1: The ASRL perception vehicle, a Buick Encore. It is equipped with a suite of sensors, including a Velodyne HDL-64E, Bumblebee2 stereo camera, and an Applanix POS-LV system. The top-left corner image is a closer look at the roof rack.

We collected data by driving the vehicle in Richmond Hill, Ontario. This includes residential areas, larger and busier roads on major streets, and highways. A total of 60.4 km of data was collected. Lidar data was logged at 10 Hz, where each scan is a single revolution.

4.2 Simulated Dataset

A qualitative analysis of a detection method requires accurate groundtruth labels. Since manually labelling data is a tedious and intensive task, we rely on simulated data. This section discusses the simulator we chose, CARLA (Dosovitskiy et al., 2017), and the benchmark we establish using the collected data.

4.2.1 Simulator

CARLA is an open-source simulator for autonomous driving research (Dosovitskiy et al., 2017). The implementation of the simulator is an open-source layer over Unreal Engine 4, a video game engine developed by Epic Games. Two urban maps are provided with 2.9 km (Town 1) and 1.9 km (Town 2) of drivable roads. We currently made 5 min sequences from each map, which we plan on expanding in the near future. Visit http://asrl.utias.utoronto.ca/datasets/mdlidar/index.html for more detail about the dataset and download links.

We made modifications to the CARLA source code to produce datasets matching a real Velodyne HDL-64E (e.g., laser positions and orientations). Notice the jagged pattern of the lidar measurements where the scan starts and stops in Figure 4.2. We capture motion distortion by making each laser take a measurement once every simulation step, resulting in 128000 measurements with a maximum range of 120 m at a frequency of 10 Hz. This means we simulate with an extremely small discrete timestep (0.05 milliseconds). While this is too computationally expensive for a real-time simulator, we only need to collect an offline dataset. See an example pointcloud in Figure 4.2, which also has a corresponding image from a simulated camera in the bottom-right corner.

Key points that highlight our configuration of CARLA are as follows:

- We used CARLA version 0.7.1.
- All vehicles, including the sensor vehicle, are driven with the provided autopilot implementation.
- There are 90 other vehicles driving in the roads of Town 1.
- There are 60 other vehicles driving in the roads of Town 2.

As with any simulation, there are notable limitations. Currently there are only vehicles (i.e., no pedestrians or cyclists). This has been revised in later versions of CARLA

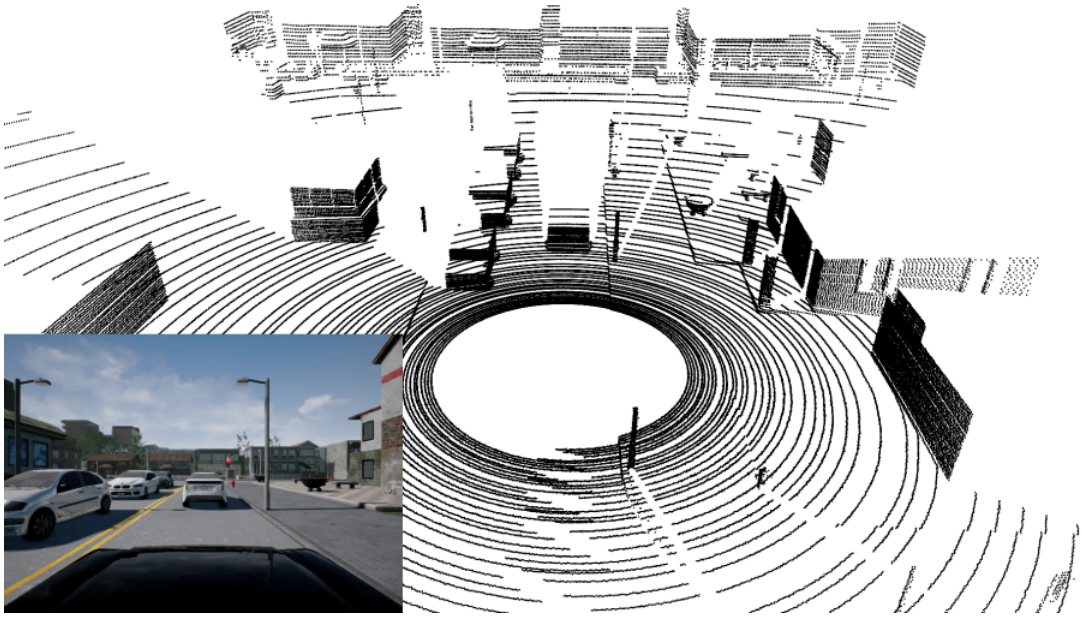


Figure 4.2: Example image and pointcloud pair from the CARLA simulator. The image is from a simulated camera sensor mounted at the front of the sensor vehicle. The pointcloud is the product of a lidar sensor mounted on top of the vehicle.

and we intend on updating our dataset in the near future. Unfortunately, dynamic objects use primitive collision geometry, which is what the simulated lidar rays observe. For example, the vehicles in Figure 4.2 are rendered with detail in the image (bottom-left), but the corresponding pointcloud is made of rectangular boxes and spheres. Objects introduced in recent updates to CARLA, such as pedestrians, are ellipsoidal blobs, so they do not justify immediate priority in updating our dataset.

Another limitation is the lack of intensity measurements. Simulating intensity is potentially possible considering the simulator has information on surface material and geometry is known, but implementation will require substantial work.

Finally, the driving simulation only consists of roads with two lanes and at-most three-way intersections. There are no four-way intersections, which is more complicated to implement as it requires more sophisticated traffic rules.

Overall, CARLA provides an urban driving simulation that we believe to have enough fidelity in scene detail to be used as a comparison for benchmarking detection methods. Consider the example pointcloud in Figure 4.2, which has scene detail from large buildings

to more small-scale detail such as water fountains, fire hydrants, and traffic signs.

4.2.2 Benchmark

We establish a benchmark using simulated data by defining a groundtruth label threshold and comparison metrics based on precision and recall. For groundtruth, points moving faster than 0.2 m/s are considered dynamic. We define true positives (TP) as points correctly labelled dynamic. We define false positives (FP) as points incorrectly labelled dynamic, and false negatives (FN) as points incorrectly labelled static.

