Understanding a Dynamic World: Dynamic Motion Estimation for Autonomous Driving using LIDAR

by

Arash K. Ushani

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Ryan M. Eustice, Chair
Associate Professor Jason J. Corso
Associate Professor Odest C. Jenkins
Professor Benjamin J. Kuipers

Arash K. Ushani

aushani@umich.edu

ORCID iD: 0000-0002-4308-1141

ACKNOWLEDGMENTS

Whew! Over the past five years, I have been so focused on getting through the PhD program that it is only now that I have the chance to reflect on it all. It has been quite the journey. I certainly would never have made it this far were it not for so many people.

First of all, I am deeply grateful to Ryan, my advisor. I am not sure exactly what you saw in that wide-eyed kid, but with your help and nurturing, he has made it to the other side. I remember in one of our first conversations, I asked about which courses to take; you responded with a few suggestions that had a “high dot product” with the work I would be doing. After hearing that phrase, I remember thinking “man, I must really be in the right place.” Thank you for making me feel welcome all those years ago, and thank you for the continued guidance that has allowed me to make it this far.

Next, I would like to thank my committee — Professor Corso, Professor Jenkins, and Professor Kuipers. Thank you for asking the hard questions, providing feedback on research and direction, and your advice and conversations regarding my goals.

Additionally, I am also grateful to the Ford Motor Company and the Toyota Research Institute for funding this work under awards N015392 and N021515.

As anyone who has been in grad school can attest, the experience is much more than the research and work you do. When I’m old and gray, the fond memories I’ll look back on won’t be debugging code or making paper revisions, but the people I met along the way.

Thank you to all the members of PeRL who have toiled in the trenches with me: Ayoung, Gaurav, Schuyler, GT, Nick, Enric, Jeff, Paul, Vittorio, Wolcott, Stephen, Steve, Alex, Derrick, Jie, Steven, Josh, Lu, James, and Maani. It is humbling to think that the group that welcomed me when I joined is disjoint to the group that I will say good bye to, but I hope the tradition of camaraderie will live on.

Thank you to all my frisbee friends from HAC, Burns Park, Ann Arbor league, Rocket Lawnchair, and Autonomous. No matter how stressful my academic life was, I always looked forward to playing with you.

Thank you to all the members of PC (an acronym which is intentionally left unexpanded) over the years: Katie, Nina, Alexa, Nils, Steven, Nader, and Eric. I will always fondly look back on grilling and bike rides. More often than not, I was the slowest in our bike gang, so I appreciate you guys (usually) not dropping me. And Steven, thanks for crashing into me only ~~once~~ twice!

I have been blessed to be part of such a large and loving family. Thank you for all of your continued support over the past 28+ years. If I were to cover everyone and everything that I am grateful for, this document would never end, so please forgive me for being brief.

Baba Taji, I will always be inspired by what you were able to accomplish despite only six years of schooling. If I find a tenth of the success you have had, I will consider myself a very lucky man. Maman Taji, you were the one to witness my first words. I always think of you when I look to the moon. Baba Ali, thank you for always letting me be your scientist. I wish that the cars I've worked on would sing and dance like you. Maman Farkhondeh, we always joke about tea and ice cream. But in reality, sharing a laugh with you is what fills me up best.

Aria, there is so much I can say, so let me just say this. You are not the brother I deserve, but the one I need.

Baba, I remember you finishing your own PhD when I was in elementary school. Soon after, you were the first to introduce me to all of this. To borrow from Patrick Rothfuss, "little did I know how cunningly I was being taught." Without you, I never would have set off on this path.

And Maman. I know Baba says that his greatest fortune in life is being your husband, but I have been far more fortunate to be your son.

TABLE OF CONTENTS

Acknowledgments . . . . **ii**

List of Figures . . . . **vii**

List of Tables . . . . **ix**

List of Appendices . . . . **x**

List of Acronyms . . . . **xi**

Abstract . . . . **xiii**

Chapter

1 Introduction **1**

1.1 LIDAR Sensors 2
1.2 NGV Platform 4
1.3 Datasets 5
1.3.1 KITTI 5
1.3.2 Other Datasets 7
1.4 Object Tracking 8
1.4.1 Early Tracking Methods 8
1.4.2 Model-Based Methods 10
1.4.3 Model-Free Methods 12
1.5 Flow Estimation 14
1.5.1 Optical Flow 14
1.5.2 Scene Flow 15
1.5.3 Flow from LIDAR 17
1.6 Object and Scene Understanding 17
1.6.1 Reconstruction, Semantic Inpainting, and Scene Completion 18
1.6.2 Features for Scene Understanding from LIDAR 20
1.7 Autonomous Vehicle Applications 21
1.8 Thesis Outline 21
1.9 Thesis Roadmap 23

2 Continuous-Time Estimation for Dynamic Object Tracking **25**

2.1 Introduction 25
2.2 Related Work 28

2.3 Problem Statement 29
2.4 Front End 31
2.5 Method 32
2.5.1 Formulation 32
2.5.2 Continuous-Time Estimation 35
2.5.3 Gauss-Newton 37
2.6 Experimental Results 39
2.6.1 Setup 39
2.6.2 Parameter Selection 39
2.6.3 Pose Results 41
2.6.4 Point Cloud Crispness 41
2.6.5 Convergence 42
2.6.6 Runtime 42
2.7 Discussion 43
2.8 Conclusion 45

3 Real-Time Temporal Scene Flow Estimation 48

3.1 Introduction 48
3.2 Related Work 50
3.3 Problem Statement 52
3.4 LIDAR Preprocessing 53
3.4.1 Occupancy Grid 53
3.4.2 Background Filter 54
3.5 Raw Scene Flow Computation 56
3.5.1 Occupancy Constancy 56
3.5.2 Raw Scene Flow Computation 58
3.5.3 GPU Implementation 61
3.6 Temporal Scene Flow 62
3.6.1 Flow Tracklets Filter 63
3.6.2 Flow Tracklets Array 64
3.7 Motion Compensation 66
3.8 Results 66
3.8.1 Parameter Selection 66
3.8.2 Background Filter 66
3.8.3 Raw Scene Flow 67
3.8.4 Temporal Scene Flow 69
3.8.5 Motion Compensation 74
3.8.6 Runtime Performance 74
3.9 Discussion 76
3.10 Conclusion 77

4 Feature Learning for Scene Flow Estimation 78

4.1 Introduction 78
4.2 Related Work 79
4.2.1 Dynamic Object Tracking 79

4.2.2 Optical Flow and Scene Flow 80
4.2.3 Object and Scene Understanding 81
4.2.4 Feature Learning 83
4.3 Motivation and Toy Example 83
4.3.1 Network 84
4.3.2 Motivating Results 85
4.4 Method 86
4.4.1 Input 87
4.4.2 Network 89
4.4.3 Training 91
4.4.4 Flow Computation 92
4.5 Results 94
4.5.1 Visualizing the Learned Feature Space 94
4.5.2 Learned Feature Space vs. Occupancy Constancy 96
4.5.3 Scene Flow Results 96
4.6 Conclusion 100

5 Conclusion 101

5.1 Contributions 101
5.2 Future Work 102
5.2.1 Improved Object Modeling 102
5.2.2 Reasoning about Occlusions and Temporal Features 103
5.2.3 Incorporating Semantics 103

Appendices 105

Bibliography 115

LIST OF FIGURES

1.1 A LIDAR sensor . . . 3
1.2 The autonomous vehicle platform . . . 4
1.3 Sample data from B1 . . . 5
1.4 Sample data from the KITTI dataset . . . 6
1.5 A cartoon depiction of the tracking problem . . . 9
1.6 The perspective problem in tracking . . . 10
1.7 The occlusion problem in tracking . . . 11

2.1 A point cloud generated by our proposed dynamic object tracker . . . 26
2.2 A cartoon example of the problem setup . . . 30
2.3 An illustration of our measurement model . . . 34
2.4 A sample function represented by B-spline basis functions . . . 38
2.5 The average residual after each iteration over the stationary and dynamic set . 43
2.6 Our proposed method tracking a person riding a bicycle . . . 44
2.7 Sample velocity and heading profiles . . . 45
2.8 Our proposed method tracking a bus, top-down view . . . 46
2.9 A fault case of our proposed tracker . . . 47

3.1 An overview of our temporal scene flow pipeline . . . 49
3.2 Background filter feature vectors . . . 54
3.3 Occupancy constancy feature vectors . . . 57
3.4 An overview of the raw scene flow computation process. . . . 59
3.5 An overview of the temporal scene flow filtering process. . . . 65
3.6 Precision-recall curve of the background filter . . . 67
3.7 Error histograms for raw scene flow measurement by class . . . 68
3.8 Error histogram for raw scene flow measurement for all foreground . . . 69
3.9 The CDF of the error of our temporal scene flow estimates . . . 71
3.10 Sample temporal scene flow results when the platform is stationary . . . 72
3.11 Sample temporal scene flow results when the platform is moving . . . 73
3.12 An example of motion compensation for temporal scene flow . . . 75

4.1 The encoding and decoding networks used for the MNIST toy example. . . . 84
4.2 Learned latent encoding for MNIST using the autoencoder approach . . . 86
4.3 Learned feature representations for MNIST using the feature learning approach 87
4.4 The encoder network architecture . . . 90
4.5 The loss function $L_{\mathbf{f}_1,\mathbf{f}_2}$ for matching and non-matching locations . . . 91

4.6 A visual look at the distances in the feature space for a turning car 95
4.7 A visual look at the distances in the feature space for two pedestrians 98
4.8 Performance of classifiers based on occupancy constancy and feature learning . 99

A.1 An overview of GPU functionality and features 107

B.1 Sample point cloud and occupancy grid from the KITTI dataset 111
B.2 Cartoon example of ray tracing 113

LIST OF TABLES

2.1 The tracking performance of our proposed tracker 41
2.2 Point cloud entropy of the models generated by our proposed tracker 42

3.1 Error statistics for raw scene flow measurements 70
3.2 Error statistics for temporal scene flow measurements 70
3.3 Runtime performance of our proposed temporal scene flow method 76

4.1 MNIST classification results 88
4.2 Error statistics for the scene flow estimate. 99

A.1 Key GPU specificiations 109

B.1 Parameter values for occupancy grid construction 114

LIST OF APPENDICES

A GPU Programming 106

B Efficient Construction of Occupancy Grids using a GPU 110

LIST OF ACRONYMS

2D two-dimensional

3D three-dimensional

4D four-dimensional

API application programming interface

CDBN convolutional deep belief network

CDF cumulative distribution function

CPU central processing unit

CRF conditional random field

CNN convolutional neural network

CUDA Compute Unified Device Architecture

DATMO detection and tracking of moving objects

DT deep tracking

EKF extended Kalman filter

EM expectation-maximization

GAN Generative Adversarial Network

GNN global nearest neighbor

GPS global positioning system

GPU graphics processing unit

ICP iterative closest point

INS inertial navigation system

LBP loopy belief propagation

LIDAR light detection and ranging

LSTM long short-term memory

MAP maximum *a posteriori*

MCMC Markov chain Monte Carlo

MHT Multi-Hypothesis Tracking

MLE maximum likelihood estimate

MRF Markov random field

NGV Next Generation Vehicle

RGB-D red, green, blue, and depth

RNN recurrent neural network

SHOT signature of histograms

SIFT scale-invariant feature transform

SIMD single instruction multiple data

SLAM simultaneous localization and mapping

STL Standard Template Library

UKF Unscented Kalman filter

ABSTRACT

In a society that is heavily reliant on personal transportation, autonomous vehicles present an increasingly intriguing technology. They have the potential to save lives, promote efficiency, and enable mobility. However, before this vision becomes a reality, there are a number of challenges that must be solved. One key challenge involves problems in dynamic motion estimation, as it is critical for an autonomous vehicle to have an understanding of the dynamics in its environment for it to operate safely on the road. Accordingly, this thesis presents several algorithms for dynamic motion estimation for autonomous vehicles. We focus on methods using light detection and ranging (LIDAR), a prevalent sensing modality used by autonomous vehicle platforms, due to its advantages over other sensors, such as cameras, including lighting invariance and fidelity of 3D geometric data.

First, we propose a dynamic object tracking algorithm. The proposed method takes as input a stream of LIDAR data from a moving object collected by a multi-sensor platform. It generates an estimate of its trajectory over time and a point cloud model of its shape. We formulate the problem similarly to simultaneous localization and mapping (SLAM), allowing us to leverage existing techniques. Unlike prior work, we properly handle a stream of sensor measurements observed over time by deriving our algorithm using a continuous-time estimation framework. We evaluate our proposed method on a real-world dataset that we collect.

Second, we present a method for scene flow estimation from a stream of LIDAR data. Inspired by optical flow and scene flow from the computer vision community, our framework can estimate dynamic motion in the scene without relying on segmentation and data association while still rivaling the results of state-of-the-art object tracking methods. We design our algorithms to exploit a graphics processing unit (GPU), enabling real-time performance.

Third, we leverage deep learning tools to build a feature learning framework that allows us to train an encoding network to estimate features from a LIDAR occupancy grid. The learned feature space describes the geometric and semantic structure of any location observed by the LIDAR data. We formulate the training process so that distances in this learned feature space are meaningful in comparing the similarity of different locations. Accordingly, we demonstrate that using this feature space improves our estimate of the dynamic motion in the environment over time.

In summary, this thesis presents three methods to aid in understanding a dynamic world for autonomous vehicle applications with LIDAR. These methods include a novel object tracking algorithm, a real-time scene flow estimation method, and a feature learning framework to aid in dynamic motion estimation. Furthermore, we demonstrate the performance of all our proposed methods on a collection of real-world datasets.

CHAPTER 1

Introduction

Our society today is heavily dependent on personal transportation. In the United States alone, there are over four million miles of highways as of 2018 and an additional 4.1 million miles of other roads as of 2016 (Beningo et al., 2018). Personal vehicles accounted for over a trillion dollars of household transportation costs in 2016, which represented over 90% of total household transportation expenditures in the United States (Firestine, Notis, and Randrianarivelo, 2018).

Unfortunately, our extensive reliance on personal transportation is not without drawbacks. Over the course of more than three trillion vehicle miles traveled in the United States, more than six million crashes occurred in 2015 (National Highway Traffic Safety Administration, 2018). These crashes resulted in over 35,000 deaths and over two million injuries with an estimated cost of about 242 billion dollars in property damage.

One potential way to help mitigate these issues is the application of robotic technologies. Not only could a fully robotic self-driving vehicle help prevent fatalities, injuries, and property damage, but it would also enable mobility for those who cannot drive, whether due to disability, age, or other factors. There are over 100 million people in the United States who are not licensed to operate a motor vehicle (National Highway Traffic Safety Administration, 2018).

Indeed, as robotic technology has progressed, this vision is slowly becoming a reality. In recent years, many groups in academia and industry have launched significant self-driving vehicle projects, including Ford Motor Company, Toyota Research Institute, Waymo, and others. This rapidly growing industry is estimated to be as large as 7 trillion dollars (Lanctot, 2017).

A number of challenges must be solved in order to develop vehicles that can drive safely and smartly on our roads alongside human drivers. These challenges include problems in controls, planning, perception, and mapping. Among these problems, it is critical that an autonomous vehicle be able to detect and understand the dynamics of the world around it.

Any autonomous vehicle will need to robustly understand the motion of dynamic objects in its operating environment, such as other cars, cyclists, and pedestrians that are on the road.

Accordingly, this thesis focuses on advancing the state-of-the-art in the perception of dynamic motion for autonomous vehicle applications. First, we propose a method to jointly estimate the trajectory and appearance of a dynamic object from an array of light detection and ranging (LIDAR) sensors. Next, we propose a method for estimating dynamic motion in the surrounding environment directly from LIDAR scans, without relying on a segmentation result or data association through time. Finally, we propose an encoding network to learn features in a dynamic scene that can be used to improve the motion estimate within the environment. We demonstrate the performance of these systems on real-world data collected from autonomous vehicle platforms.

The rest of this chapter is organized as follows. First, we present an overview of sensors, platforms, and datasets used in this area of work in Section 1.1, Section 1.2, and Section 1.3, respectively. Next, we review literature in object tracking in Section 1.4, flow estimation in Section 1.5, and object and scene understanding in Section 1.6. Finally, we present an outline of contributions in Section 1.8 and a roadmap for the rest of this thesis in Section 1.9.

1.1 LIDAR Sensors

The work presented in this thesis focuses on LIDAR sensor data. A sample LIDAR sensor can be seen in Fig. 1.1. The depicted LIDAR sensor operates by using a spinning platform to shine an array of lasers onto the environment. While different LIDAR sensors use different configurations, the underlying principle of these sensors is similar. A laser beam strikes an object in the world and its reflection is detected by the sensor. By measuring time-of-flight, the sensor determines the distance between it and the reflecting object. This distance then results in a three-dimensional (3D) point observation by accounting for the orientation of the laser. As the array of lasers spins, many data points are collected by the sensor. These points are accumulated, resulting in a point cloud. Sample point clouds collected from various platforms can be seen in Fig. 1.3 and Fig. 1.4.

Additionally, a LIDAR sensor can measure the intensity of the reflection of the laser beam. While the measurement of intensity may be prone to errors due to calibration, incidence angle, or lighting conditions, it can be of use in certain applications. For example, the map of the road shown in Fig. 1.3 was generated using a map built from this intensity data.