We compute precision, P, and recall, R, in two ways. Given the scan index, n, and the total number of scans, N, the *total* computation is:

$$\mathrm{P}_t = \frac{\sum_n^N \mathrm{TP}_n}{\sum_n^N (\mathrm{TP}_n + \mathrm{FP}_n)}, \; \mathrm{R}_t = \frac{\sum_n^N \mathrm{TP}_n}{\sum_n^N (\mathrm{TP}_n + \mathrm{FN}_n)}. \tag{4.1}$$

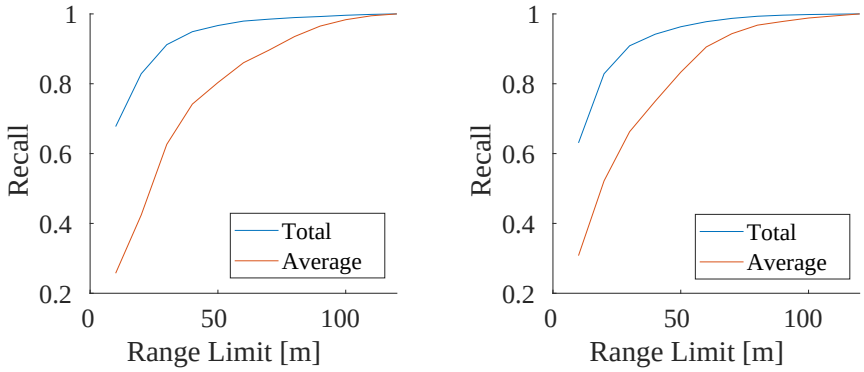
Given the total number of valid scans for precision, N_p, and the total number of valid scans for recall, N_r, the *average* computation is:

$$\mathrm{P}_a = \frac{\sum_n^{N_p} \mathrm{TP}_n / (\mathrm{TP}_n + \mathrm{FP}_n)}{N_p}, \; \mathrm{R}_a = \frac{\sum_n^{N_r} \mathrm{TP}_n / (\mathrm{TP}_n + \mathrm{FN}_n)}{N_r}. \tag{4.2}$$

Scans where the denominator is 0 are ignored (e.g., $\mathrm{TP}_n + \mathrm{FP}_n = 0$), which is why we distinguish N_p and N_r.

The reason for providing two alternative methods for computing precision and recall is demonstrated by the plots in Figure 4.3. The plots show recall evaluated with groundtruth at varying range limits (i.e., measurements greater than the range limit are ignored). Observe how the total recall, R_t, is higher for low range limits compared to the average recall. R_a. This is due to how there are more points on objects closer to the lidar, which are easier to correctly label. Objects further away from the sensor are more difficult to correctly label, but such objects have less points. Thus the neglection of far-away objects is downplayed when computing R_t. In comparison, R_a averages over each

lidar scan, limiting the downplay of far-away objects compared to R_t. Therefore we provide two alternative methods to capture the effect of measurement range and pointcloud density on detection performance.



(a) Town 1. (b) Town 2.

Figure 4.3: Recall (*total* and *average* - see Equation (4.1) and (4.2)) using groundtruth labels on two simulated sequences with varying limited range. Total recall is high at low range limits because nearby objects have more points, downplaying far-away ones.

4.3 Summary

In summary, this chapter presents the lidar data required to evaluate our dynamic detection method. We collected real lidar data using a Velodyne HDL-64E in driving scenarios, which unfortunately does not have groundtruth labels of dynamic objects. This limits our use of the real dataset for a qualitative evaluation. Instead, we look to a proficient simulator for collecting lidar data with accurate groundtruth. We use CARLA, an open-source simulator for autonomous driving research. We modify the simulator to suite our needs, such as implementing a Velodyne-like lidar sensor with motion distortion, and make the dataset public for others to use. We also establish a benchmark for labelling lidar data as dynamic or static at the point level by defining two alternatives for computing precision and recall.

Chapter 5

Pipeline Simulation and Experimental Results

In this chapter we break down the detection pipeline to quantitatively analyze the individual components that were described in detail in Chapter 3. We then evaluate the full detection pipeline for quantitative and qualitative analyses using the simulated lidar benchmark and collected real data, respectively.

5.1 Analysis of Pipeline Components

This section analyzes the contribution of individual components of pipeline (e.g., point-cloud comparison, freespace check, etc.) by removing components from the full pipeline and testing the resulting (simpler) pipelines against the simulated data. Note that we only use the Town 1 simulated dataset. Range measurements are injected with a zero-mean Gaussian noise, with a standard deviation of 0.01 m. As mentioned in Section 3.3.2, we vary the value of the error threshold parameter, λ_{error}, to generate precision and recall values. We set the scan gap, n_{gap}, to 4. Note that for these simulated results, we used the groundtruth trajectory instead of our scan alignment algorithm to focus on the detection aspect.

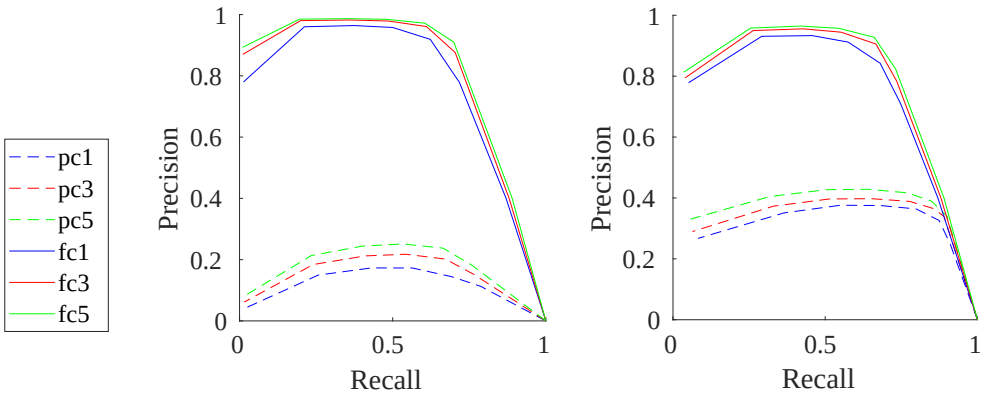
5.1.1 Freespace

In Chapter 3 we saw the limitation of only using the pointcloud comparison for dynamic object detection through qualitative examples (e.g., images of labelled pointclouds). In this subsection we will show the same conclusion, but quantitatively using the Town 1 dataset.

We define two variations of the pipeline, *pc* and *fc*. The first pipeline, pc, only uses the pointcloud comparison component. In other words, referring to the pipeline diagram in Figure 3.1, the output of the pc pipeline is at the step marked by the letter (a). The second pipeline, fc, uses both the pointcloud comparison component and the freespace check components. The output of the fc pipeline is at the step marked by the letter (b).

We evaluate both pipelines against the Town 1 dataset, using the two methods of computing precision and recall, as defined in Chapter 4 (see Equation (4.1) and (4.2)). An additional parameter we vary is the number of reference scans in the pointcloud comparison, n_{scans}. We vary n_{scans} as 1, 3, and 5. The precision-recall curves resulting from the two pipelines is shown in Figure 5.1. The total precision-recall curves are on the left, with the average variation on the right. The numbers in the legend indicate the value of n_{scans}.