Camera systems have also been used for applications and problems similar to those investigated in this thesis, and indeed some of the presented work is inspired by techniques in computer vision. However, each sensor modality has certain advantages and disadvantages.

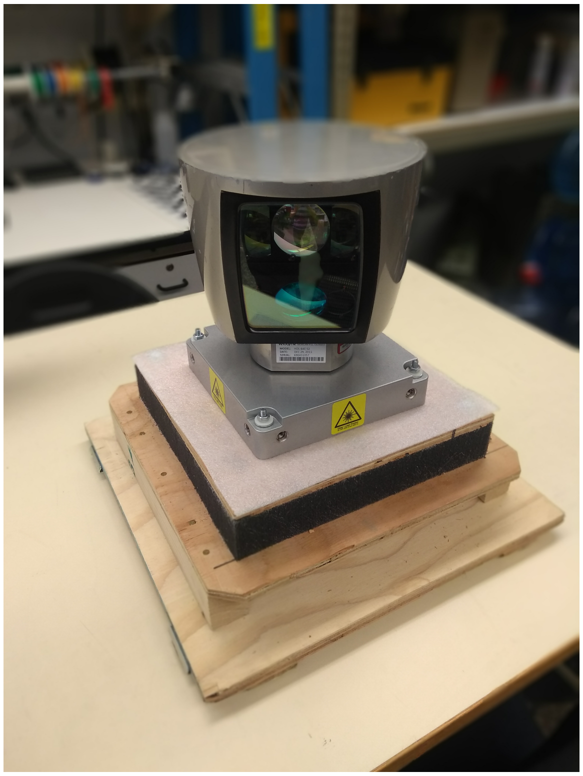


Figure 1.1: A LIDAR sensor. This is a Velodyne laser scanner with 64 laser beams.

A LIDAR sensor can directly observe 3D information whereas, in computer vision, 3D information is not directly observed. If needed, it must be estimated through other means, such as a binocular or stereocamera system, or with active depth sensing, such as the red, green, blue, and depth (RGB-D) Kinect sensor, although the range of LIDAR sensors typically is superior. Despite the advantage of LIDAR systems in 3D geometry, camera systems provide superior appearance data. RGB color is much more descriptive and reliable than the intensity or reflectivity observed by LIDAR, especially when not accumulated over time. However, camera systems are dependent on a source of light, and indeed the observed appearance can change dramatically with lighting changes. Because LIDAR is an active sensing modality, it is almost entirely independent to changes in ambient lighting.

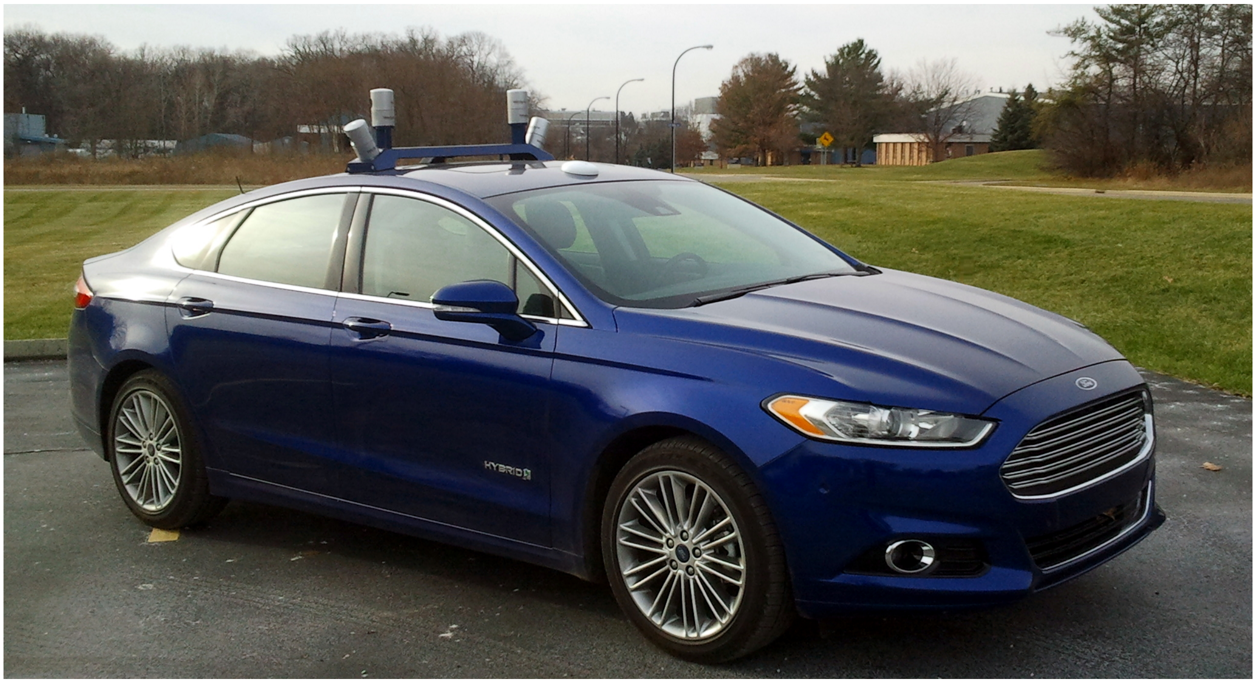


Figure 1.2: The autonomous vehicle platform. The vehicle pictured here is known as B1.

1.2 NGV Platform

The Next Generation Vehicle (NGV) project was a collaboration between the University of Michigan and the Ford Motor Company from 2012 to 2016. The goal of the project was to develop technologies to help enable autonomous vehicles. The NGV project investigated research problems in areas including planning (Cunningham et al., 2015; Galceran et al., 2015), localization (Wolcott and Eustice, 2014, 2015), and perception (Ushani et al., 2015). One platform used in this project, called B1, is shown in Fig. 1.2. The main sensors used in the work presented in this thesis include four Velodyne HDL-32E 3D LIDAR scanners spinning at roughly 10 Hz and an Applanix POS-LV 420 inertial navigation system (INS) with a global positioning system (GPS) used for pose estimation. A compute cluster consisting of four PCs was located in the trunk of the car and used during online operation of the platform.

A sample of data taken from B1 can be seen in Fig. 1.3. Data from the four LIDAR sensors is accumulated over about 100 ms. This data is motion compensated for the movement of the platform vehicle during that time. This accumulated point cloud is then shown in Fig. 1.3, overlaid on a road map built from LIDAR intensity data. The displayed point cloud is shown from data collected in Ann Arbor, Michigan, USA.

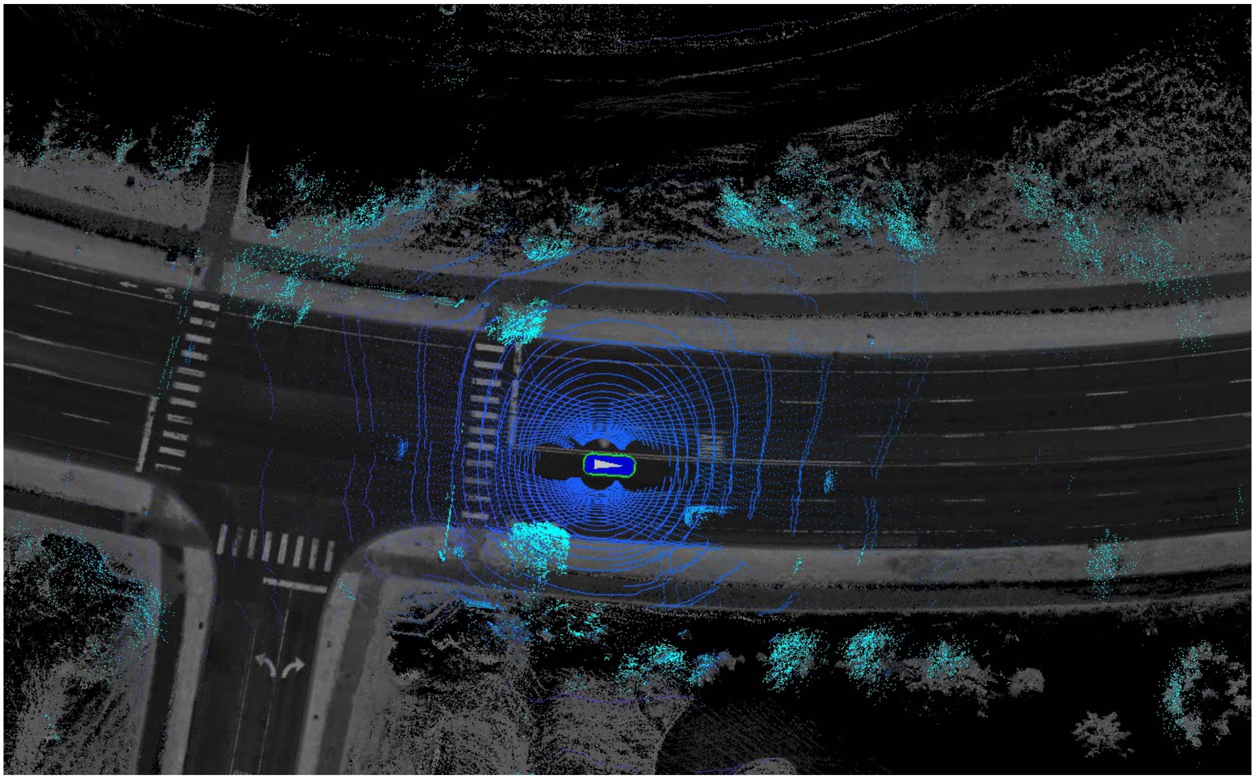


Figure 1.3: Sample data from B1. The point cloud collected by the LIDAR sensors is shown in blue from a bird's eye perspective. The platform vehicle is in the center. The colormap represents height above the ground. The rendering of the road and its associated lane markings was generated from a large scale map using the intensity data from the LIDAR sensors.

1.3 Datasets

In order to facilitate research, a number of datasets have been made publicly available by the community. These datasets provide sensor data that can be used for a variety of applications.

1.3.1 KITTI

Geiger et al. (2013) presented the KITTI dataset, collected by an autonomous vehicle platform driving in Karlsruhe, Germany. The sensor data that is used by our proposed algorithms is LIDAR data collected by a Velodyne HDL-64E, and inertial and GPS data collected by a OXTS RT3003. Additionally, the KITTI dataset provides hand labeled tracks for various obstacles, including cars, pedestrians, and bicyclists, for a number of sequences of data. These tracks provide ground truth position and orientation for obstacles over time, which can be used for training and evaluation. The dataset also includes camera data collected by two

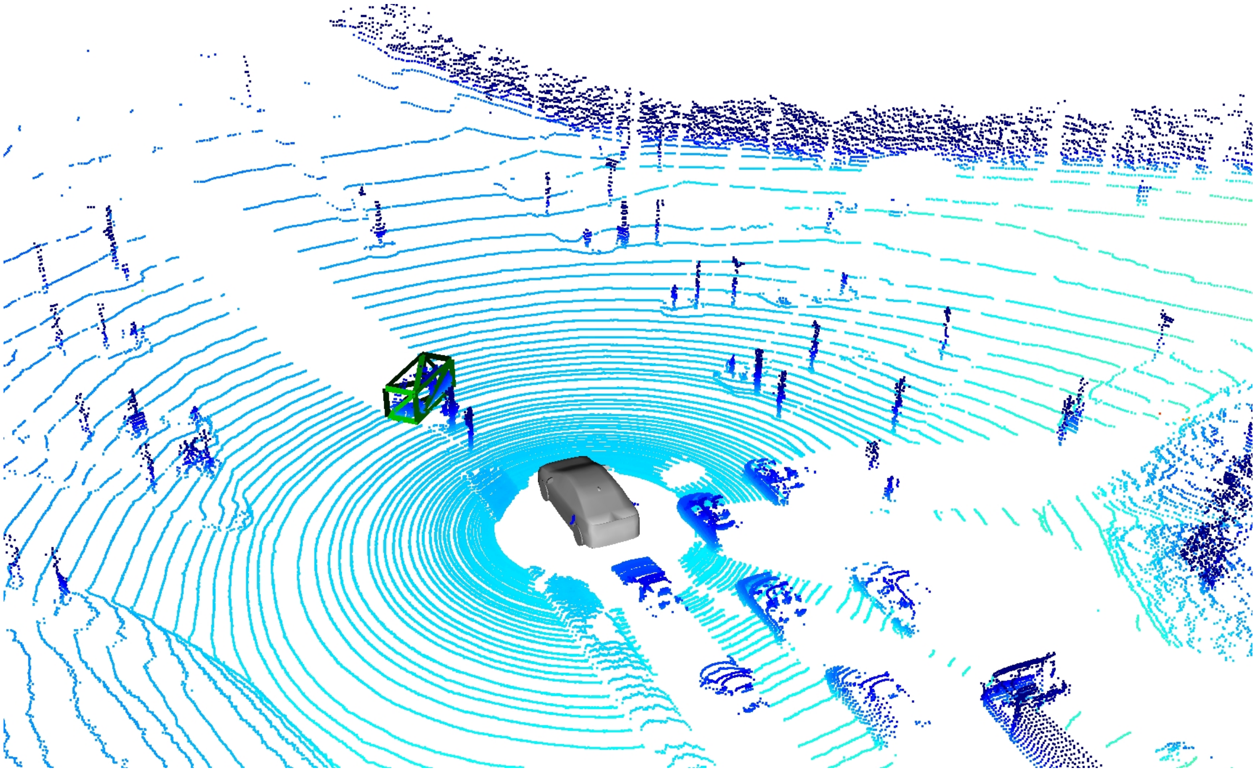


Figure 1.4: Sample data from the KITTI dataset. The platform vehicle is shown as the gray model in the center of the image. The point cloud collected by the LIDAR sensor is shown in blue. The colormap represents height above the ground, where cyan is the lowest and dark blue is the highest. The bounding box shows a labeled KITTI track.

PointGray Flea2 stereo-cameras, one color and one grayscale.

The KITTI dataset was chosen mainly for its availability of extensive manually labeled ground truth data for dynamic objects in the LIDAR data. Furthermore, the KITTI dataset is widely used as a benchmark for many tasks in the area of autonomous driving and perception.

Sample data from the KITTI dataset can be seen in Fig. 1.4. In the KITTI dataset, a single frame from the LIDAR sensor is given per timestep, consisting of data from one full revolution of the sensor. This is rendered in Fig. 1.4, colored by height above the ground plane.

1.3.2 Other Datasets

While we focus on the KITTI data for the work presented in this thesis, there are a number of other datasets used by the community.

The New College Vision and Laser Dataset (Smith et al., 2009) provides stereo vision, omnidirectional vision, and LIDAR from a single 2.2 km session. The Rawseeds project (Ceriani et al., 2009) provides two datasets with omnidirectional and stereo vision, and planar laser; both datasets have repeated sessions through the same environment. The CMU Visual Localization Dataset (Badino, Huber, and Kanade, 2011) includes monocular vision from 16 sessions covering the same trajectory over the course of a year. The Ford Campus Vision and LIDAR Dataset (Pandey, McBride, and Eustice, 2011) provides three sessions with omnidirectional vision and 3D LIDAR. The Alderley Day/Night Dataset (Milford and Wyeth, 2012) contains two sessions on the same route, one collected during the day and one at night. The Nordland Dataset (Norwegian Broadcasting Corporation, 2013), promoted by Sunderhauf, Neubert, and Protzel (2013), contains monocular vision for four 3000 km sessions collected in each of the four seasons—because the data was collected from a train, each session follows exactly the same trajectory. The Malaga Urban Dataset (Blanco-Claraco, Moreno-Duenas, and Gonzalez-Jimenez, 2014) contains a single trajectory with stereo vision and planar LIDAR. The VPRiCE Challenge Dataset (Suenderhauf, 2015) provides two sets of imagery aimed toward place recognition contests. The Cross Season Dataset (Masatoshi et al., 2015) provides imagery on a university campus once per each of four seasons. The MIT Stata Center Dataset (Fallon et al., 2013) provides multiple sessions with stereo vision over the course of a year, with 38 hours and 42 km of repeated exploration. The North Campus Long Term (NCLT) Dataset was collected by a Segway platform that was manually driven around the University of Michigan's North Campus over the course of 27 sessions spread over 15 months (Carlevaris-Bianco, Ushani, and Eustice, 2015). The sensors include a Velodyne HDL-32E LIDAR scanner, a 3DM-GX3-45 IMU system, a GPS, and a Labybug3 camera. Additionally, the dataset provides a high-rate ground truth pose estimate for the Segway computed using simultaneous localization and mapping (SLAM).

1.4 Object Tracking

Consider a robot operating in a dynamic environment. An object moving in that environment can be detected by the robot's sensors, such as a camera or LIDAR sensor. As the dynamic object moves along a path, the robot records observations of that object at different times and locations. From these observations, our goal is to estimate the position or full trajectory of the dynamic object. A secondary objective, which can be used to help estimate the trajectory, is to build some sort of model of what the object looks like. A cartoon example of this problem is shown in Fig. 1.5.