The first observation to address is why the precision-recall curves approach the 0 precision, 0 recall position. Normally we expect a trade-off between precision and recall, where the extreme parameter setting that yields perfect precision (value of 1) will have a corresponding recall value of 0, and vice versa. In our case, we instead approach 0 precision and 0 recall as we increase our threshold parameter, λ_{error}, to a very large value. Recall that precision is the relationship between TPs, correct dynamic labels, and FPs, incorrect dynamic labels. It turns out that the query point with the largest error metric over an entire data sequence will likely always be of a static surface. This is expected as the size of dynamic objects are at most a few meters, while the maximum lidar measurement range is 120 m. Much higher error will result on static surfaces simply



(a) Total (b) Average

Figure 5.1: Precision-recall curves comparing two pipeline variations, pc and fc. On the left is the total variation and on the right is the average variation (see Equation (4.1) and (4.2)). The numbers in the legend indicate the number of reference scans, n_{scans}. We see that the recall of the pc pipeline is extremely poor, where adding the freespace check in the fc pipeline dramatically improves it.

due to the moving sensor and maximum range. Therefore as λ_{error} approaches the highest error value, we are left with no TPs, and precision evaluates to 0.

While this behaviour may be undesirable, the values of λ_{error} where it occurs are at a region where we do not wish to operate in the first place (i.e., very large values that will have close to 0 recall). Therefore we ignore this behaviour for the rest of our qualitative analyses, focusing on more useful (smaller) values of λ_{error}.

More important is the comparison of the pipelines pc and fc. We observe a dramatic improvement in precision in both the total and average, which agrees with the qualitative pointcloud examples shown previously. Also note the slight increase in precision as we increase n_{scans} for both pipeline variations. This was another result that we concluded qualitatively in Chapter 3, which we now show quantitatively.

For the fc pipeline, the total curves show slightly better precision and recall. Recall in Section 4.2.2 we discussed how the total computation downplays the effect of objects in the distance which have less points. We see this behaviour here as the total curves are slightly better. Strangely, the pc pipeline shows better precision in the average curves compared to its total curves. This result does make sense however, the strange behaviour

is just an outcome of the poor performance. Consider how the pc pipeline will label (incorrectly) almost all points as dynamic, except for the instances where the sensor vehicle is stationary (e.g., because of a traffic light). Those scans with no movement are cases where pointcloud comparisons alone perform very well and will correctly label static points. The average precision computation gives more weight to those scans with no movement. In the end, this difference between the total and average pc pipeline curves means very little as both have incredibly poor performance.

5.1.2 Motion Compensation

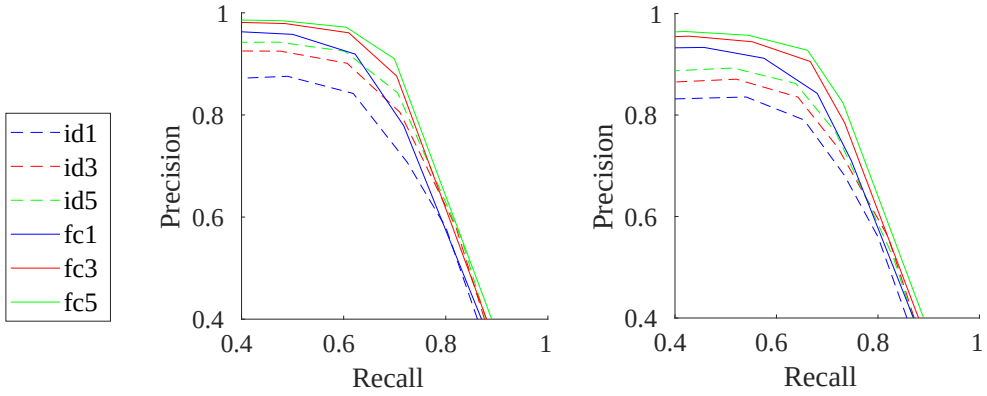
Our freespace check components use the novel freespace querying algorithm we formulated in Chapter 3, which uses a continuous-time trajectory to compensate for the moving-while-scanning operation of spinning-lidars. Here we quantitatively show the benefit by comparing our method to the nearest-ray strategy of Pomerleau et al. (2014) which assumes pointclouds are not motion-distorted.

We formulate a new pipeline variation, *id*, which shares the same structure as the fc pipeline of the previous subsection (i.e., the output is at the step labelled (b) in Figure 3.1). The difference between id and fc is the nearest-ray algorithm. Where fc uses our iterative nonlinear optimization using our continuous-time trajectory, id uses the spherical kd-tree approach of Pomerleau et al. (2014).

The nearest-ray search of id is done by computing the spherical coordinates (i.e., azimuth and elevation, not including range) of the reference freespace scan in its local frame, from which we build an efficient 2D search data structure (e.g., kd-tree) of azimuth and elevation values. Query points of interest are transformed to the local frame of the reference scan, and then also converted to spherical coordinates. Identifying the nearest ray for each query point is completed by simply searching for the nearest neighbour in the search structure. Everything else about the freespace check, such as determining whether the query point is inside, on the border of, or outside freespace remains the

same between the two pipelines. Note that this includes the pointcloud comparison. Both pipelines use the same pointcloud comparison method, which involves undistorting the pointclouds. However, the pointclouds we use to compute spherical coordinates are not motion-compensated.

Figure 5.2 shows the precision-recall curves resulting from the two pipelines on the Town 1 dataset. As in the previous pipeline variation experiment, we also vary the number of reference scans, n_{scans}.



(a) Total (b) Average

Figure 5.2: Precision-recall curves comparing two pipeline variations, id and fc. On the left is the total variation and on the right is the average variation (see Equation (4.1) and (4.2)). The numbers in the legend indicate the number of reference scans, n_{scans}. We see that our method of compensating for motion has better precision. Recall values are similar between the two pipelines.

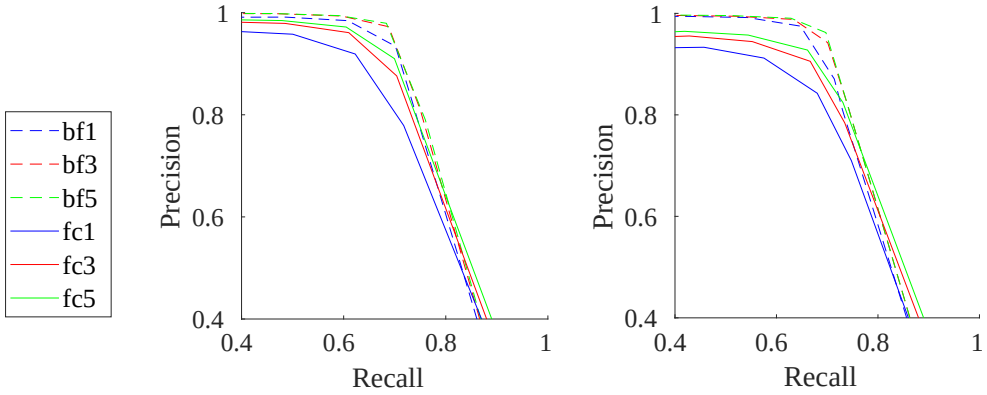
We see that our method has better precision, while recall values are mostly similar. The similarity in recall between the two pipelines is understandable since not compensating for motion will not significantly affect the alignment of subsequent scans (also note we are using the groundtruth trajectory), where the alignment determines the portions of dynamic objects that are labelled dynamic (recall that objects overlap when n_{gap} is not sufficiently large). This is especially true when the lidar vehicle moves at a constant velocity, which happens approximately half the time during the sequence. The increase in incorrect dynamic labels (i.e., FPs) in the id pipeline, causing lower precision, is a result of incorrectly approximating the laser ray paths. The effect of the incorrect ap-

proximations worsen during acceleration, which the sensor vehicles does the other half of the time during the sequence (e.g., traffic intersections).

5.1.3 Box Filter

In Chapter 3 we qualitatively showed the need for filtering incorrect dynamic labels after the freespace checks (i.e., FPs). We accomplish this by introducing a box filter in the lidar imagespace. Here we show the quantitative effect of adding the box filter component after the freespace checks.

We formulate a new pipeline variation, *bf*, which is the same as the fc pipeline, but with the box filter component added. In other words, in the pipeline diagram of Figure 3.1, the output of the bf pipeline is indicated by the letter (c). The score threshold, λ_{filter}, was set to 10. We use the exact filter as shown in Section 3.5. Once again, we also vary the number of reference scans, n_{scans}. The resulting precision-recall curves are shown in Figure 5.3.



(a) Total (b) Average

Figure 5.3: Precision-recall curves comparing two pipeline variations, bf and fc. On the left is the total variation and on the right is the average variation (see Equation (4.1) and (4.2)). The numbers in the legend indicate the number of reference scans, n_{scans}. We see that adding the box filter component increases performance, most notably the precision.

We see an improvement in precision for bf compared to fc, with not much change to the recall. Notice how varying n_{scans} loses its effectiveness in the bf pipeline, evident by

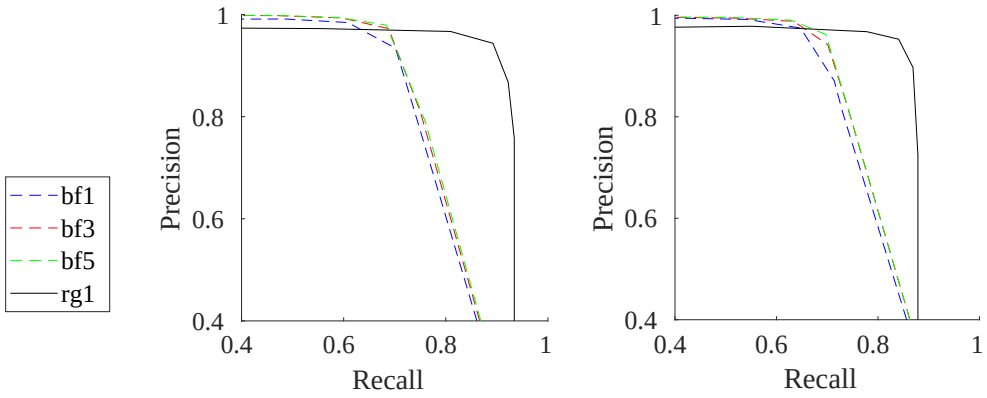
how it is unclear if there is a performance improvement over the progression of increasing n_{scans}. We consider that the detection pipeline may not require n_{scans} greater than 1 with the inclusion of the box filter component, which is desirable as it requires less computation.

5.1.4 Region Growth

In Chapter 3 we established that the combination of pointcloud comparisons and freespace checks is not able to guarantee that dynamic objects will be detected in their entirety. This is due to the effect of the scan gap, n_{gap}, where we cannot choose a desirable value for n_{gap} because the speed and geometry of dynamic objects are not known beforehand. We instead aim to partially label dynamic objects and later use a region growth component to recover the dynamic points mislabelled as static. Previously we showed the effect of region growth qualitatively with example image of a query pointcloud. Here we will show the effect quantitatively.

We compare the pipeline variation that included up to the box filter, bf, with the full detection pipeline, which we denote as *rg*. In other words, the output of the rg pipeline is indicated by the letter (d) in the pipeline diagram of Figure 3.1. We do not vary n_{scans} for rg, setting it to a value of 1. Figure 5.4 shows the precision-recall curves.

Notice the significant improvement in recall using the full pipeline with region growth, where now the upper-right corner of the precision-recall curve is much closer to the upper-right corner of the graph (i.e., indicating good performance). The precision of the full pipeline does decrease slightly compared to the bf pipeline. This is attributed to instances of incorrect region growth. For example, a dynamic cluster made up of FPs will be made larger by the region growth. Instances of incorrect growth to static surfaces due to inaccurate surface normal computations also occur. However, the trade-off is minor. With a slight decrease in precision, the detection pipeline gains significantly in recall for an overall improved performance. The rg pipeline recall for the average curve is



(a) Total (b) Average

Figure 5.4: Precision-recall curves comparing two pipeline variations, bf and rg. On the left is the total variation and on the right is the average variation (see Equation (4.1) and (4.2)). The numbers in the legend indicate the number of reference scans, n_{scans}.

noticeably lower than its total counterpart. Here we clearly see the implication outlined in Section 4.2.2, particularly what is shown in Figure 4.3 – the total computation for recall downplays the neglection of detecting objects at further distances.

5.2 Evaluation of Final Pipeline

In this section we evaluate the full detection pipeline on the entire simulated benchmark and real data. We first evaluate it quantitatively using the simulated benchmark of data obtained from CARLA, described in Chapter 4, using both sequences (i.e., Town 1 and Town 2). We then further evaluate the pipeline qualitatively using a lidar data obtained from a Velodyne HDL-64E mounted on a vehicle and operated in driving scenarios.

5.2.1 Simulated Benchmark

As in Section 3.1, we compute precision-recall curves by varying the error threshold, λ_{error}. We benchmark the full detection pipeline on both data sequences, Town 1 and Town 2, and show the precision-recall curves in Figure 5.5.

Noise was added to range measurements with varying standard deviation. Note that

a Velodyne HDL-64E has a range standard deviation rated less than 0.02 m. The scan gap was set to 4, allowing sufficient object displacement for a 10 Hz lidar. The score threshold was set to 10, which was determined experimentally on data from Town 2. As determined in Section 5.1.3, the inclusion of the box filter component decreases the effect of having more scans in the reference pointcloud during the pointcloud comparison. We set n_{scans} to be 1. Table 5.1 provides a summary of all parameters and their values. We emphasize that these parameters are specific to the lidar of our experiments (recall the simulated lidar was implemented to be identical to the lidar of our real data) and do not depend on the application setting. None of our algorithms in the pipeline make assumptions about the type of objects to detect or the setting. However, this means that the parameters do not generalize to other lidars. Sensor characteristics to consider are the resolution of the measurements, maximum range, and spinrate. Note that for these simulated results, we used the groundtruth trajectory instead of lidar odometry to focus on the detection aspect.

Table 5.1: A list of parameters, their descriptions, and their values.