More formally, there is a dynamic object that we are tracking with our sensors. We wish to estimate the state of this object over time, $\mathbf{x}(t)$, and build a representation of its appearance, M, from a set of observations from our sensors, $\mathrm{Z} = \{\mathbf{z}_{1:n_z}\}$. This can commonly be represented as an maximum *a posteriori* (MAP) problem,

$$\mathbf{x}(t)^\star, \mathrm{M}^\star = \operatorname*{argmax}_{\mathbf{x}(t),\mathrm{M}} p(\mathbf{x}(t), \mathrm{M} \,|\, \mathrm{Z}). \tag{1.1}$$

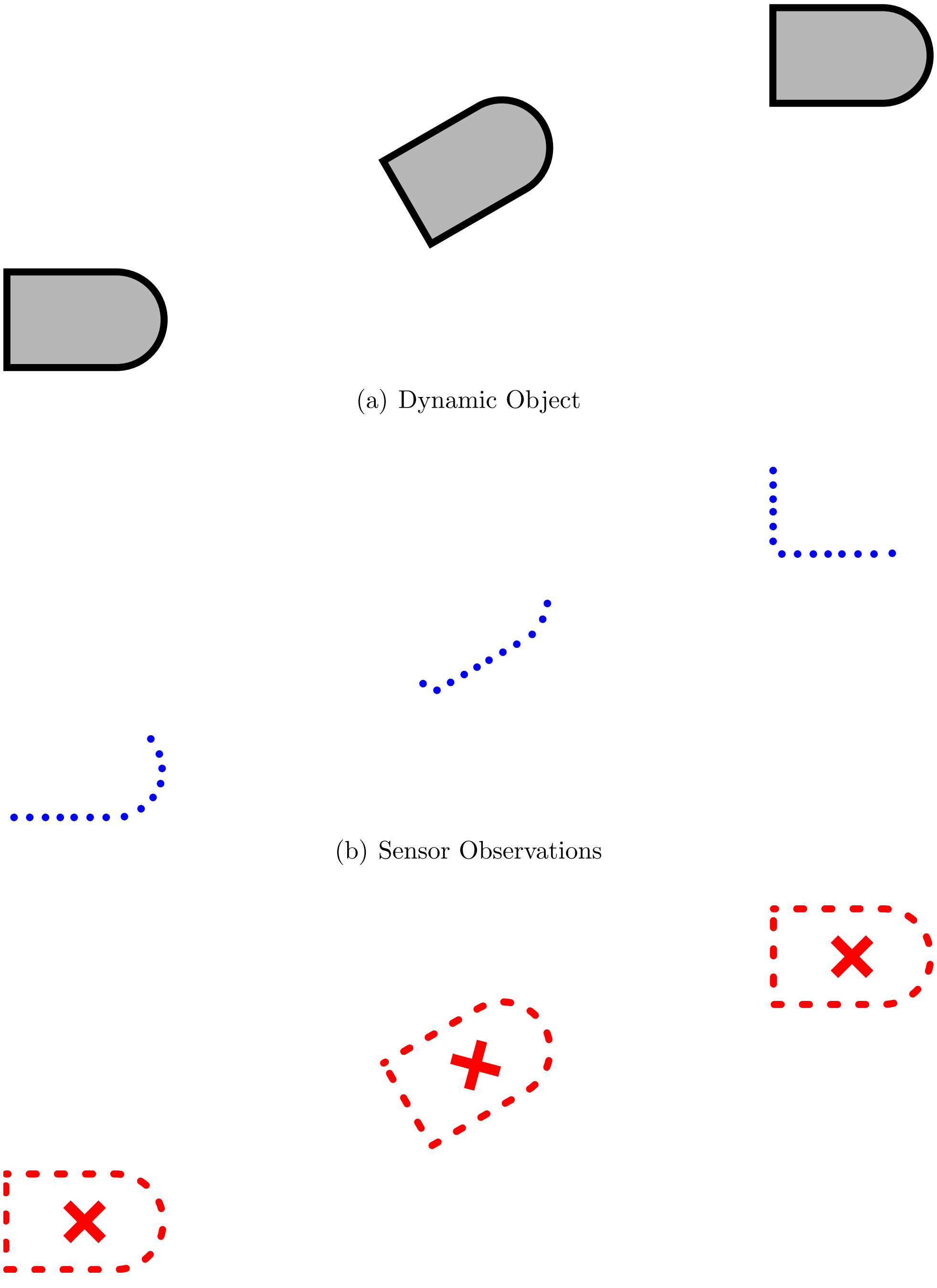
$\mathbf{x}(t)$ can be a discrete set of states at each timestep, $\mathbf{x}_1, \ldots, \mathbf{x}_n$, or a continuous function. M can take a wide variety of forms, from simple models like bounding boxes, to more expressive models such as point clouds or mesh models. Depending on the application, the object's model, M, may not be required to be estimated at all.

1.4.1 Early Tracking Methods

The object tracking problem has been studied extensively for many years. Various approaches were developed for various applications including vehicle tracking, pedestrian monitoring, and aerospace target tracking. Methods have been developed using various sensors such as LIDAR, radar, or camera systems. Many of these early algorithms would use the same three main steps.

First, sensor data is segmented and dynamic objects are identified. Many methods would segment the scan into clusters or connected components (Kaestner et al., 2012; Leonard et al., 2008; Streller, Furstenberg, and Dietmayer, 2002; Wender and Dietmayer, 2008). Some other techniques would extract features to detect objects. For example, Zhao and Thorpe (1998) would detect line segments and use these with the Hough Transform to detect vehicles.

Then, data association is performed between the new observations and the objects that were previously tracked. Many methods rely on global nearest neighbor (GNN) or variants of it for data association (Aycard et al., 2006; Vu, Aycard, and Appenrodt, 2007). Azim



(a) Dynamic Object

(b) Sensor Observations

(c) Tracking Estimate

Figure 1.5: A cartoon depiction of the tracking problem. In this example, there is an object moving in the environment over time, shown in Fig. 1.5(a). Over time, our sensors provide us with a set of observations from the object, shown in Fig. 1.5(b). Our goal is to to estimate the position of the object and a model of what it looks like, shown in Fig. 1.5(c).

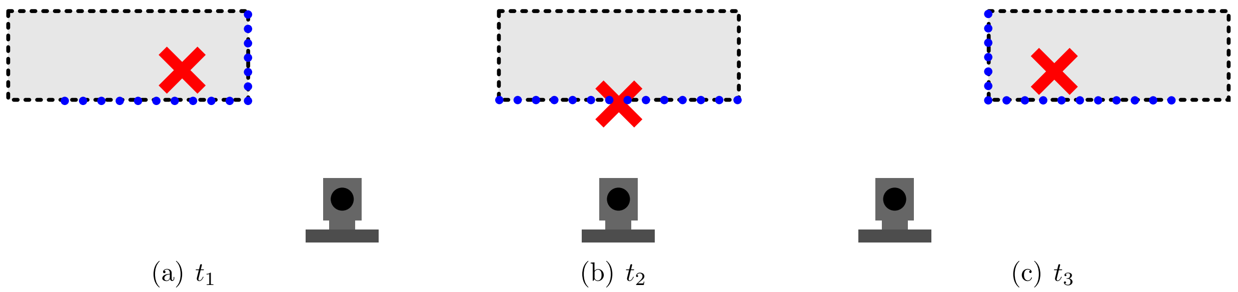


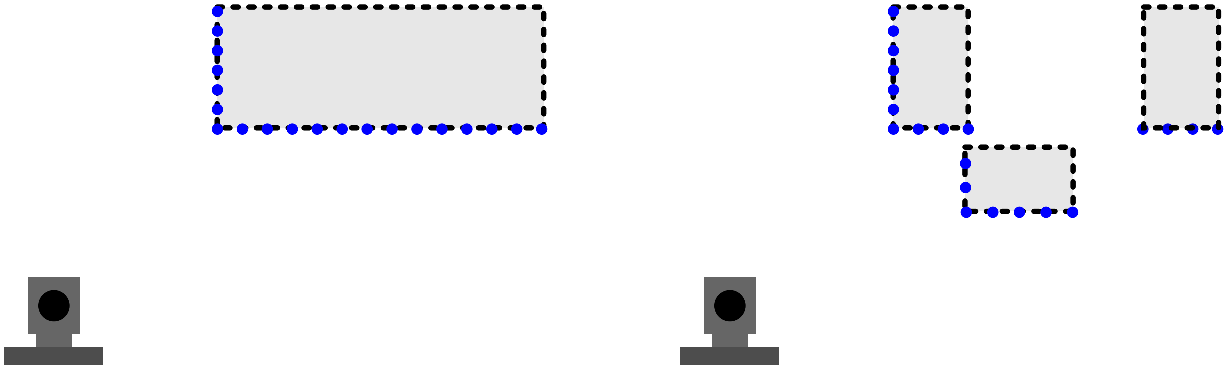
Figure 1.6: The perspective problem in tracking. In this cartoon example, a non-moving object, depicted in gray at the top, is observed at three different locations by the sensor, depicted at the bottom. Sensor readings are shown as blue dots. Changes in perspective can introduce a bias in the estimate of the object's location, shown in red.

and Aycard (2012) improve upon GNN by taking into account semantic class labels for objects and discarding associations made between differing classes. Streller, Furstenberg, and Dietmayer (2002) would predict where a previously observed object would be in the next frame to construct a search area for the current timestep.

Finally, these observations are incorporated into a Bayesian filter. This is commonly done via an extended Kalman filter (EKF), Unscented Kalman filter (UKF), or particle filter where the observations are typically reduced to bounding boxes or centroids. Zhao and Thorpe (1998) use three different motion models in an EKF to represent three different modes of dynamics for a moving vehicle. Darms, Rybski, and Urmson (2008) use a model selection algorithm to choose the best model for tracking from sensor data. Some approaches rely on Multi-Hypothesis Tracking (MHT) for better robustness (Kaestner et al., 2012; Särkkä, Vehtari, and Lampinen, 2007; Wang et al., 2007).

1.4.2 Model-Based Methods

While the methods described above have had much success, they are prone to errors in certain scenarios. One problem arises from changes in viewpoint as our perspective of an object changes, as depicted in Fig. 1.6. For example, if we observe a stationary object as we drive by it, we will first observe the rear of the object, followed by its side, and finally its front. Despite the object not moving, a bounding box or centroid approach will lead to observations of the object at seemingly different locations. Additionally, occlusions can be quite problematic, causing errors in segmentation, as depicted in Fig. 1.7. An occluding object may cast a sensor shadow onto an object behind it such that a clustering algorithm would split the object into two different segments. Perspective or occlusion problems can result in reduced accuracy at best and complete tracking failure at worst.



(a) No Occlusion

(b) With Occlusion

Figure 1.7: The occlusion problem in tracking. In this cartoon example, two similar objects are observed by the sensor at the bottom left. In Fig. 1.7(a), the sensor observes the object without any occlusion. In Fig. 1.7(b), the object is partially occluded by a smaller object. Thus, it now appears as two distinct objects, one at the front and one at the back.

Several methods have been developed to solve these issues of perspective and occlusion. These methods commonly sought to leverage the model of the object itself to make the tracking algorithms more robust.

A simple geometric model of the object, such as a two-dimensional (2D) bounding box, can be used to improve tracking performance. Petrovskaya and Thrun (2008, 2009) constructed a 2D “virtual scan” from a 3D LIDAR scan, leveraging free, occupied, and occluded space. A ray-based measurement model is then used to check for consistency of the sensor data against a bounding box model of the object.

A similar approach is presented by Vu and Aycard (2009). They propose a Markov chain Monte Carlo (MCMC) method to find the optimum tracking solution. For aerospace tracking, Baum and Hanebeck (2014) use a star-convex model to represent a target, allowing for a more detailed shape to be found.

More recently, non-parametric methods have been explored as well. Several methods have proposed using iterative closest point (ICP) for tracking or motion estimation (Feldman, Hybinette, and Balch, 2012; Moosmann and Fraichard, 2010; Moosmann and Stiller, 2013). Snapshots of a moving object can be aligned using a scan registration technique such as ICP. The relative poses given by the registration can then be used in a filtering framework. However, this approach can be slow and prone to the local minima issues that are common with gradient-based methods such as ICP.

Held et al. (2014) improve upon these methods by using annealing histograms. This allows for an efficient search of a 2D translation to find the best alignment of the current observation with the previous scan. Unlike a gradient-based ICP approach, this search is not prone to local minima. The motion result can then be refined for rotation as well. Additionally, by using a camera sensor, color appearance can be incorporated into this framework.

Wang (2004) sought to solve the related problems of SLAM and detection and tracking of moving objects (DATMO) together. Wang, Thorpe, and Suppe (2003) integrate these two problems, detecting and tracking dynamic objects without a priori knowledge of the objects by relying on odometry and a SLAM map. Wolf and Sukhatme (2004) propose a system where multiple occupancy grids are maintained, one representing static structures and one capturing dynamic objects. Their results demonstrated how modeling both types of objects help improve results for mapping both dynamic and static parts of the environment.

The model-based approaches discussed here typically make similar assumptions regarding the sensor data that is used as input to these algorithms. Notably, they are reliant on segmentation and data association in one way or another. While several techniques exist to help account for errors in these steps, such as MHT, these systems can suffer from catastrophic failure when there is an error in either of these steps.

1.4.3 Model-Free Methods

Several approaches in dynamic motion estimation from LIDAR data do not explicitly rely on a model for the object being tracked. Instead, these approaches often rely on grid-based representations, such as occupancy grids or a grid of moving particles, to estimate the dynamic motion in the environment or predict future occupancy states of the world.

Vu, Aycard, and Appenrodt (2007) and Azim and Aycard (2012) both employ scan differencing for detection. In this procedure, two occupancy grids at different times are compared. Any space or voxels in the occupancy grid that are occupied at one time but not the other are labeled as dynamic. These dynamic voxels are then tracked over time.

Danescu, Oniga, and Nedevschi (2011) present a grided particle filter approach. Particles are distributed throughout a grid world, each with an estimate of its own location and velocity. As these particles move throughout the grid, a resampling procedure creates or destroys them according to sensor observations. As a result of this resampling, the surviving particles at any location in the grid reflect an estimate of the occupancy of that location and any motion associated with an object that might be present there.

Tanzmeister et al. (2014) similarly present a grid-based mapping and tracking approach that models both dynamic and static objects. Data association, segmentation, and filtering or classification of dynamic or static objects are not required. This representation can be used to output both an occupancy state of the environment and a velocity map that describes the motion in the world.

Other methods turn to deep learning to estimate dynamic motion. Choi, Lee, and Oh (2016) use a recurrent neural network (RNN) to predict the future occupancy state of an environment by estimating velocities at each voxel. This RNN approach is shown to outperform baseline optical flow techniques. Byravan and Fox (2017) introduce SE3-NETS, a neural network that predicts rigid body motion from raw point clouds and actions on objects. Their work focuses on applications for robotic manipulation tasks, such as with a Baxter robot.

Ondruska and Posner (2016) extend the work of Choi, Lee, and Oh (2016) to explicitly track objects through occlusions. A simulation with many dynamic objects is created. These objects commonly occlude each other as they move about. A stream of synthetic sensor data from this simulation is fed into the network. The network learns to predict the full state of the environment, including the objects that are occluded for a short duration of time.

Dequaire et al. (2017) further extend the work of Ondruska and Posner (2016) in a framework called deep tracking (DT). Using a similar approach, a network is trained using real-world data to observe the world from a stream of sensor data and predict a future occupancy grid. The goal of DT is to model the distribution $P(\mathbf{x}_t|\mathbf{z}_{1:t})$ with a RNN, where $\mathbf{x}_t$ is state of the world at time t and $\mathbf{z}_{1:t}$ are the sensor observations through time t. The distribution is modeled through use of an underlying latent state, $\mathbf{h}_t$. Thus,

$$P(\mathbf{x}_t|\mathbf{z}_{1:t}) = P(\mathbf{x}_t|\mathbf{h}_t) \tag{1.2}$$

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{z}_t), \tag{1.3}$$

where both $P(\mathbf{x}_t|\mathbf{h}_t)$ and $f(\mathbf{h}_{t-1}, \mathbf{z}_t)$ are modeled using the neural network. One key advantage of DT is that it does not require hand-labeled training data. From a stream of occupancy grid data, ground truth results for a future occupancy state can be directly observed. While the full state of the world, $\mathbf{x}_t$, will never be observed due to occlusions, the network can be trained on the observable ground truth.

Instead of relying on a discrete occupancy grid, Senanayake et al. (2016) use Hilbert Maps (Ramos and Ott, 2016) for a continuous occupancy representation. Using what they called “hinged features”, dynamic aspects of the environment can be captured, and a future occupancy state can be predicted.

1.5 Flow Estimation

Dynamic motion estimation from raw sensor data, without modeling distinct objects, has been studied extensively in various communities. In the field of computer vision, estimating raw motion from a stream of images, known as optical flow, is a well studied problem. In the case of a stereocamera system, this is known as scene flow. Several techniques have also been developed to estimate dynamic motion directly from LIDAR data.