Parameter	Description	Value
λ_{error}	Threshold on error metric	varied
n_{gap}	Scan gap	4
n_{scans}	Number of reference scans	1
λ_{filter}	Threshold for box filter score	10
r_{neigh}	Radial bound for nearest neighbour search	0.6 m
$\lambda_{\text{parallel}}$	Threshold for parallel surface check	0.8

We have already discussed the effect on recall using the average computation, where it is lower than its total counterpart because it does not downplay the effect of far-away objects as much. We confirm this behaviour in the Town 2 data sequence. Also notice how greater range noise has a more detrimental effect on the total precision and recall compared to the average. Here we see how having more weight on nearby objects (i.e., due to the number of points) results in lower performance values. Nearby surface observations with high noise decreases the quality of our surface normal estimates, most significantly

impacting our region growth algorithm. This further highlights the importance of having both ways of computing precision and recall. Fortunately, we see our detection pipeline is minimally affected by the range noise within a realistic specification, which is less than 0.02 m standard deviation for a Velodyne HDL-64E. There is a noticeable jump between range noise settings of 0.02 m and 0.03 m, possibly due to the value of r_{neigh} (set to 0.06 m), which is used to identify point neighbours for surface normal computation. The increase in noise from 0.02 m to 0.03 m causes a sudden drop in surface normal computation accuracy.

Overall, we perform noticeably worse in the Town 2 sequence compared to Town 1. This is attributed to the structure of the Town 2 simulation setting, which contains more structure that causes partial occlusion instances (e.g., by fences) than Town 1, which we struggle with. Compared to detection methods that use prior data, such as learning-based or map-based methods, we lose out on recall because of the latency in detecting objects accelerating from being stationary because of the scan gap, n_{gap}.

We hope for future comparisons to other works as our dataset is public. For now we make an indirect comparison to Dewan et al. (2017) as the state of the art. They used two manually labelled sequences of Velodyne HDL-64E data at the point level in driving scenarios. The variation of their detection pipeline without deep learning, which is still setting-dependent because of ground point removal, reports the following maximum F1-score precision-recall values: 72.8 precision and 92.3 recall (38 s length data sequence), and 59.5 precision and 69.6 recall (50 s length data sequence). Their results report that adding learning increases precision, but at the cost of recall. They do not distinguish their precision-recall computation in two ways like we did and is unclear about the exact computation. We stress that a fair comparison is not possible since they used real data, but we at least see our total precision-recall values using longer, simulated sequences are comparable.

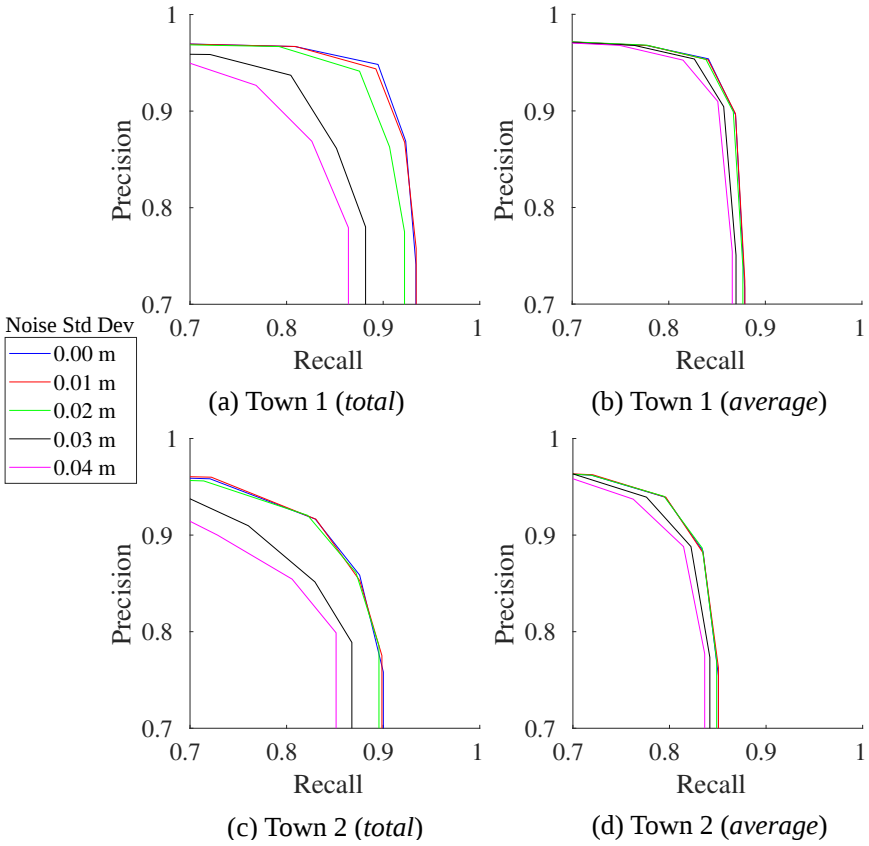


Figure 5.5: Precision-recall plots (*total* and *average* - see Equation (4.1) and (4.2)) on two simulated sequences. The standard deviation of range measurement noise was varied. Note that a Velodyne HDL-64E has a standard deviation rated less than 0.02 m.

5.2.2 Realworld Dataset

Here we provide further evaluation of the detection pipeline using the real lidar dataset described in Chapter 4. Lidar data was collected using a Velodyne HDL-64E mounted on top of a vehicle. Unlike the simulated benchmark evaluation, we apply the continuous-time lidar odometry algorithm presented in Chapter 3 for scan alignment and trajectory construction. The error threshold, λ_{error} was set to 0.5 m. Other parameters were not changed from the simulated benchmark.

An important limitation that was overlooked in the simulated benchmark is the inability to distinguish between measurements that do not return because of the sensor range limit, and measurements that do not return because of poor surface reflection (i.e.,

faulty returns). In both cases, the sensor records a range value of 0. Faulty laser returns often occur on poorly reflecting surfaces, such as dark vehicles (Petrovskaya and Thrun, 2009) or glass windows. Our freespace querying algorithm was designed with the ability to use maximum range measurements as freespace. As it is not obvious how they can be distinguished from faulty measurements, we cannot use maximum range measurements for freespace computation without incorrectly using faulty returns. Fortunately, this is not detrimental for ground-based applications, since spinning-lidar lasers are slightly angled downward to the ground. In other words, there will almost always be a surface (e.g., the ground) behind dynamic objects from the perspective of the sensor. Applications where objects have no geometry behind them for the lidar to perceive (e.g., flying objects) are unfortunately an issue.

Our pipeline works well in scenarios with consistent motion (e.g., no traffic slowdowns). Figure 5.6 is a collage of real data examples, showing 22 different dynamic objects. Our pipeline struggles with occluded objects, which is also reflected in our simulated benchmark as detriments to the computed recall values. An example is row 3, column 6, where we fail to detect the object highlighted with the green box. The pipeline also struggles with inaccurate surface normal computations, causing incomplete region growth (columns 1 and 2), or excessive growth to static points (column 4). The image in row 2, column 6 shows a barely visible vehicle due to many faulty returns. We note there is less concern with newer lidar models, which have more dependable laser returns.

5.3 Summary

In summary, we present the results of the dynamic object detection pipeline in this chapter. Using the detection benchmark we established in Chapter 4 with the simulated lidar datasets from CARLA, we quantitatively evaluated our detection pipeline and discussed the results. This included looking at individual components of the pipeline to show their

significance to the overall detection performance. We further verified through real data collected on a Velodyne HDL-64E that our detection method is not limited to simulated data. From testing on real data, however, we identified that we cannot distinguish between faulty laser returns and maximum range returns, which limits our ability to compute freespace for maximum range returns.

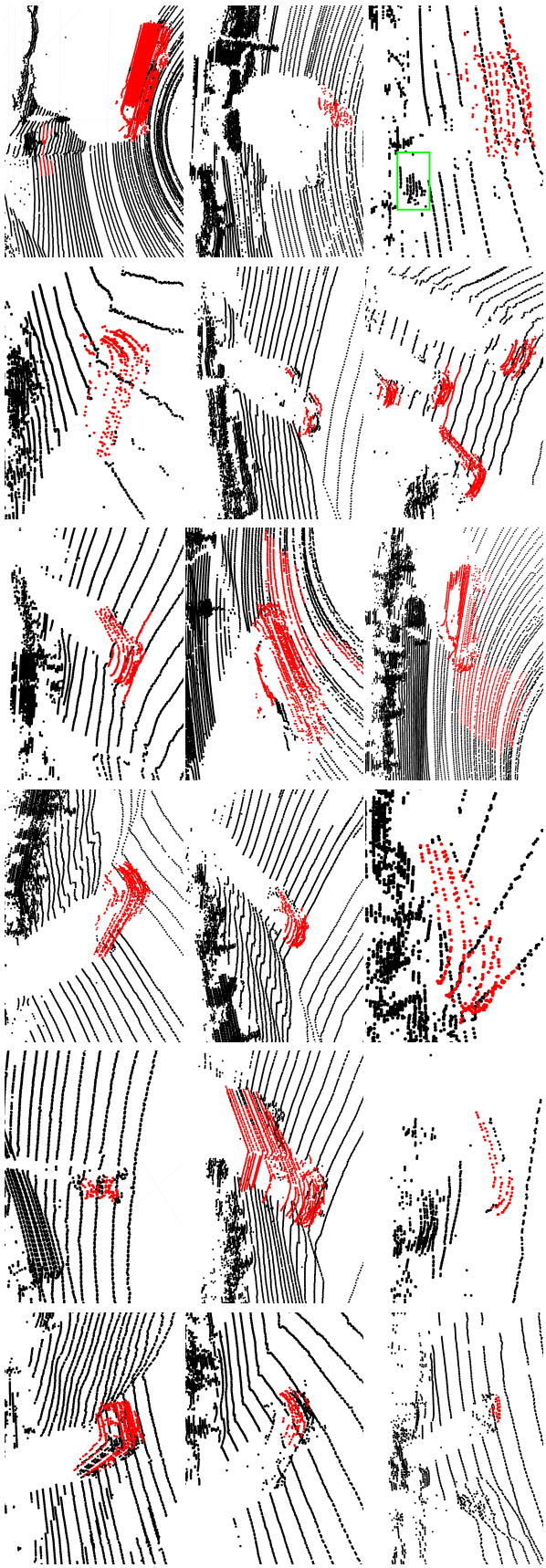


Figure 5.6: Real data pointcloud examples (22 dynamic objects) of our detection method from a Velodyne HDL-64E. False detections can occur (row 1, column 6), but rarely persist in the next scan. Inaccurate surface normals cause incomplete region growth (columns 1 and 2) or growth to static points (column 4). The green box in the last image indicates an unlabelled dynamic object due to occlusion by the other object in the freespace scan(s). The example in row 2, column 6 is a barely visible vehicle due to many faulty returns.

Chapter 6

Conclusion

This thesis presented an online detection method for labeling 3D lidar points as dynamic (moving) or static (stationary). Our goal was to formulate a detection method that does not require prior information on objects (i.e., model-free), the setting (e.g., maps), and even training data (e.g., for learning methods). We accomplished this by using only the most recently acquired lidar scans and made comparisons between them to identify discrepancies that are indicative of dynamic objects. Comparisons of pointclouds alone are not enough as there are issues with spatial sparsity of points and viewpoint occlusions. We addressed these issues by also checking against the freespace of lidar scans using a novel method that accounts for the moving-while-scanning operation of spinning-lidars. A caveat of this approach is the inability to detect objects of interest that are momentarily stationary (e.g., vehicles stopped in traffic). We deemed this as an acceptable trade-off for a detection method that is applicable to a wide variety of scenarios.

Unfortunately, a public lidar dataset suitable to test our work did not exist previously due to the difficulty of obtaining accurate groundtruth labels. We instead established a benchmark of simulated lidar data with point-level groundtruth on dynamic objects (http://asrl.utias.utoronto.ca/datasets/mdlidar/index.html) to produce a quantitative evaluation of our detection method. We make this dataset public in hopes that

other researchers interested in a similar problem can make use of it and compare to our results. We further validated our detection method through a qualitative evaluation using real data collected with a Velodyne HDL-64E.

Apart from evaluating the full detection pipeline, we used the simulated data to quantitatively evaluate and show the significance of the individual components of the pipeline. For example, we showed quantitatively the benefit of compensating for the moving-while-scanning operation when querying the freespace, which other existing detection methods do not consider.

In both simulated and real lidar data, our method found difficulty in detecting partially occluded objects. From operating on real data, we observed an important limitation that was overlooked in simulation – the inability to distinguish between faulty measurement returns and measurements that do not return due to the sensor range limit. The former case we simply want to ignore, while the latter case is desirable to use for freespace computation. This limits our detection method to objects that reliably have geometry behind them (from the perspective of the sensor), which is always the case for ground-based detection, but not for instances such as flying objects in the air.

In summary, the novel contribution of this thesis is a dynamic object detection method that is both model-free and setting-independent, labeling measurements as dynamic or static. Our method explicitly handles the moving-while-scanning distortion effect of 3D spinning-lidar sensors, which existing methods do not consider. While our work is flexible and can be applied to a wide variety of applications, it should not be treated as competitors to learning-based detectors, such as DNN methods to detect class-specific objects, or detectors that make comparisons to maps (e.g., change detection). The methods are complementary and can be combined into a larger detection suite, where our method can act as a safety net. We quantitatively evaluated our work on simulated lidar data, which we made public. We qualitatively evaluated our work with real data from a Velodyne HDL-64E. We spend the rest of this section discussing future work.

6.1 Future Work

We conclude this thesis with a discussion on future work, notably about topics that became apparent during and after evaluating our contributed detection method.

6.1.1 Faulty Lidar Returns

Recall the issue of faulty lidar returns. We refer to a measurement as faulty when the measurement contact surface is within the range capability of the sensor, but does not produce a strong enough return signal due to poor reflectance. Future work should look into developing a method to distinguish them from measurements that do not return due to the maximum range in order to accurately represent the freespace of lidar scans.