1.5.1 Optical Flow

In the field of computer vision, optical flow has been a popular research area in which motion is estimated in an image due to a moving platform or dynamic objects in the environment. While motion estimation in camera data and LIDAR have many similarities, it is important to note the unique advantages and challenges that each sensor modality provides. For example, unlike LIDAR sensors that provide a relatively sparse set of observations, cameras provide a rich view of the scene. In addition to denser measurements, cameras also provide a much more accurate estimate of the pixel appearance (such as RGB color or grayscale intensity). While some LIDAR sensors do report the intensity of the laser return, these intensity values are usually not very discriminative in general and are unreliable, as they could vary significantly due to the angle of incidence or between different laser beams in a sensor.

Optical flow is typically approached by solving for a 2D motion field in the image plane that preserves some constancy metric (such as brightness constancy) and a regularization term to promote spatially smooth flow (Horn and Schunck, 1981; Lucas and Kanade, 1981).

For example, we can express brightness constancy, where we assume that the appearance of a point in the world stays constant, by writing:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) \approx I(x, y, t), \tag{1.4}$$

where $I(x, y, t)$ is the brightness or intensity of the pixel (x, y) in the image I at time t, and $(\Delta x, \Delta y)$ is the motion in the image plane at (x, y) over the time interval Δt. We can approximate I using a first order Taylor series,

$$I(x + \delta x, y + \delta y, t + \delta t) \approx I(x, y, t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}, \tag{1.5}$$

where we omit higher order terms. Thus, we can express our brightness constancy constraint by stating that:

$$\frac{\partial I}{\partial x}\frac{\delta x}{\delta t} + \frac{\partial I}{\partial y}\frac{\delta y}{\delta t} + \frac{\partial I}{\partial t}\frac{\delta t}{\delta t} = 0. \tag{1.6}$$

Note that $\frac{\delta x}{\delta t}$ and $\frac{\delta y}{\delta t}$ represent the 2D optical flow that we are interested in, which we will denote with u and v. $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$, and $\frac{\partial I}{\partial t}$ are the derivatives of $I(x, y, t)$ in x, y, and t, which we will denote with I_x, I_y, and I_t, respectively. Thus, we have

$$I_x u + I_y v = -I_t. \tag{1.7}$$

Note that our system is underconstrained, as (1.7) provides only a single constraint for the two unknowns u and v. This is often referred to as the aperture problem. Several methods exist to add additional constraints to solve for optical flow.

A seminal example is the Horn-Schunck method (Horn and Schunck, 1981). In this approach, it is assumed that the flow field is smooth over the entire image I. This results in an energy minimization problem, where the energy is given by:

$$E = \iint \underbrace{I_x u(x,y) + I_y v(x,y) + I_t}_{\text{Brightness Constancy (1.7)}} + \alpha^2 \underbrace{\left(\left\|\vec{\nabla} u(x,y)\right\|^2 + \left\|\vec{\nabla} v(x,y)\right\|^2\right)}_{\text{Smoothing Term}} dx\, dy, \tag{1.8}$$

where α is a manually chosen parameter that determines the smoothing weight.

Other methods have been proposed as well. Lucas and Kanade (1981) propose a local smoothing approach rather than a global one. Liu et al. (2008) propose using scale-invariant feature transform (SIFT) feature matching to help compute flow between two images. Barnes et al. (2009) propose matching patches between different images to find correspondences for image editing, called PatchMatch. PatchMatch can be adapted to help estimate optical flow (Hu, Song, and Li, 2016).

1.5.2 Scene Flow

Scene flow, similarly to optical flow, seeks to estimate a motion field from camera imagery. The key difference is that scene flow aims to estimate a 3D motion field, with the use of a stereo camera or some depth sensor. Scene flow was originally introduced by Vedula et al. (1999). Many approaches follow a generally similar framework to optical flow. Combining a 2D motion estimate from optical flow with a depth estimation method (such as stereo matching in a binocular camera setting) can lead to a 3D motion estimate. A survey of scene flow methods is provided by Yan and Xiang (2016).

Disparity-based methods estimate $(u, v, \delta d)$, where (u, v) is the motion from an optical flow-like approach and δd is the difference in disparity. Isard and MacCormick (2006) simultaneously estimate motion and disparity. Huguet and Devernay (2007) use a variational framework to jointly estimate the scene flow and the 3D reconstruction of the scene. Wedel et al. (2008) decouple the problems of disparity estimation and motion estimation and can achieve 5 Hz run-time performance.

Another branch of methods deal with point clouds or similar representations. Hadfield and Bowden (2011) rely on a collection of particles moving in space to estimate flow. Additionally, by using a Kinect sensor, they explicitly rely on this depth data rather than using a multi-view approach. Basha, Moses, and Kiryati (2013) propose using a 3D point cloud parametrization in a variational framework to jointly estimate scene flow and 3D structure. Ferstl et al. (2014) frame a variational energy minimization problem that is solved using the primal-dual algorithm.

Patch-based methods have found success as well. Hornacek, Fitzgibbon, and Rother (2014) find patch correspondences to densely estimate flow in an optimization framework. Vogel, Roth, and Schindler (2014) propose an energy minimization framework based on matching piecewise-planar patches.

Recently, scene flow has emerged in autonomous vehicle applications. The KITTI scene flow evaluation suite (Geiger et al., 2013) has provided a valuable benchmark for evaluating algorithms in scene flow evaluation in an autonomous vehicle setting. Many methods have been developed for this application domain (Behl et al., 2017; Jaimez et al., 2015; Menze and Geiger, 2015; Vogel, Schindler, and Roth, 2013, 2015). However, these state-of-the-art scene flow techniques from the computer vision community are generally not capable of real-time performance. At the time of this writing, the top nine submissions to the KITTI scene flow evaluation benchmark take five minutes or longer to process a single scene. In the current leading approach on the KITTI benchmark, Behl et al. (2018) present an approach to learn a end-to-end network to predict scene flow from 3D points generated from images. Their network is trained on a dataset created by augmenting the KITTI dataset. In addition to flow, the learned network is able to perform detection as well, producing bounding boxes. However, the reported runtime of this method is 10 minutes per scene.

Unfortunately, due to the differences in sensing modalities, many of the methods developed for scene flow estimation from a stereocamera system do not directly translate from computer vision to LIDAR sensing.

1.5.3 Flow from LIDAR

Recently, the general ideas of optical flow and scene flow have started to be applied to LIDAR data as well. While similar in spirit, these techniques must take different approaches due to the differences in sensing modality.

There is some overlap in this area with grid-based tracking methods. Both Danescu, Oniga, and Nedevschi (2011) and Tanzmeister et al. (2014) propose frameworks that estimate motion in an occupancy grid that can be interpreted as a scene flow estimate.

Dewan et al. (2016) estimate rigid scene flow between LIDAR scans. They formulate an energy minimization problem based on matching signature of histograms (SHOT) feature descriptors (Tombari, Salti, and Di Stefano, 2010) for a subset of keypoints. However, these descriptors can be prone to errors with sensor noise or ambiguous inputs.

Liu, Qi, and Guibas (2018) develop a network called FlowNet3D. This end-to-end network learns to predict scene flow from LIDAR point clouds. Trained from synthetic data only, it demonstrates impressive results on the real-world KITTI dataset.

1.6 Object and Scene Understanding

While sensors such as LIDAR or camera systems can provide rich data that is useful for dynamic motion estimation and other tasks, these sensors are not without limitations. Notably, they are prone to occlusions. For example, an occluding object may block a LIDAR sensor from scanning an area, leaving a gap in the data where it is unknown whether the occluded space is free or occupied. These types of occlusions often present challenges in tracking and motion estimation. However, these issues can be mitigated if a representation is built that is robust to these issues. Additionally, representations can be learned that take advantage of the semantics of the scene, something that cannot be accomplished by a purely geometric or structural representation such as an occupancy grid.

Building an understanding or representation of an object or a scene is a task that has applications in many areas. For example, in the tasks of reconstruction, inpainting, or scene completion, a noisy or incomplete view of an object or scene, such as an image or a voxel grid, is given. From this representation, the goal is to de-noise the representation and/or infer the missing portions. In feature learning tasks, the goal is to learn a feature space where a representation of an image or a point can be used downstream in other areas, such as classification or segmentation. In recent years, there has been much work done in building and leveraging these representations of objects and scenes from partial views and scans. Work in this area commonly leverages deep learning methods.

1.6.1 Reconstruction, Semantic Inpainting, and Scene Completion

The tasks of reconstruction, semantic inpainting, and scene completion have been extensively studied. In recent years, deep learning approaches have found much success. Two of the most widely used methods are the autoencoder and the Generative Adversarial Network (GAN).

1.6.1.1 Autoencoder Approaches

Autoencoders and their variants have two parts. An encoding network, or encoder, transforms some high dimensional data, such as an image or a 3D volumetric grid, into a low dimensional latent representation. A decoding network, or decoder, then takes this low dimensional latent representation and recreates the input data. This full network is usually trained on a loss function that seeks to minimize the reconstruction error between the input and output (for example, pixel-wise L2 loss or cross entropy) (Hinton and Salakhutdinov, 2006; Vincent et al., 2010).

Girdhar et al. (2016) train an autoencoder for 3D object reconstruction. In addition to voxel-wise cross entropy loss, they also train a network to map from image data to the same latent representation and use a Euclidean loss between the two. They show that their learned latent representation, with some feature augmentation, is somewhat class-discriminative, despite not explicitly being trained to be so.

Guizilini and Ramos (2017) consider 3D reconstruction using autoencoders in the context of building large-scale maps from LIDAR. Their autoencoder learns to reconstruct shape primitives in the world. However, they do not consider distinct objects, but rather segments of the map that they cluster to estimate the occupancy state of unknown space.

Dai, Qi, and Nießner (2017) use an autoencoder to perform shape completion. The latent representation is augmented by the output of a separate independently trained object classification network (Qi et al., 2016). The output of the autoencoder is fine-tuned by finding nearest neighbors in an object database to build the final reconstruction.

Choy et al. (2016) build a network that is an encoder followed by an long short-term memory (LSTM) network followed by a decoder. This network takes images as input and produces 3D volumetric grids. It is trained without relying on semantic object class labels. Fan, Su, and Guibas (2017) use a similar approach to instead generate point clouds. They explore how different loss functions capture shape properties differently. Lin, Kong, and Lucey (2018) also use an image encoder and structure decoder to predict the 3D structure of objects from one or more images to build a dense point cloud. They train their network with both a single object class at a time and multiple classes at once.

Pathak et al. (2016) use an autoencoder to recover missing portions of an image by semantic inpainting. They use a combination of L2 pixel-wise reconstruction loss and an adversarial loss over the whole reconstructed image to ensure it appears realistic after semantic inpainting.

1.6.1.2 GAN Approaches

Originally introduced by Goodfellow et al. (2014), a GAN is often used to recreate the distribution of some training dataset. To train a GAN, an adversarial game is played between two networks, the generator and the discriminator. The generator takes as input a noise vector (which can be thought of as a low dimensional latent representation of the sample) and produces a sample that resembles the input data. The discriminator is given both synthetic samples produced by the generator and real samples from the training dataset and must discriminate between the two. When fully trained, the generator is capable of producing samples that reflect the distribution of the training dataset.

Wu et al. (2016) leverage a GAN for object reconstruction, generation, and classification. They report better results when they train a separate GAN for each semantic object class. This method is improved upon by Smith and Meger (2017), but the results are still better when separately trained.

Wang et al. (2017) consider semantic shape inpainting by using an autoencoder as the generator in a GAN. The network is trained on a combination of voxel-wise reconstruction loss and the GAN objective function.

Yeh et al. (2017) consider semantic inpainting of face images using a GAN. A latent representation is found using the available image data. Then, this latent representation is used with the generator from the GAN to recreate the full image. Qualitative results show an improvement of this method as compared to autoencoder approaches.

In a similar application, Liu, Yu, and Funkhouser (2017) train a 3D GAN to aid users in 3D modeling. A loss function that combines realism (as measured by a discriminator) and semantic dissimilarity (using intermediate activations from a classifier) is used.

1.6.1.3 Other Approaches

For 3D objects, Wu et al. (2015) were one of the first to build 3D deep learning models that can be leveraged for recognition, reconstruction, or classification of 3D objects. They design a convolutional deep belief network (CDBN) that models the relationship and dependencies of a 3D voxel grid and object labels, similar in spirit to the early work of Hinton, Osindero, and Teh (2006).

The task of scene completion becomes more difficult when dealing with a complex environment rather than a single distinct object. However, this can be achieved in applications with repeated structure. Song et al. (2017) performance scene completion from a single depth image of an indoor environment using an end-to-end 3D convolutional neural network (CNN). Dai et al. (2018) use a similar approach for scene completion, but additionally predict semantic labels as well.

1.6.2 Features for Scene Understanding from LIDAR

Recently, many features have been developed for use with point clouds specifically, such as from LIDAR data. Often, these features are manually constructed for use with a specific task. Rusu, Blodow, and Beetz (2009) propose Point Feature Histograms to describe the local geometry about a location in the 3D point cloud. The effectiveness of this feature is demonstrated in 3D registration. Bronstein and Kokkinos (2010) present a scale invariant feature for shape recognition and retrieval. Aubry, Schlickewei, and Cremers (2011) introduce the Wave Kernel Signature for shape analysis.

More recently, feature learning has been applied in this area. These types of approaches have found success in many areas of robotics, such as face recognition (Wen et al., 2016) and long-term image matching (Carlevaris-Bianco and Eustice, 2014). Generally, these methods construct a network to transform the input data to the feature space and use a loss function to promote the separability of the features for the desired task. For example, Wen et al. (2016) proposed center loss, where features from the same class are pulled towards the same center, and centers from different classes are forced to stay apart. Carlevaris-Bianco and Eustice (2014) used a loss function that increases as the Euclidean distance between matching feature pairs grows, and decreases as the Euclidean distance between non-matching feature pairs grows.

Some deep learning methods do not explicitly learn a feature representation, instead leveraging an end-to-end deep learning approach with LIDAR point clouds. For example, Engelcke et al. (2017) develop a CNN for object detection in 3D point clouds.

More recently, PointNet and PointNet++ are two deep learning techniques for point cloud feature representation (Qi et al., 2017a,b). Given an input point cloud, these methods find a feature representation that is invariant to the order of the points and invariant to a global transformation, such as a rotation or translation. The performance of the learned feature representation is demonstrated for tasks including segmentation and classification.

Zeng et al. (2017) present 3DMatch. Patches are extracted from RGB-D reconstructions. Correspondences are collected from different views. A CNN is then trained to learn a geometric descriptor that outperforms existing methods in determining correspondences from these descriptors.

1.7 Autonomous Vehicle Applications

This thesis focuses on dynamic motion perception algorithms for autonomous vehicle applications. In this application domain, any such algorithm must work in conjunction with various other systems on the autonomous vehicle, including controls, planning, and mapping. Ideally, a dynamic motion estimation algorithm would allow for the autonomous vehicle to operate safely (e.g., no collisions with moving objects) and smartly (e.g., be capable of navigating around moving objects to reach the desired location) in any dynamic environment.

However, evaluating such an algorithm in this context can prove difficult. For example, if a failure occurs, it can be hard to know if the failure was due to an error in the dynamic motion estimation or a different system, such as the planner. Accordingly, in this thesis, we focus the evaluation of our proposed methods on the error of the motion estimate that is observed as compared to ground truth. While this evaluation allows us to describe the performance of our dynamic motion estimation algorithms, such an approach is not perfect. For example, an error in the velocity estimate of a car that is far away is not as important as an error in the velocity estimate of a car directly in front of the autonomous vehicle. Nevertheless, this approach allows for evaluating the performance of the dynamic motion estimation specifically, without muddying the results with the performance of other systems.

1.8 Thesis Outline

This thesis aims to extend the state-of-the-art in dynamic motion estimation using LIDAR for the application of autonomous vehicles. We consider two main problems.

- We are given a stream of sensor data from LIDAR sensors. From this data, we explore how to estimate both the trajectory of dynamic objects in the environment and build a model of their shape or appearance. It is critical to model the relationship between the shape or appearance of the object and its trajectory in our observations (see Section 1.4.2). Additionally, the rolling shutter nature of the sensor, especially when multiple sensors are used together, presents its own challenges.

- Since object tracking algorithms are often dependent on segmentation and data association, we explore dynamic motion estimation techniques that do not rely on these methods and thus are more robust. In this problem, we seek to estimate the raw motion in the environment from a stream of LIDAR data.

Toward these two problems, we have made the following contributions:

1. A framework where an object trajectory and a general model of the object shape is estimated. Using continuous-time estimation tools, the proper rolling-shutter nature of LIDAR sensors is modeled, and any number of unsynchronized sensors can be adequately handled.

2. An algorithm for estimating raw motion, or temporal scene flow, between successive LIDAR scans. By designing our methods to leverage a graphics processing unit (GPU), this algorithm is capable of real-time performance.

3. A deep learning framework for feature learning to better understand and represent the environment. An encoding network is trained so that distances in the learned feature space are meaningful. This learned feature space allows for improved performance of the estimate of dynamic motion in the environment.

The work in this thesis has appeared in the following publications:

A. K. Ushani, N. Carlevaris-Bianco, A. G. Cunningham, E. Galceran and R. M. Eustice. *Continuous-Time Estimation for Dynamic Obstacle Tracking.* In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Hamburg, Germany, September 2015.