Faulty measurements are often a negligible issue in most lidar applications as they are unnoticeable when working with only pointclouds. Consider the pointcloud in the top image of Figure 6.1, which was collected using a Velodyne HDL-64E S3 sensor in a parking lot. In the bottom image, we visualize an imagespace representation of the same lidar scan that created the above pointcloud. In black are the points of the pointcloud above, but in red are measurements that did not return and thus are not shown in the pointcloud. We see that the amount of faulty returns is immense. Entire vehicles, which are still visible in the pointcloud, can be discerned from the bottom image.

The S3 is the latest model among the 64-laser variants from Velodyne. We should take this as a valid issue that occurs from the limitation of the hardware and that it likely will not be resolved in the near future by newer lidar products.

6.1.2 Detection Uncertainty

Our detection pipeline outputs point-level labels between dynamic, for points on moving surfaces, and static, for points on stationary surfaces. What is lacking however, is a measure of uncertainty in the detection labels.

Figure 6.1: The top image is a pointcloud taken from a Velodyne HDL-64E S3 in a parking lot. The bottom image is an imagespace representation of the same lidar scan that produced the pointcloud. In red are measurements that did not return (e.g., faulty measurements). We see from the bottom that there are a large amount of faulty returns, which normally is not realised when working with pointclouds.

Given the detection of dynamic objects in a lidar scan, a follow-up task to accomplish in an autonomous navigation system would be tracking the detections over time in a state estimation framework. Classic state estimation machinery involves the process of reducing state uncertainty using observations which themselves have a measure of

uncertainty. The same applies to the multi-object tracking problem, where the detections can be trated as the observations. If the detections do not have a measure of how reliable they are, it is unclear how they can be used in an estimation framework.

Tracking is not the only use case for detection outputs with uncertainty. Consider the mapping problem, where an accurate and reliable map properly filters out measurements of dynamic objects. A binary label, without an uncertainty measure, of whether measurements should be included or not will work, but is rather ad hoc, and may not be robust. A good mapping framework will take into account uncertainty.

Therefore future work will involve looking into how the current detection pipeline can accurately take into account the measurement uncertainty of the raw lidar measurements for the end-goal of outputting detection labels with an uncertainty measure. The absence of an uncertainty measure in our detection output limits its usefulness in other autonomous navigation tasks.

Appendix A

$SE(3)$ Definitions

We refer to Barfoot (2017) for all the definitions listed here. What is shown is the bare minimum to be able to compute the shown quantities. For further understanding we highly recommend consulting the source literature.

The linear operator $\wedge$ takes a vector, $\boldsymbol{\phi} \in \mathbb{R}^3$, and turns it into a $\mathbb{R}^{3\times3}$ matrix as follows:

$$\boldsymbol{\phi}^\wedge = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix}^\wedge = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix}. \tag{A.1}$$

The operator $\vee$ is the inverse of $\wedge$. We overload the $\wedge$ operator to take a vector, $\boldsymbol{\xi} \in \mathbb{R}^6$, to turn it into a $\mathbb{R}^{4\times4}$ matrix as follows:

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \mathbf{0}^T & 0 \end{bmatrix}, \quad \boldsymbol{\rho}, \boldsymbol{\phi} \in \mathbb{R}^3, \tag{A.2}$$

where once again, $\vee$ is the inverse.

For $\boldsymbol{\phi} = \phi\,\mathbf{a} \in \mathbb{R}^3$, where ϕ is the angle of rotation and $\mathbf{a}$ is the unit-length axis of

rotation, the left Jacobian of $SO(3)$ is

$$\mathbf{J}(\boldsymbol{\phi}) = \frac{\sin\phi}{\phi}\mathbf{1} + \left(1 - \frac{\sin\phi}{\phi}\right) + \frac{1-\cos\phi}{\phi}\mathbf{a}^{\wedge}. \tag{A.3}$$

The left Jacobian of $SE(3)$ is

$$\boldsymbol{\mathcal{J}}(\boldsymbol{\xi}) = \begin{bmatrix} \mathbf{J}(\boldsymbol{\phi}) & \mathbf{Q}(\boldsymbol{\xi}) \\ \mathbf{0} & \mathbf{J}(\boldsymbol{\phi}) \end{bmatrix}, \tag{A.4}$$

where

$$\begin{aligned} \mathbf{Q}(\boldsymbol{\xi}) = \frac{1}{2}\boldsymbol{\rho}^{\wedge} &+ \left(\frac{\phi - \sin\phi}{\phi^3}\right)(\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge} + \boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge} + \boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}) \\ &+ \left(\frac{\phi^2 + 2\cos\phi - 2}{2\phi^4}\right)(\boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge} + \boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge} - 3\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}) \\ &+ \left(\frac{2\phi - 3\sin\phi + \phi\cos\phi}{2\phi^5}\right)(\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge} + \boldsymbol{\phi}^{\wedge}\boldsymbol{\phi}^{\wedge}\boldsymbol{\rho}^{\wedge}\boldsymbol{\phi}^{\wedge}) \end{aligned} \tag{A.5}$$

The GP interpolation in Equation 3.6 has time-dependant coefficient matrices, $\boldsymbol{\Lambda}(\tau)$ and $\boldsymbol{\Omega}(\tau)$. They are defined as

$$\boldsymbol{\Lambda}(\tau) = \boldsymbol{\Phi}(\tau, t_{k-1}) - \boldsymbol{\Omega}(\tau)\boldsymbol{\Phi}(t_k, t_{k-1}), \quad \boldsymbol{\Omega}(\tau) = \mathbf{Q}_k(\tau)\boldsymbol{\Phi}(t_k, \tau)^T\mathbf{Q}_k(t_k)^{-1}, \tag{A.6}$$

where $t_{k-1} \leq \tau < t_k$ and the $\boldsymbol{\Phi}(t, t') \in \mathbb{R}^{12\times 12}$ state transition function

$$\boldsymbol{\Phi}(t, t') = \begin{bmatrix} \mathbf{1} & (t-t')\mathbf{1} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}. \tag{A.7}$$

Bibliography

Anderson, S. and Barfoot, T. D. (2015). Full STEAM ahead: Exactly sparse Gaussian process regression for batch continuous-time trajectory estimation on SE(3). In *Intelligent Robots and Systems (IROS)*, pages 157–164. 11, 20, 22, 24

Atanacio-Jiménez, G., González-Barbosa, J.-J., Hurtado-Ramos, J. B., Ornelas-Rodríguez, F. J., Jiménez-Hernández, H., García-Ramirez, T., and González-Barbosa, R. (2011). Lidar velodyne HDL-64E calibration using pattern planes. *International Journal of Advanced Robotic Systems*, 8(5):59. 6

Azim, A. and Aycard, O. (2012). Detection, classification and tracking of moving objects in a 3D environment. In *Intelligent Vehicles Symposium (IV)*, pages 802–807. 13