A. K. Ushani, R. W. Wolcott, J. M. Walls, and R. M. Eustice. *A Learning Approach for Real-Time Temporal Scene Flow Estimation from LIDAR Data.* In Proceedings of the IEEE/RSJ International Conference on Robotics and Automation, Singapore, May 2017.

A. K. Ushani and R. M. Eustice. *Feature Learning for Scene Flow Estimation from LIDAR.* In Proceedings of the Conference on Robot Learning, Zurich, Switzerland, October 2018.

1.9 Thesis Roadmap

These contributions are discussed in the following chapters:

Chapter 2 We present a system for dynamic object tracking for autonomous vehicles. In this work, we seek to simultaneously estimate *both* the trajectory of the object and the object's shape. These two tasks are inherently coupled—given only noisy partial views, one cannot accurately estimate the trajectory of an object if its shape is unknown, nor can one estimate its shape without knowing its trajectory. Additionally, we use a continuous time estimation framework to incorporate sensor data that is collected at a fast rate (e.g., LIDAR). Using these methods, we are able to obtain smooth trajectories and crisp point clouds for tracked dynamic objects. We test our proposed tracker on real-world data collected by our autonomous vehicle platform and demonstrate that it produces improved results when compared to a standard centroid-based EKF tracker.

Chapter 3 We present an approach for estimating temporal scene flow, i.e. a 2D motion field, from successive LIDAR scans of the environment in real-time. Our approach does not require any segmentation or data association result, but produces results that are competitive with dynamic object tracking algorithms that have such a requirement. We introduce *occupancy constancy*, which allows us to measure how locations at two different timesteps are similar in terms of their geometric structure. We present an energy minimization problem to estimate raw flow between scans, and then we incorporate these measurements into a filtering framework to estimate temporal scene flow. We evaluate our method on the KITTI dataset.

Chapter 4 We present a feature learning approach to improve upon the flow estimate. Rather using a hand-designed feature or metric, such as occupancy constancy, we use deep learning techniques that allow us to find a feature encoding that is optimized for our task. We demonstrate the improved performance of the scene flow estimate using our learned feature space, as evaluated on the KITTI dataset.

Chapter 5 We summarize the contributions of this thesis. Additionally, we explore a few avenues for future work.

Appendix A We briefly review relevant details of GPU programming that are essential in the implementation of our algorithms. The principles highlighted in this appendix are leveraged in Chapter 3, Chapter 4, and Appendix B.

Appendix B We discuss the efficient generation of occupancy grids using a GPU. The work presented in this appendix is used in Chapter 3 and Chapter 4.

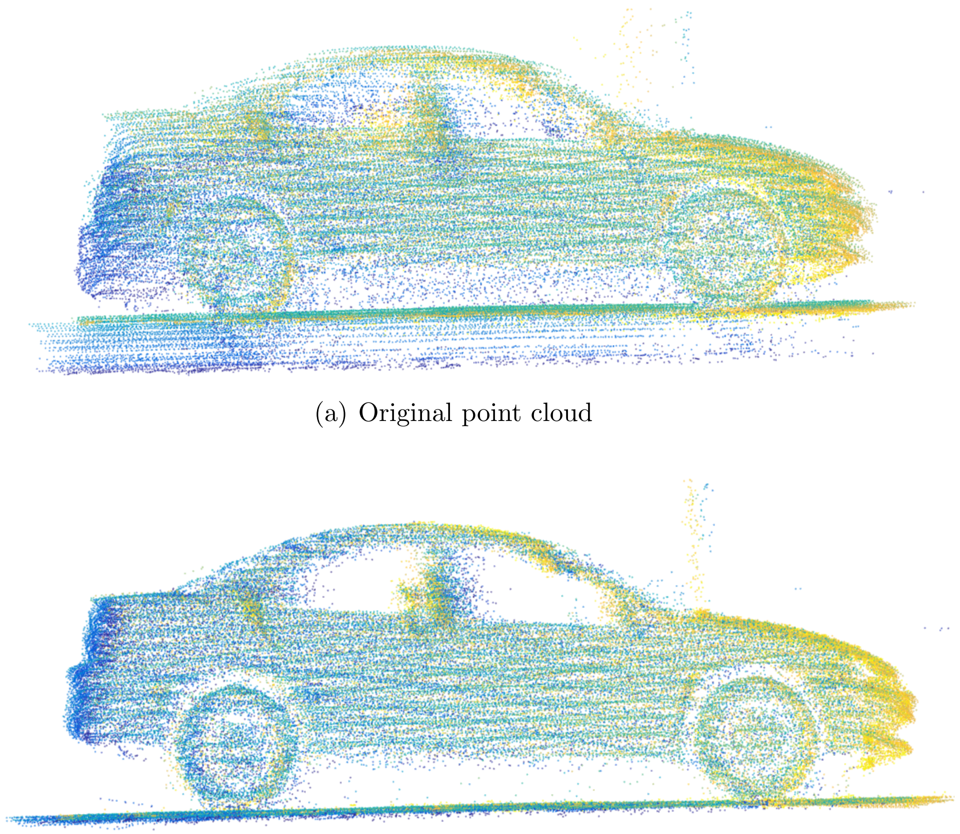
CHAPTER 2

Continuous-Time Estimation for Dynamic Object Tracking

In this chapter, we present a system for dynamic object tracking for autonomous vehicles. We seek to simultaneously estimate *both* the trajectory of the object and the object's shape. These two tasks are inherently coupled—given only noisy partial views, one cannot accurately estimate the trajectory of an object if its shape is unknown, nor can one estimate its shape without knowing its trajectory. To address this challenge, we note that simultaneous localization and mapping (SLAM), where a robot must build a map of the environment while localizing itself within the map, presents similar challenges. By treating the object's shape as a "map" in the object's moving reference frame, we can formulate the object tracking and shape estimation problem similarly to SLAM. Additionally, we use a continuous time estimation framework to incorporate sensor data that is collected at a fast rate (e.g., light detection and ranging (LIDAR)). Using these methods, we can obtain smooth trajectories and crisp point clouds for tracked objects. We test our proposed tracker on real-world data collected by our autonomous vehicle platform and demonstrate that it produces improved results when compared to a standard centroid-based extended Kalman filter (EKF) tracker. This work was published in Ushani et al. (2015).

2.1 Introduction

As autonomous cars continue to develop, one important challenge is to be able to accurately track dynamic objects in the environment, such as other vehicles, bicycles, or pedestrians. Accurate estimates of object positions and velocities are essential to any planning framework that seeks to create safe trajectories. Not only does a planner need to avoid objects in its environment, but often planners must also predict the future actions of vehicles given an accurate trajectory history (Galceran et al., 2015; Xu et al., 2014). Additionally, a history of



(a) Original point cloud

(b) Proposed point cloud

Figure 2.1: A point cloud generated by our proposed dynamic object tracker. Points in the point cloud are colored by the time when they were observed, from blue to yellow. The crispness of the generated point cloud is reflective of the tracking accuracy. Best viewed in color.

an object's pose or its point cloud representation can be used to classify the object (Douillard et al., 2010; Teichman and Thrun, 2012).

Dynamic object trackers can be prone to errors and biases if they do not adequately model the shape of the object being tracked. For example, consider an autonomous vehicle driving past a parked car while tracking it. At first, only the rear of the car is observed. As the parked car is passed, only the side is observed. Afterward, only the front face is observed. During this process, if we do not maintain an estimate for the object model, our tracker may falsely believe that this parked car has moved due to the change in our perspective. Such errors can have adverse effects on a planning system that relies on object tracking to produce a safe plan for the autonomous vehicle. To account for biases such as this, we can use an estimate of some model or representation of the object's structure.

Another challenge is how to incorporate high-rate sensor data. Many current autonomous vehicles rely on one or more LIDAR laser sensors that are collecting data at high rates, for example, collecting over 700,000 points per second. Many current trackers make an implicit assumption that a set of data or a discrete "snapshot" (i.e., set of LIDAR observations in succession collected from a single scan of the object) is collected at a single point in time. While this snapshot assumption helps provide some known structure to the object and makes the problem formulation more simple, it does not account for any movement of the object while the snapshot is being collected, leaving these trackers prone to errors. These errors can be exacerbated when there are multiple sensors or different sensor modalities that are not synchronized. However, due to the fast rate of observations coming from these sensors, it quickly becomes intractable to try to compute the pose at each time associated with an observation. One method of handling sensors with fast data rates is to use continuous time estimation, as proposed by Furgale, Barfoot, and Sibley (2012) and further developed by Anderson and Barfoot (2013) and Anderson, Dellaert, and Barfoot (2014). This allows us to represent the object's path not as a discrete set of poses, but as a linear combination of continuous basis functions, reducing the number of variables by several orders of magnitude.

In this chapter, we address the above issues of errors and biases by noting that the challenges inherent to accurate object tracking as described above are similar to those in a typical SLAM problem. There are a few differences between object tracking and SLAM, however. For example, in object tracking the "map" is in the reference frame of the object. Additionally, we are estimating the object's motion, rather than our own. Nonetheless, we will show that we can solve the object tracking problem using a formulation similar to that of state-of-the-art SLAM formulations. Specifically, our contributions are:

1. Modeling and solving of object tracking using a formulation similar to SLAM.

2. Incorporating continuous-time estimation tools to handle fast rate sensors.

3. Evaluating the proposed method on a real-world dataset.

We show that our approach leads to more accurate tracks and object models compared to a standard centroid-based EKF tracker. We evaluate the performance in terms of tracking error and point cloud crispness on a real-world dataset we collected using an autonomous vehicle platform.

2.2 Related Work

Object tracking has been extensively studied. Many different methods and approaches have been developed, with applications ranging from vehicle tracking to aeronautical target tracking.

Many early approaches used three main steps. First, sensor data, such as from a LIDAR scan, is segmented into distinct dynamic objects. Segmentation algorithms include standard clustering or connected component methods (Kaestner et al., 2012; Leonard et al., 2008; Streller, Furstenberg, and Dietmayer, 2002; Wender and Dietmayer, 2008), while some approaches seek to extract features such as line segments to detect objects (Zhao and Thorpe, 1998).

Next, data association is performed to link the new observations with objects that were previously seen. For example, global nearest neighbor (GNN) is commonly used (Aycard et al., 2006; Vu, Aycard, and Appenrodt, 2007), sometimes augmented with semantic information (Azim and Aycard, 2012) or a tracking prediction (Streller, Furstenberg, and Dietmayer, 2002).

Lastly, a filtering framework is used. Many methods use a Bayesian filter, such as an EKF, Unscented Kalman filter (UKF), or particle filter. The observation model used is typically a bounding box or centroid model (Levinson et al., 2011), while the motion model can be chosen based on a semantic class prediction (Darms, Rybski, and Urmson, 2008) or different modes of dynamics (Zhao and Thorpe, 1998). For improved robustness, many methods use Multi-Hypothesis Tracking (MHT) (Kaestner et al., 2012; Särkkä, Vehtari, and Lampinen, 2007; Wang et al., 2007).

More recently, several object trackers make use of some simple object model to improve the tracking results. There are different ways of estimating this model. Petrovskaya and Thrun (2009) use anchor points (such as the center or corner of the perceived object) to estimate a geometric model, such as a box, in order to aid in tracking. Kaestner et al. (2012) use a generative model to extract a bounding box from the object. Darms, Rybski, and Urmson (2008) model objects as points or as boxes, depending on the situation. Vu and Aycard (2009) fit a box model to the object. In the aerospace field, Baum and Hanebeck (2014) approximate an extended object with a simple geometric shape such as an ellipse and then estimate the parameters of this shape. While these simplistic methods are easy to implement and are often fast, they make assumptions about the object model that can lead to tracking errors, which we seek to avoid.

Techniques from scan registration have also been applied for object tracking. Feldman, Hybinette, and Balch (2012) leverage iterative closest point (ICP). Snapshots of an object are aligned together and the relative poses are used as observations in a Kalman filter. However, this has been shown to be slow and prone to local minima, especially when there are errors in correctly segmenting and associating LIDAR observations with an object (Held, Levinson, and Thrun, 2013).

Other techniques for registering snapshots have been explored. Held et al. (2014) propose a method to search for a 2D translation to register the most recent snapshot with previously observed snapshots of the object. This can then be refined for rotation and finally used in a Kalman filter. However, this implicitly assumes that sensor data is available in snapshots, each consisting of many points observed at the same time, and does not model the true nature of the timing of the sensor, leaving it prone to errors due to the object's movement during the collection of the snapshot.

Some object trackers take a somewhat different approach and use an occupancy grid. Tanzmeister et al. (2014) and Danescu, Oniga, and Nedevschi (2011) use a grid-based method where every cell maintains a particle filter and is classified as being a static or dynamic object based on the statistics of the particles in it. Recently, deep learning methods have been leveraged to predict rigid body motion in the scene (Byravan and Fox, 2017) or a future occupancy state of the environment (Choi, Lee, and Oh, 2016; Dequaire et al., 2017; Ondruska and Posner, 2016).

Our proposed object tracker seeks to leverage a generalized object model, as opposed to a simple geometric model such as a bounding box, that can be used to track any kind of object (e.g., car, motorcycle, bicycle, or pedestrian). Additionally, we are interested in leveraging continuous time estimation, as proposed by Furgale, Barfoot, and Sibley (2012), in order to properly handle the fast rate sensors commonly used in this area, accounting for the object's motion during the collection of a snapshot. This technique has been shown to be a successful approach to SLAM problems with high rate sensors (Anderson and Barfoot, 2013) as is the case in our domain.

2.3 Problem Statement

Let $\mathbf{x}(t)$ be the state of the object at time t, and let $\mathbf{z}_{1:n_z}$ be our n_z observations, with $\mathbf{z}_i \in \mathbb{R}^3$ and each associated with a time t_i. Each observation is in the world frame, and we assume that the vehicle has an accurate localization system such that uncertainty in the world frame position of these observations is negligible over the time scale of tracking.

Let $\mathbf{m}^o_{1:n_m}$ be a point cloud representation of the object, consisting of n_m points expressed in the object frame. We denote a point in the object frame as $\mathbf{p}^o$. Points in the world frame are denoted $\mathbf{p}$. We explore the parameter n_m in Section 2.6.

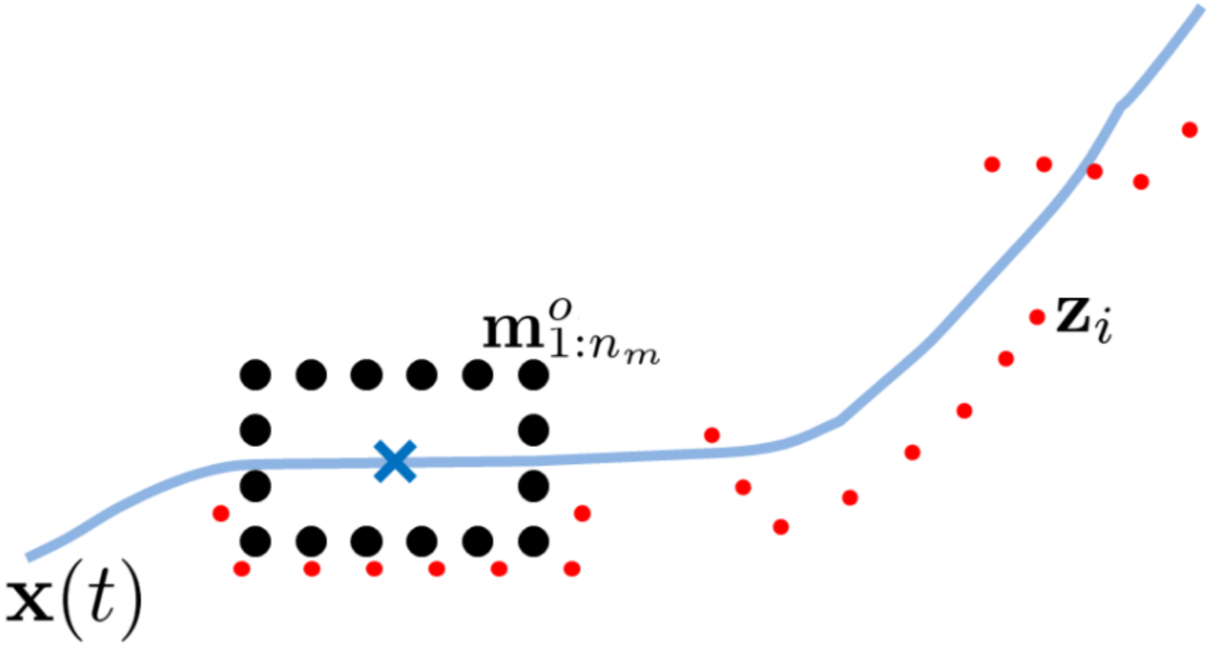


Figure 2.2: A cartoon example of the problem setup. The object moves along some path $\mathbf{x}(t)$, displayed in blue. The object is modeled by n_m model points, each depicted as a black dot. We have observations such as $\mathbf{z}_i$, each depicted as a red dot. Note that each snapshot roughly captures the geometric shape of the object, but is prone to errors due to the object's motion during its collection. In this work, we do not make the assumption that the points $\mathbf{z}_i$ in a single snapshot correspond to the same object pose $\mathbf{x}(t)$.