Barfoot, T. D. (2017). *State Estimation for Robotics*. Cambridge University Press. 24, 40, 73

Barfoot, T. D. and Furgale, P. T. (2014). Associating uncertainty with 3-D poses for use in estimation problems. *IEEE Transactions on Robotics*, 30(3):679–693. 24

Barfoot, T. D., Tong, C. H., and Särkkä, S. (2014). Batch continuous-time trajectory estimation as exactly sparse Gaussian process regression. In *Robotics: Science and Systems (RSS)*. 22

Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-D shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. 11

Bohren, J., Foote, T., Keller, J., Kushleyev, A., Lee, D., Stewart, A., Vernaza, P., Derenick, J., Spletzer, J., and Satterfield, B. (2008). Little ben: The ben franklin racing team's entry in the 2007 DARPA urban challenge. *Journal of Field Robotics (JFR)*, 25(9):598–614. 5

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30. 10

Carlevaris-Bianco, N., Ushani, A. K., and Eustice, R. M. (2016). University of Michigan north campus long-term vision and lidar dataset. *The International Journal of Robotics Research*, 35(9):1023–1035. 15

Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). Multi-view 3D object detection network for autonomous driving. In *Computer Vision and Pattern Recognition*. 2, 12

Dewan, A., Caselitz, T., Tipaldi, G. D., and Burgard, W. (2016a). Motion-based detection and tracking in 3D lidar scans. In *International Conference on Robotics and Automation (ICRA)*, pages 4508–4513. 13

Dewan, A., Caselitz, T., Tipaldi, G. D., and Burgard, W. (2016b). Rigid scene flow for 3D lidar scans. In *Intelligent Robots and Systems (IROS)*, pages 1765–1770. 13, 14

Dewan, A., Oliveira, G. L., and Burgard, W. (2017). Deep semantic classification for 3D lidar data. In *Intelligent Robots and Systems (IROS)*, pages 3544–3549. 14, 63

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16. 3, 16, 48, 49, 50

Furgale, P. and Barfoot, T. D. (2010). Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560. 1

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research (IJRR)*, 32(11):1231–1237. 11, 13, 15

Hebel, M., Arens, M., and Stilla, U. (2011). Change detection in urban areas by direct comparison of multi-view and multi-temporal ALS data. In *Photogrammetric Image Analysis*, pages 185–196. Springer. 12

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). Octomap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots (AR)*, 34(3):189–206. 10

Jeong, J., Cho, Y., Shin, Y.-S., Roh, H., and Kim, A. (2018). Complex urban lidar data set. In *International Conference Robotics and Automation (ICRA)*. 15

Klassing, K., Wollherr, D., and Buss, M. (2008). A clustering method for efficient segmentation of 3D laser data. In *International Conference on Robotics and Automation (ICRA)*, pages 4043–4048. 44

Ku, J., Mozifian, M., Lee, J., Harakeh, A., and Waslander, S. (2018). Joint 3D proposal generation and object detection from view aggregation. In *Intelligent Robots and Systems (IROS)*. 12

Maddern, W., Harrison, A., and Newman, P. (2012a). Lost in translation (and rotation): Rapid extrinsic calibration for 2D and 3D lidars. In *International Conference on Robotics and Automation (ICRA)*. 6

Maddern, W., Harrison, A., and Newman, P. (2012b). Lost in translation (and rotation): Rapid extrinsic calibration for 2D and 3D lidars. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3096–3102. IEEE. 11

Magnusson, M., Lilienthal, A., and Duckett, T. (2007). Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics*, 24(10):803–827. 11

McGarey, P., Yoon, D., Tang, T., Pomerleau, F., and Barfoot, T. (2018). Field deployment of the tethered robotic eXplorer to map extremely steep terrain. In *Field and Service Robotics (FSR)*. 12, 24

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., et al. (2008). Junior: The Stanford entry in the urban challenge. *Journal of field Robotics (JFR)*, 25(9):569–597. 5

Moosmann, F., Pink, O., and Stiller, C. (2009). Segmentation of 3D lidar data in non-flat urban environments using a local convexity criterion. In *Intelligent Vehicle Symposium (IV)*, pages 215–220. 45

Moosmann, F. and Stiller, C. (2013). Joint self-localization and tracking of generic objects in 3D range data. In *International Conference on Robotics and Automation (ICRA)*, pages 1146–1152. 9, 13, 14

Moravec, H. P. (1988). Sensor fusion in certainty grids for mobile robots. *AI magazine*, 9(2):61. 10

Pandey, G., McBride, J. R., and Eustice, R. M. (2011). Ford campus vision and lidar data set. *The International Journal of Robotics Research*, 30(13):1543–1552. 15

Petrovskaya, A. and Thrun, S. (2009). Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2-3):123–139. 12, 65

Pomerleau, F., Colas, F., Siegwart, R., et al. (2015). A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104. 23

Pomerleau, F., Krüsi, P., Colas, F., Furgale, P., and Siegwart, R. (2014). Long-term 3D

map maintenance in dynamic environments. In *International Conference on Robotics and Automation (ICRA)*. 10, 12, 38, 40, 57

Postica, G., Romanoni, A., and Matteucci, M. (2016). Robust moving objects detection in lidar data exploiting visual cues. In *Intelligent Robots and Systems (IROS)*, pages 1093–1098. 13

Rasmussen, C. E. and Williams, C. K. (2006). Gaussian processes for machine learning. *The MIT Press, Cambridge, MA, USA*. 20, 21

Roynard, X., Deschaud, J., and Goulette, F. (2018). Paris-Lille-3D: A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *International Journal of Robotics Research (IJRR)*, 37(6):545–557. 15

Tang, T. Y., Yoon, D. J., Pomerleau, F., and Barfoot, T. D. (2018). Learning a bias correction for lidar-only motion estimation. In *Computer Robot and Vision*. 12, 24

Underwood, J. P., Gillsjö, D., Bailey, T., and Vlaskine, V. (2013). Explicit 3D change detection using ray-tracing in spherical coordinates. In *International Conference on Robotics and Automation (ICRA)*, pages 4735–4741. 2, 12

Ushani, A. K., Wolcott, R. W., Walls, J. M., and Eustice, R. M. (2017). A learning approach for real-time temporal scene flow estimation from lidar data. In *International Conference on Robotics and Automation (ICRA)*, pages 5666–5673. 14

Zeng, Y., Hu, Y., Liu, S., Ye, J., Han, Y., Li, X., and Sun, N. (2018). RT3D: Real-time 3-D vehicle detection in lidar point cloud for autonomous driving. *IEEE Robotics and Automation Letters*, 3(4):3434–3440. 12

Zhang, J. and Singh, S. (2014). LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, page 9. 11

Zhang, J. and Singh, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *International Conference on Robotics and Automation (ICRA)*, pages 2174–2181. 11

Zlot, R. and Bosse, M. (2014). Efficient large-scale 3d mobile mapping and surface reconstruction of an underground mine. In *Field and Service Robotics (FSR)*, pages 479–493. 11