We see an example in Fig. 2.2. A dynamic object is modeled by $\mathbf{m}^o_{1:n_m}$, depicted by the black points. These points are represented in the frame of the object, depicted by the blue cross, as it moves along some trajectory defined by the continuous time state $x(t)$, depicted by the blue curve. Over time, we observe a set of points from the object, depicted by the red dots.

We seek to find the maximum of the joint posterior density:

$$\mathbf{x}(t)^\star, \mathbf{m}^o_{1:n_m}{}^\star = \underset{\mathbf{x}(t), \mathbf{m}^o_{1:n_m}}{\operatorname{argmax}} \; p(\mathbf{x}(t), \mathbf{m}^o_{1:n_m} \,|\, \mathbf{z}_{1:n_z}), \tag{2.1}$$

solving for the object pose over time $\mathbf{x}(t)$ and model $\mathbf{m}^o$. Note that we have framed the problem as a maximum *a posteriori* (MAP) estimation problem, similar to the formulation that is used in Furgale, Barfoot, and Sibley (2012).

We define the object state $\mathbf{x}(t)$ at time t as:

$$\mathbf{x}(t) = \begin{bmatrix} x \\ y \\ z \\ \phi \\ v \\ v_z \\ \dot{\phi} \end{bmatrix}, \tag{2.2}$$

where the 3D position of the object in the world frame is given by (x, y, z), ϕ and v are, respectively, the current heading of the object and its forward speed, v_z is the speed in the vertical direction, and $\dot{\phi}$ is the rate of turning. We assume that the object's roll and pitch are negligible for performance reasons, as any typical object in our environment would be constrained to a small roll and pitch. However, if desired, roll and pitch could be added to the state vector.

2.4 Front End

The front end to our system is similar to that of Leonard et al. (2007) and Leonard et al. (2008). We refer to this system as our baseline tracker.

We have four LIDAR sensors on our vehicle platform. These sensors continually stream point cloud data. Using an odometry estimate from an inertial navigation system (INS), this point cloud data can be motion-compensated for the movement of the ego-vehicle. Furthermore, our vehicle runs a global localization system similar to that of Levinson and Thrun (2010). Thus, we can transform the point cloud data we receive into a global reference frame.

Once we have about 100 ms of data from the LIDAR sensors corresponding to roughly a full revolution per sensor, we can process the accumulated point cloud. First, we identify LIDAR observations that belong to a dynamic object. This is done by first discretizing the environment into 2D grid cells. Each cell contains the observations from the LIDAR whose (x, y) position fall within the cell. The variance of the z coordinate is measured. If this variance is above a threshold, then all LIDAR observations in that cell are labeled as being an object. Optionally, this module can be supplied with a map of the ground plane (built offline) to significantly improve performance of the object detector.

The LIDAR observations are then segmented into what we call *chunks*. Each chunk is a small four-dimensional (4D) (in time as well as space) container that holds a number of LIDAR observations. LIDAR observations are either added to previously existing chunks or create new ones, depending on their distance to the closest chunk. These chunks are then segmented using connected components.

Once we have segments of chunks, we perform data association for each segment. We maintain an EKF filter that tracks the state of each object we are currently tracking. Using the predicted location and uncertainty of each object, we run GNN for data association. If there is no existing object EKF that is associated for a given segment, we initialize a new EKF. Additionally, if an EKF for an object does not receive any new observations for a given period of time, we delete the EKF and remove the object from the list of objects we are currently tracking. Finally, once we have performed data association, initialized all new objects, and removed all objects no longer observed, we update every EKF with the new observation, the centroid of each segment, and then continue processing more LIDAR data.

We use this EKF estimate for each dynamic object to initialize our proposed tracking method. Similar to other trackers such as Held et al. (2014), we assume that we have accurate segmentation of the LIDAR observations and data association of the segments through time, both for the baseline tracking system and our proposed method.

2.5 Method

Our method simultaneously optimizes for object state over time $\mathbf{x}(t)$ as well as the point cloud model for the object $\mathbf{m}^o_{1:n_m}$ through an iterative batch optimization process. The following sections describe the formulation for the models used, followed by the optimization procedure.

2.5.1 Formulation

Starting from (2.1) and applying Bayes' rule, we have:

$$p\big(\mathbf{x}(t), \mathbf{m}^o_{1:n_m} | \mathbf{z}_{1:n_z}\big) = \eta p\big(\mathbf{x}(t), \mathbf{m}^o_{1:n_m}\big) p\big(\mathbf{z}_{1:n_z} | \mathbf{x}(t), \mathbf{m}^o_{1:n_m}\big), \tag{2.3}$$

where η is a normalization constant. We make the assumption that the object trajectory and the object model are independent:

$$p\big(\mathbf{x}(t), \mathbf{m}^o_{1:n_m}\big) \approx p\big(\mathbf{x}(t)\big) p\big(\mathbf{m}^o_{1:n_m}\big), \tag{2.4}$$

and thus we arrive at:

$$p\big(\mathbf{x}(t), \mathbf{m}^o_{1:n_m} | \mathbf{z}_{1:n_z}\big) \approx \eta p\big(\mathbf{x}(t)\big) p\big(\mathbf{m}^o_{1:n_m}\big) p\big(\mathbf{z}_{1:n_z} | \mathbf{x}(t), \mathbf{m}^o_{1:n_m}\big). \tag{2.5}$$

We now describe the three terms.

2.5.1.1 Process Model

$p\big(\mathbf{x}(t)\big)$ is the process model of the object. Our method is generic for any motion model. In our application, we will use a constant velocity unicycle model $\dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t)\big) + \mathbf{w}(t)$ where:

$$\mathbf{f}\big(\mathbf{x}(t)\big) = \begin{bmatrix} v\cos\phi \\ v\sin\phi \\ v_z \\ \dot{\phi} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{2.6}$$

and $\mathbf{w}(t)$ is zero-mean Gaussian noise with covariance $\mathrm{Q}_u\delta(t - t')$, where $\delta(t)$ is the Dirac delta function. This yields (Jazwinski, 1970):

$$p\big(\mathbf{x}(t)\big) \propto \exp\left\{\int_{t_0}^{t_f} \mathbf{e}_u(\tau)^\top \mathrm{Q}_u^{-1} \mathbf{e}_u(\tau) d\tau\right\}, \tag{2.7}$$

where

$$\mathbf{e}_u(\tau) = \dot{\mathbf{x}}(\tau) - \mathbf{f}(\mathbf{x}(\tau)), \tag{2.8}$$

and t_0 to t_f represents the timespan over which we wish to compute $p\big(\mathbf{x}(t)\big)$.

2.5.1.2 Object Model

$p(\mathbf{m}^o_{1:n_m})$ is our prior on the object model. We use this prior to enforce that objects be of a reasonable size, and do so by weakly constraining each model point to be near the origin:

$$p(\mathbf{m}^o_{1:n_m}) = \prod_{i=1}^{n_m} \mathcal{N}(\mathbf{m}^o_i; \mathbf{0}, \mathrm{R}_m), \tag{2.9}$$

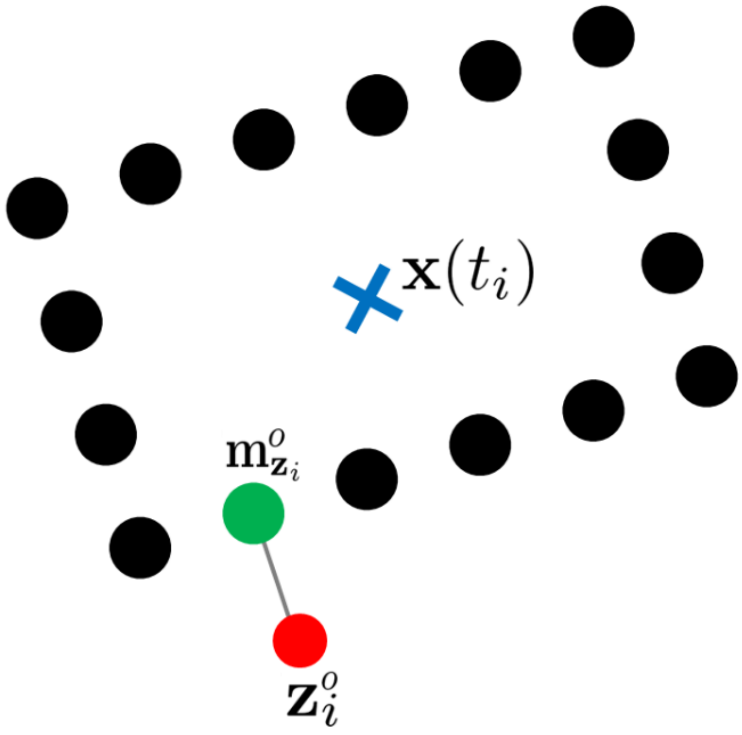


Figure 2.3: An illustration of our measurement model. The observation, the red point, is associated with the closest model point, the green point. The black points represent other points that make up the object model.

where

$$\mathrm{R}_m = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix}, \tag{2.10}$$

modeling the largest expected length, width, and height of object we wish to track.

The object model we present here is fairly simplistic. Other model types, such as a mesh model, could be considered. However, a more complex model would quickly add computational cost to our approach. We find that a point cloud model with a weak prior on size provides a compromise between computational complexity and model expressiveness (as opposed to a bounding box, for example).

2.5.1.3 Measurement Model

$p(\mathbf{z}_{1:n_z}|\mathbf{x}(t), \mathbf{m}^o_{1:n_m})$ is our measurement model. We treat our measurements $\mathbf{z}_{1:n_z}$ as being conditionally independent given the state $\mathbf{x}(t)$ and the model $\mathbf{m}^o_{1:n_m}$. For each $\mathbf{z}_i$, we first associate our observation with a point in the model $\mathbf{m}^o$. We do this by projecting $\mathbf{z}_i$ from the world frame into the object frame at time t_i to find $\mathbf{z}^o_i$. Then, we find the nearest neighbor of $\mathbf{z}^o_i$ in $\mathbf{m}^o_{1:n_m}$ to find $\mathbf{m}^o_{\mathbf{z}_i}$. Thus, our measurement model becomes:

$$\mathbf{z}^o_i = \mathbf{m}^o_{\mathbf{z}_i} + \mathbf{n}_i, \tag{2.11}$$

$$\mathbf{e}_{z_i} = \mathbf{z}^o_i - \mathbf{m}^o_{\mathbf{z}_i}, \tag{2.12}$$

where $\mathbf{n}_i \sim \mathcal{N}(\mathbf{0}, \mathrm{R}_z)$ and $\mathbf{z}^o_i$ is the measurement $\mathbf{z}_i$ projected into the object frame according to $\mathbf{x}(t_i)$, as shown in Fig. 2.3.

2.5.2 Continuous-Time Estimation

In a discrete-time setting, we would instantiate discrete variables representing the object state at $\mathbf{x}(t_0), \ldots, \mathbf{x}(t_{n_z})$. However, we treat each observation from our LIDAR sensor to be a separate measurement (as opposed to a snapshot of observations all assumed to have been taken at the same time). Creating a state variable for each one of these observations would require thousands of variables for every second an object is tracked, quickly becoming intractable.

To address this issue, we use continuous time estimation. This technique was first introduced by Furgale, Barfoot, and Sibley (2012). Rather than having to track a large number of variables, one for each timestep being considered, a set of temporal basis functions are used to parameterized a continuous time function. A maximum likelihood estimate (MLE) problem can be expressed in continuous time to solve for the parameters. In this way, the size of the variables needed is kept manageable, and thus the problem remains tractable.

We model the state of the object being tracked as a linear combination of temporal basis functions:

$$\mathbf{x}(t) = [\boldsymbol{\phi}_\mathbf{1}(t),\ \boldsymbol{\phi}_\mathbf{2}(t),\ \ldots,\ \boldsymbol{\phi}_\mathbf{n}(t)]\,\mathbf{c} \tag{2.13}$$

$$= \Phi(t)\mathbf{c}. \tag{2.14}$$

As our basis functions, we select B-splines of degree 4 (de Boor, 1978). Each B-spline function has limited support, which helps make the problem sparse.

Thus, instead of solving for a large number of state variables, the variables we are solving for are reduced to just the vector of weights $\mathbf{c}$ that operates on the basis functions. The number of variables is reduced by several orders of magnitude, depending on the resolution of B-splines desired.

We thus define the full set of variables that we are solving for as:

$$\boldsymbol{\theta} = \begin{bmatrix} \mathbf{c} \\ \mathbf{m}^o_{1:n_m} \end{bmatrix}. \tag{2.15}$$

2.5.2.1 B-Splines

A B-spline is a piecewise polynomial function. The different pieces of the polynomials meet at points called "knots". For a continuous function of time, these knots are simply points in time $t_0, \ldots, t_m$.

They can be efficiently computed using the Cox-de Boor recursion formula (de Boor, 1978). In the base case of $p = 0$, we have:

$$B_{i,p=0}(x) = \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1} \\ 0 & \text{otherwise} \end{cases}. \tag{2.16}$$

For the recursive case $p > 0$, we have:

$$B_{i,p}(x) = \frac{x - t_i}{t_{i+p} - t_i} B_{i,p-1}(x) + \frac{t_{i+p+1} - x}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(x). \tag{2.17}$$

Note that for $t_k \leq x < t_{k+1}$, $B_{i,p}(x)$ is only non-zero for $k - p \leq i \leq k$. This sparsity in the basis functions can be exploited for improved runtime performance.

Finally, for a given set of weights $\mathbf{c}$, the continuous function is given by:

$$f(x) = \sum_{i=k-p}^{k} c_i B_{i,p}(x). \tag{2.18}$$

With de Boor's algorithm, we can directly compute the function $f(x)$ without having to compute the intermediate B-spline basis functions, saving computation time (de Boor, 1978).

This is given by the following recurrence relation:

$$d_{i,0}(x) = \begin{cases} c_i & \text{if } k-p \leq i \leq k \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

$$d_{i,r}(x) = \begin{cases} \big(1-\alpha_{i,r}(x)\big)d_{i-1,r-1} + \alpha_{i,r}(x)d_{i,r-1} & k-p+r \leq i \leq k \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

$$\alpha_{i,r}(x) = \frac{x-t_i}{t_{i+1+p-r}-t_i}. \tag{2.21}$$

An example demonstrating how B-spline basis functions can represent continuous functions is shown in Fig. 2.4. The continuous function shown in black is a weighted sum of eight B-spline basis functions. In this way, the entire function is parameterized by eight values, $c_0, \ldots, c_7$.

2.5.3 Gauss-Newton

We formulate a cost function for optimization by taking the negative log-probability of (2.1), yielding:

$$-\log\big(p(\mathbf{x}(t), \mathbf{m}^o_{1:n_m}|\mathbf{z}_{1:n_z})\big) = k + J_m + J_u + J_z, \tag{2.22}$$

where

$$J_m = \sum_{i=1}^{n_m} \frac{1}{2}\mathbf{m}_i^{o\top}\mathrm{R}_m\mathbf{m}_i^o, \tag{2.23}$$

$$J_u = \int_{t_0}^{t_f} \mathbf{e}_u(\tau)^\top \mathrm{Q}_u^{-1}\mathbf{e}_u(\tau)d\tau, \tag{2.24}$$

$$J_z = \sum_{i=1}^{n_z} \mathbf{e}_{z_i}^\top \mathrm{R}_z \mathbf{e}_{z_i}^\top, \tag{2.25}$$

and k is a constant.

To solve for $\boldsymbol{\theta}$, we start with an initial guess $\bar{\boldsymbol{\theta}}$, we linearize each of J_m, J_u, and J_z about $\bar{\boldsymbol{\theta}}$. For each, we find $\frac{\delta J}{\delta \boldsymbol{\theta}}^\top$, which is of the form $\mathrm{A}\boldsymbol{\delta\theta} + \mathbf{b}$. We take $\mathrm{A}_m + \mathrm{A}_u + \mathrm{A}_z = \mathrm{A}$ and $\mathbf{b}_m + \mathbf{b}_u + \mathbf{b}_z = \mathbf{b}$. Thus, we have the Gauss-Newton update step $\mathrm{A}\boldsymbol{\delta\theta} = -\mathbf{b}$, by which we iteratively update $\boldsymbol{\theta}$ until convergence.

When solving, we compute a full batch update using all of the data for the object, recomputing the data associations for the measurement model for each iteration. Note that

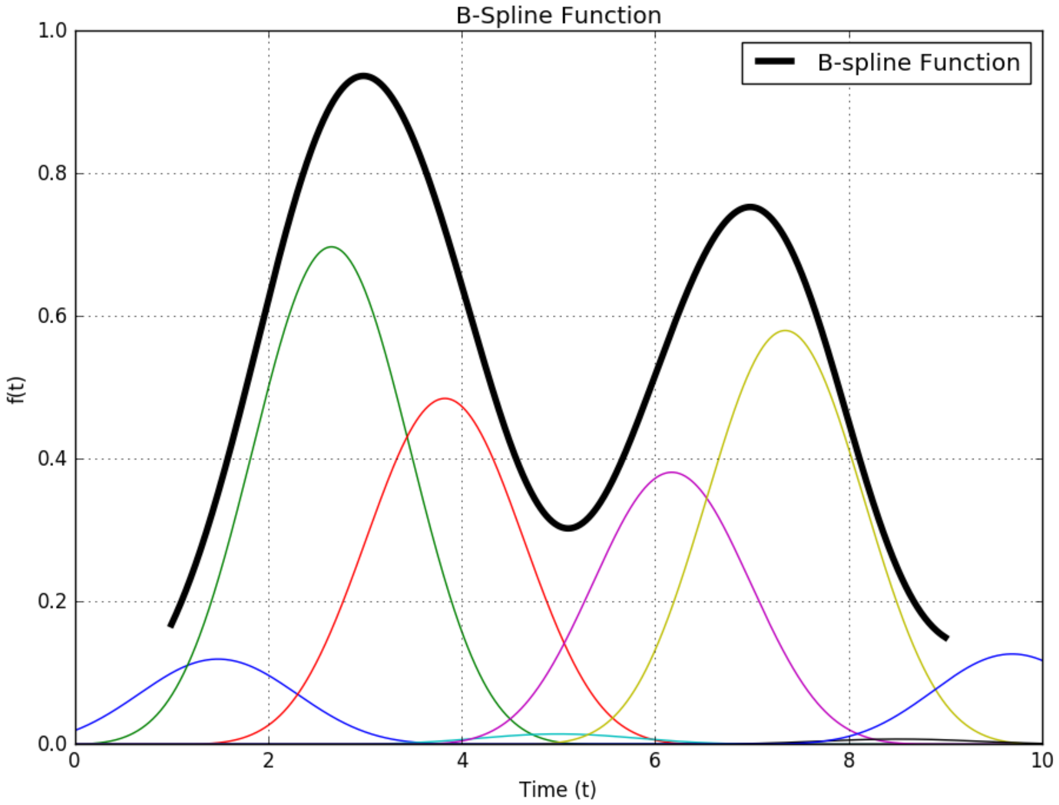


Figure 2.4: A sample function represented by B-spline basis functions. The function, shown with the thick black line, can be represented as a function of B-spline basis functions (shown in thin colored lines). In this particular example, eight B-spline basis functions are used to create a continuous time representation of the function we are interested in.

the matrix A is sparse due to the limited support of the B-splines basis functions, which allows the use of sparse linear solvers that scale to larger numbers of points.

To initialize $\boldsymbol{\theta}$, we use the baseline EKF tracker described in Section 2.4. Specifically, we first fit a continuous function parameterized by B-splines to the pose estimate history given by the EKF tracker to generate our initial guess for $\mathbf{c}$. Then, we project the measurements $\mathbf{z}_{1:n_z}$ into the object frame and subsample these points linearly in time to generate our initial guess for $\mathbf{m}^o_{1:n_m}$.

2.6 Experimental Results

In this section, we present our evaluation and experimental results. We first describe the setup, evaluation, data, and parameter selection. Then, we present results on the accuracy of the pose estimate and the crispness of the generated point cloud. Lastly, we discuss convergence and runtime.

2.6.1 Setup

Our proposed method was evaluated using the NGV Ford Fusion autonomous platform, as discussed in Section 1.2. A global localization system similar to that of Levinson and Thrun (2010) allows us to leverage prior maps of the ground plane to improve the performance of the baseline tracker described in Section 2.4. Experiments were run on a 2.80 GHz Intel Core i7-3840QM CPU.

We replicate the experimental methodology used by Held et al. (2014) by looking at tracking error and point cloud crispness. We evaluated our method on a dataset collected around the University of Michigan, Ann Arbor North Campus.

We evaluate our method in two ways. First, to evaluate the quality of the positional tracking, we identify parked cars in our dataset and evaluate the performance of our tracker on these objects. The tracking system does not know that these objects are not moving, and thus it tracks them over time. Because we know these objects are stationary, we essentially have a ground truth we can use to evaluate our tracker. We call this the stationary set, containing 52 objects each observed for an average of 6.05 s. Then, we look at the crispness of the point cloud created by dynamic object tracks. If we are properly tracking the object, then we would expect to see a clear, crisp point cloud view of it. We call this the dynamic set, containing 52 objects each observed for an average of 6.28 s. In both cases, we have manually discarded objects that are a result of errors in segmentation.

2.6.2 Parameter Selection

We find that a relatively sparse representation of the object's point cloud model $\mathbf{m}^o_{1:n_m}$ can still produce good results while greatly improving the runtime performance, even when the model consists of just a few hundred points. For these results, we set n_m to 300 points. In fact, often if we set n_m too large, the model can tend to overfit any inaccuracies in the initialization from the baseline system, leading to poorer results, as reflected in Table 2.1. Note that after optimizing for $\mathbf{x}(t)$, we can then reproject all of our observations $\mathbf{z}_{1:n_z}$ according to $\mathbf{x}(t)$ into the object's reference frame. Thus, we can recreate the full, dense, point cloud.

The number of measurements n_z that we have grows quickly. Even if we only observe an object for a few seconds, we quickly accumulate hundreds of thousands of observations. We downsample our observations to 5000 observations per object track when computing $p(\mathbf{z}_{1:n_z}|\mathbf{x}(t), \mathbf{m}^o_{1:n_m})$.

For the process model, we set Q_u according to the maximum accelerations and turning rates we expect a vehicle to undertake and by observing vehicle motion in training data. Accordingly, we set:

$$\mathrm{Q}_u = \begin{bmatrix} 0.01\text{ m} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01\text{ m} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01\text{ m} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.001\text{ rad} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9\text{ m/s} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1\text{ m/s} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.08165\text{ rad/s} \end{bmatrix}^2 .$$

For the object model, we considered the largest size of the objects we wish to track, and thus set $\sigma_x = \sigma_y = \sigma_z = 10$ m.

For the measurement model, we set $\mathrm{R}_z = \sigma_z \mathrm{I}$. When choosing σ_z, we must consider both error due to the noise of the LIDAR sensors and error due to the relative sparsity of our model points. We chose $\sigma_z = 0.50$ m. Setting the measurement uncertainty R_z in this way is similar to the approach taken by Held et al. (2014).

For the B-splines, we have a choice of how many B-spline basis functions to use in representing our state. Using more splines would allow us to more finely represent our state over time; on the other hand, the more splines we use, the larger $\boldsymbol{\theta}$ will be, taking longer to solve. Thus, there is a tradeoff to consider between accurate representation of state versus computation time. We use a basis function per dimension for every 0.5 s that the object is tracked.

Table 2.1: The tracking performance of our proposed tracker. In this table, we show results for two values of n_m as compared to the baseline EKF centroid tracker over the stationary set.

Tracker Error	**Baseline EKF**	**Proposed** $n_m = 300$	**Proposed** $n_m = 3000$
Pos. RMSE	0.388 m	0.162 m	0.183 m
Vel. RMSE	0.543 m/s	0.314 m/s	0.328 m/s
ϕ *RMSE*	0.527 rad	0.071 rad	0.090 rad
$\dot{\phi}$ *RMSE*	0.139 rad/s	0.026 rad/s	0.027 rad/s

2.6.3 Pose Results

We report on the tracking error of the stationary set. We compute the position error and heading error by comparing the (x, y, z) and ϕ estimate over time to the mean position and heading, respectively. We compute the velocity and rate of turning error by comparing the velocity and heading estimate to an expected value of 0 m/s and 0 rad/s, respectively. These results are shown in Table 2.1. We see a clear improvement in tracking performance over the baseline EKF centroid tracker as described earlier, particularly in heading. Additionally, note that the performance of the proposed tracker is better with $n_m = 300$ than $n_m = 3000$. We attribute this degraded performance when n_m is larger to overfitting inaccuracies in the initialization.

2.6.4 Point Cloud Crispness

In the general case, with objects that are not known to be stationary, we evaluate the performance of our proposed tracker with regards to the quality of the generated point could. We want a crisp point cloud, or equivalently one with low entropy. A crisper point cloud is indicative of better tracking performance. We evaluate our proposed tracker by using the point cloud entropy as defined by Sheehan, Harrison, and Newman (2012):

$$H[\mathbf{z}^o_{1:n_z}] = -\log\left(\frac{1}{n_z^2}\sum_{i=1}^{n_z}\sum_{j=1}^{n_z}\mathcal{N}(z^o_i - z^o_j; \mathbf{0}, 2\sigma^2\mathbf{I})\right).$$

The parameter σ allows us to tune the resolution at which we evaluate crispness. We set $\sigma = 5$ cm.

The results are shown in Table 2.2 for both the stationary set and the dynamic set. Note the improvement in both cases of our proposed method over the baseline. To compute the ground truth entropy of the stationary set, we project all of the LIDAR observations into the

Table 2.2: Point cloud entropy of the models generated by our proposed tracker. In this table, we show results for our method and the baseline EKF centroid tracker. *Ground Truth* is available for the stationary set by taking the LIDAR observations in the world frame.

Tracker Entropy	**Ground Truth**	**Baseline EKF**	**Proposed** $n_m = 300$
Stationary Set	1.920	2.237	2.103
Dynamic Set	—	2.860	2.756

world frame using the known trajectory of our platform. As the objects in the stationary set are not moving in the world frame, this represents the best point cloud we can construct and thus is the lowest entropy we should expect from an ideal tracker. In the stationary set, we can see that our proposed method reduces the entropy of the point cloud by about 42% relative to the ground truth entropy as compared to the baseline tracker.

The stationary set has less entropy (i.e., it is more crisp) in general when compared to the dynamic set. This is due to the fact that in the stationary set, we commonly traveled close to cars parked on the side of the road, creating a denser point cloud. This is opposed to the dynamic set, where objects are often being tracked from a distance (for example, following a car on the road) or in different lanes. Regardless, the improvement in point cloud crispness is evident in both the stationary set and the dynamic set.

2.6.5 Convergence

We determine how well our method converges by considering the residual of the Gauss-Newton step at each iteration. As can be seen in Fig. 2.5, the error converges consistently on the tracking scenarios that we have evaluated. The decreasing residuals are indicative of better agreement in the process model, measurement model, and object model. In our experiments, the Gauss-Newton process is stopped when convergence is reached or after a maximum of 10 iterations.

2.6.6 Runtime

We find that our proposed method takes an average of 42.9 ms per iteration for our choice of parameters. This was measured on an 2.80 GHz Intel Core i7-3840QM CPU. Note that the matrix A in the Gauss-Newton step is sparse, as only about 2% of the elements are non-zero.

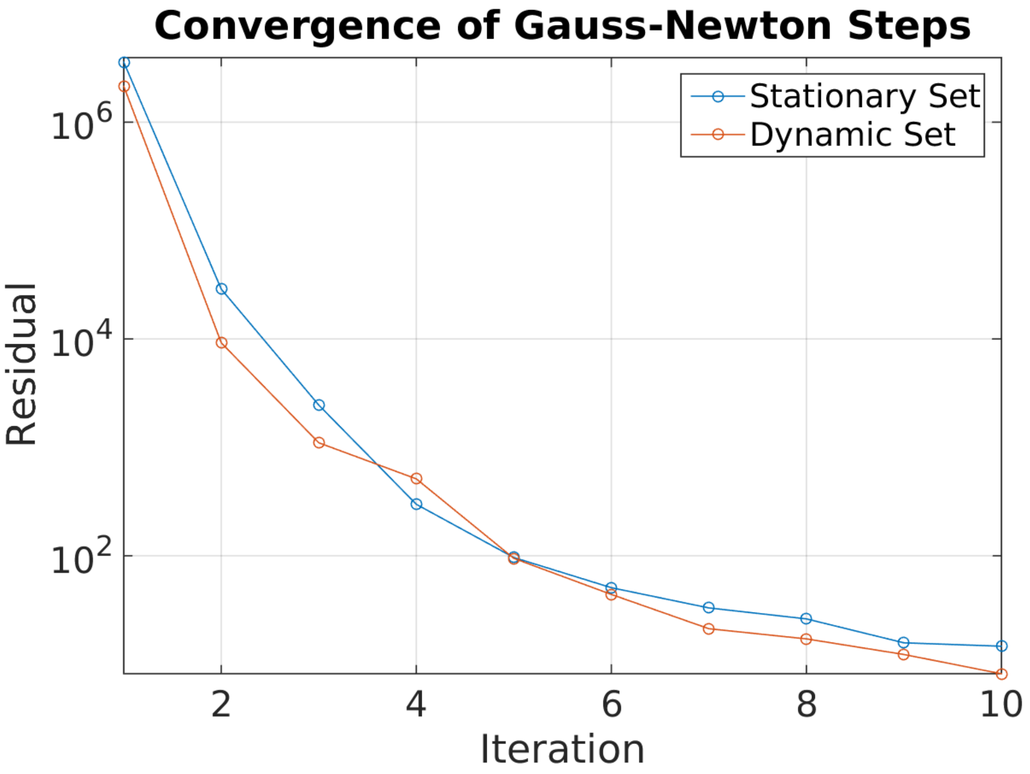


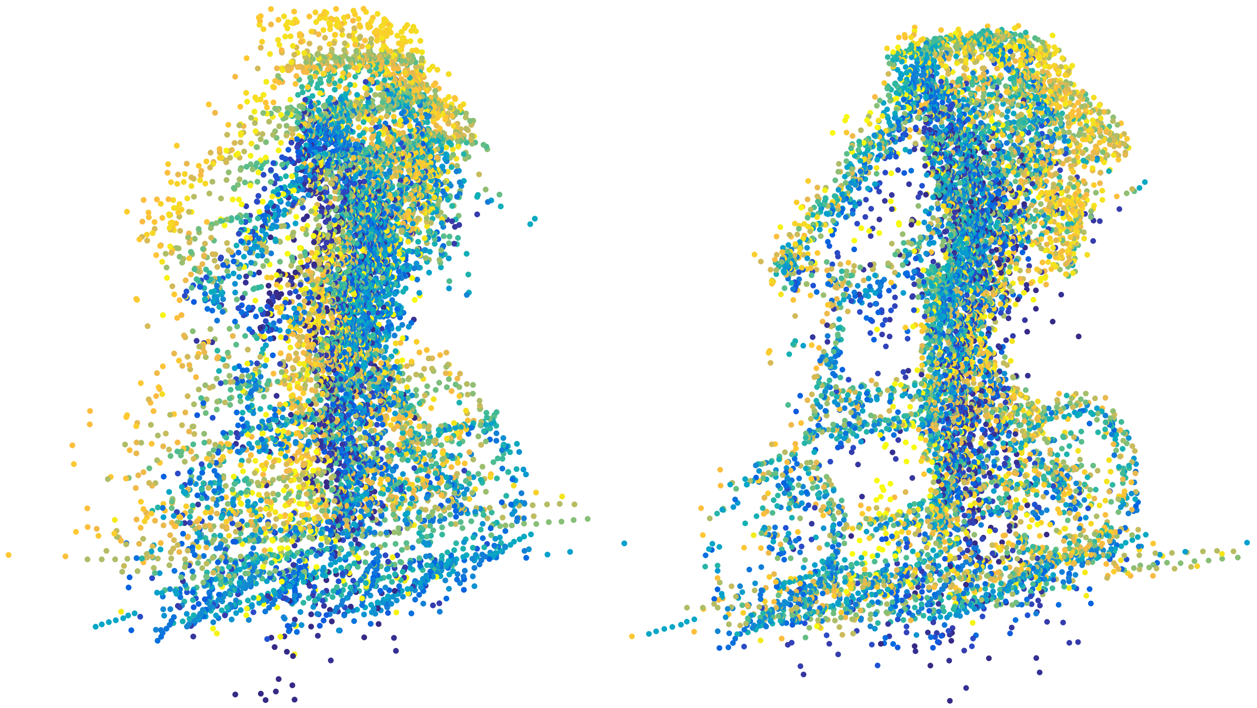
Figure 2.5: The average residual after each iteration over the stationary and dynamic set. Note the log scale. The decreasing residuals are indicative of better agreement in the process, measurement, and object models. Note how our proposed method converges quickly.

2.7 Discussion

We find that our system generally works well under nominal conditions. The tracking error for position, velocity, heading, and rate of heading are all improved with respect to the baseline EKF tracker. Additionally, the resulting point cloud is crisper.

Unlike other dynamic object trackers that rely on certain models (such as a bounding box), we are able to track any object. For example, see Fig. 2.6 for our tracker working on a bicyclist.

We find that heading and lateral position (i.e., perpendicular to the direction of travel) have particularly low error, even when initialized poorly. We show an example of our proposer method tracking a bus. As can be seen in Fig. 2.7, our proposed tracker creates a smoother velocity and heading estimate, which is much more realistic for a bus that cannot change its velocity or heading as abruptly as the baseline tracker would suggest. Furthermore, in Fig. 2.8, we see that our proposed tracker has managed to develop a crisp model of a bus despite a significant amount of heading error in the baseline method.

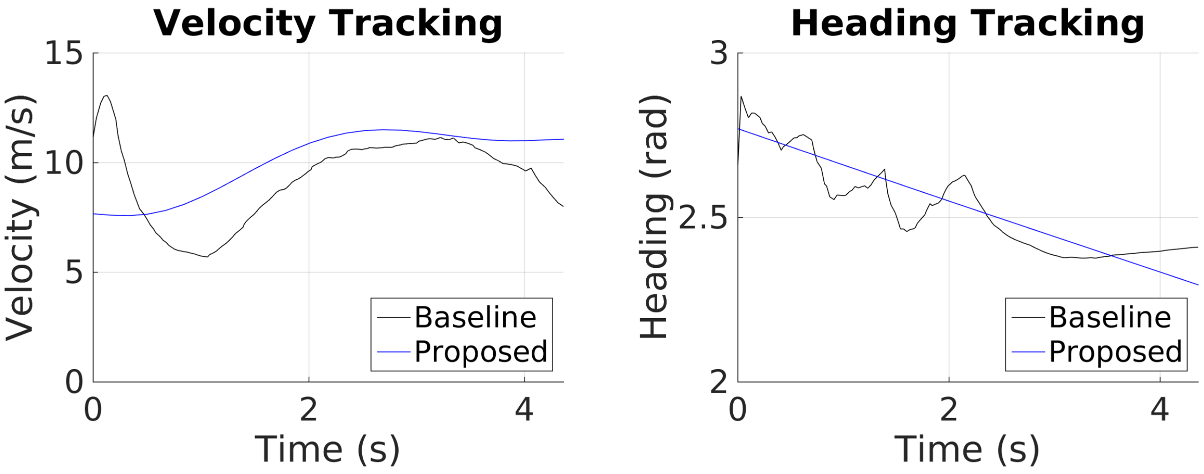

(a) Baseline

(b) Proposed

Figure 2.6: Our proposed method tracking a person riding a bicycle. Points are colored by the time when they were observed, from blue to yellow. Best viewed in color.

Certain situations are known to be troublesome. For example, sometimes we encounter the situation shown in Fig. 2.9 where we might have multiple slightly translated views of the car, which is indicative of tracking error. While this is still a significant improvement over the EKF centroid tracker, our proposed tracker has some trouble resolving the discrepancy along the direction of travel. This is likely due to bad initialization, leading the tracker to create an object model that has two translated instances of the rear face of the car. Even though this is incorrect, this object model is consistent with the measurements and the tracker will believe that it has correctly tracked and modeled the object. We believe that these issues can be addressed with a better prior on the object model, better initialization, or explicit ray casting. Ray tracing to build a representation such as an occupancy grid would allow for smarter reasoning regarding occupied, free, and unknown space. This type of reasoning is missing in a point cloud based approach. Indeed, we investigate this representation in the following chapters.

It is also important to note that our tracking system assumes the availability of accurate segmentation and data association results. This is similar to assumptions made by many works in the field. However, when these assumptions are violated, the resulting error in the tracking pipeline is catastrophic, both in our method and others. While segmentation



(a) Bus Trajectory Velocity

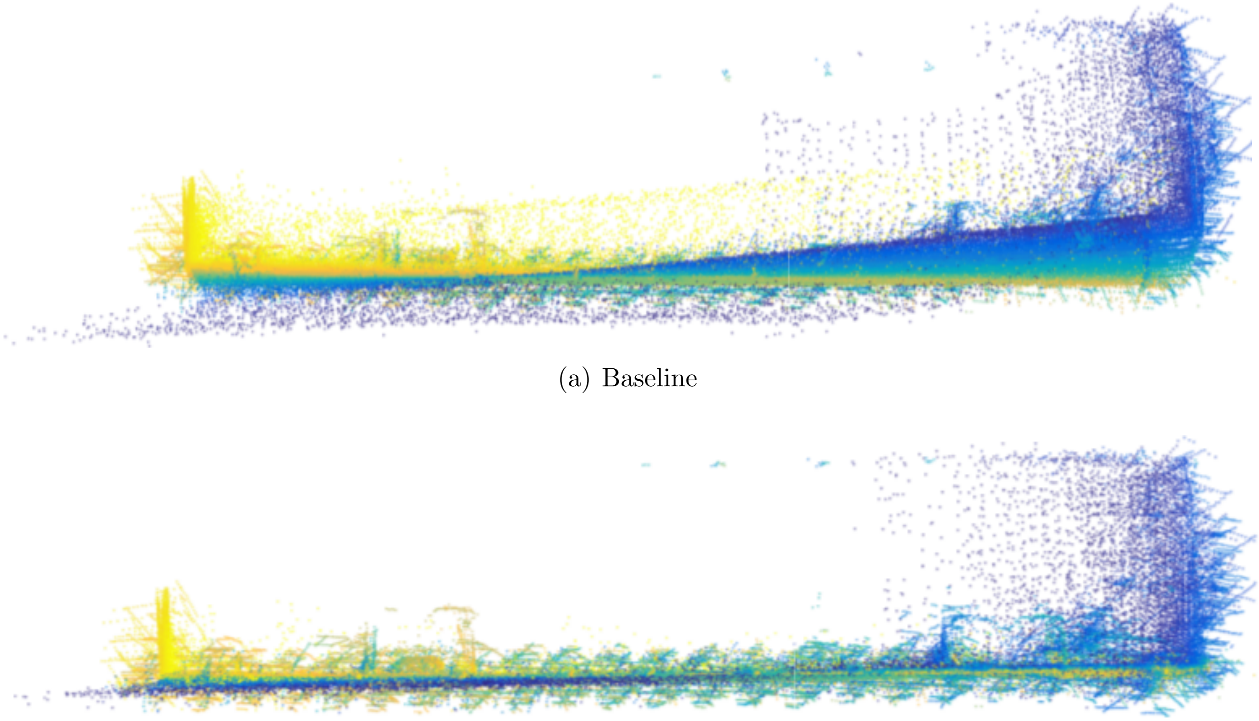
(b) Bus Trajectory Heading

Figure 2.7: Sample velocity and heading profiles. We present a comparison of the estimated velocity and heading profiles for the bus displayed in Fig. 2.8.

and data association can be improved, a robust dynamic tracking algorithm must be able to handle errors in these areas. Indeed, in the following chapters, we investigate dynamic motion estimation techniques that do not rely on these assumptions.

2.8 Conclusion

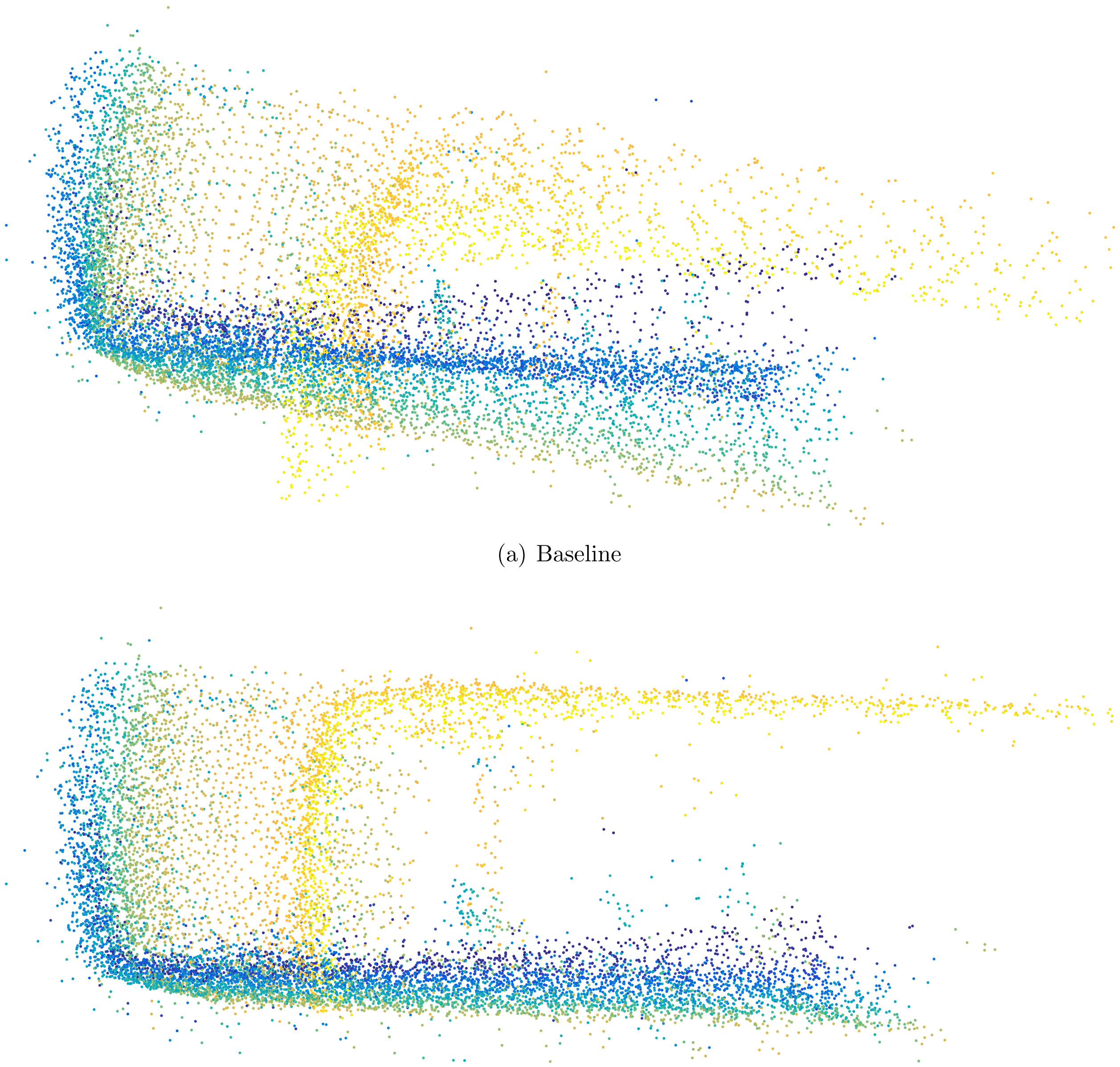
We have demonstrated how tools from SLAM and continuous-time estimation can be applied to dynamic object tracking. We showed how this approach yields improved results when compared to a baseline EKF centroid-based tracking system. Future work will consider better priors and models., such as a mesh object model. Tradeoffs between improved error functions and runtime warrant further investigation as well. Additionally, we will consider better modeling the relationship between the object trajectory and the corresponding point cloud model.



(a) Baseline

(b) Proposed

Figure 2.8: Our proposed method tracking a bus, top-down view. Points are colored by the time when they were observed, from blue to yellow. Best viewed in color.



(a) Baseline

(b) Proposed

Figure 2.9: A fault case of our proposed tracker. While the tracking performance is improved, there is still error in the direction of travel. Points are colored by the time when they were observed, from blue to yellow. Best viewed in color.

CHAPTER 3

Real-Time Temporal Scene Flow Estimation

Many autonomous systems, including autonomous vehicles, require the ability to perceive and understand motion in a dynamic environment. In this chapter, we present a novel algorithm that estimates this motion from raw LIDAR data in real-time without the need for segmentation or model-based tracking. The sensor data is first used to construct an occupancy grid. The foreground is then extracted via a learned background filter. Using the filtered occupancy grid, raw scene flow between successive scans is computed. Finally, we incorporate these measurements in a filtering framework to estimate temporal scene flow. We evaluate our method on the KITTI dataset. This work was published in Ushani et al. (2017). Source code for this work is available at https://github.com/aushani/tsf.

3.1 Introduction

Autonomous systems, such as self driving vehicles or other mobile robots, operate in dynamic environments where it is critical that they be able to accurately perceive and understand the motion of the surrounding environment. Increasingly often, these systems are equipped with one or more light detection and ranging (LIDAR) sensors. These sensors provide a point cloud representation of the world, often collecting millions of points per second.

Many algorithms that consider dynamic scene understanding work with the point cloud directly. For example, obstacle tracking techniques for self-driving vehicles often rely on detection and segmentation of objects directly from the LIDAR generated point cloud, including our previous work (Ushani et al., 2015) and others (Baum and Hanebeck, 2014; Darms, Rybski, and Urmson, 2008; Held et al., 2016; Kaestner et al., 2012; Petrovskaya and Thrun, 2009; Vu and Aycard, 2009).

However, working with point clouds alone disregards a valuable characteristic of the sensor: the notion of free space swept out between the point return and the LIDAR sensor. Occupancy grids are a commonly used representation that can be readily manipulated to

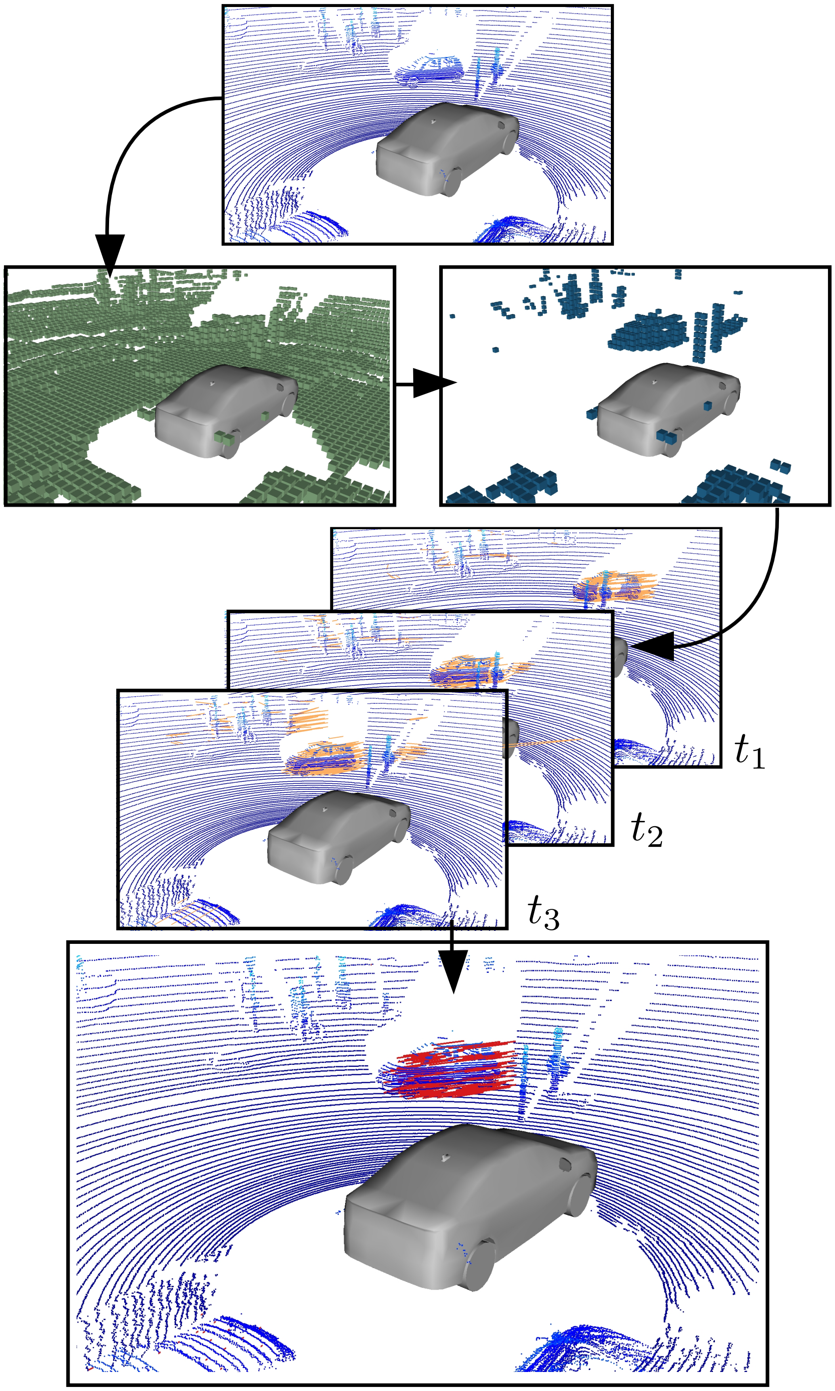


Figure 3.1: An overview of our temporal scene flow pipeline. As an input to our system, we take raw LIDAR scans. We construct occupancy grids from these scans, which are then filtered to remove the background. We then compute raw scene flow measurements using our occupancy constancy metric. Finally, we incorporate this measurement in a filtering framework to refine the estimate and reject false measurements, producing a estimate of temporal scene flow.

capture free and unknown space in addition to the occupancy of a LIDAR point return. This can be achieved by ray-casting from the sensor emitter to the returned point, populating cells of a grid with observations of “free space” (Thrun, Burgard, and Fox, 2005).

Applications of occupancy grids have been widely considered in the 2D domain. However, extending the use of occupancy grids to three-dimensional (3D) sensors has been limited. The exponential increase in processing required for handling the significant number of voxels in a 3D occupancy grid presents a key challenge. In this work, our algorithms are designed so that they can easily be offloaded to a graphics processing unit (GPU). Thus, we can better formulate a real-time temporal scene flow framework from LIDAR scanners.

In this chapter, we present a pipeline that takes raw LIDAR data as input and produces a scene flow estimate. Previously in Chapter 2, we considered the similar problem of obstacle tracking (Ushani et al., 2015). Motivated by some of the shortcomings in this previous work, a key goal of our proposed method here is to avoid relying on temporal data association, segmentation, or use of an object model. Instead, we formulate the problem similarly to that of optical flow over 3D occupancy grids. The novel contributions of this chapter include:

1. A learned framework for tracking LIDAR observations in occupancy grids, leveraging background subtraction and temporal occupancy constancy.

2. An expectation-maximization (EM) algorithm for estimating raw, incremental scene flow.

3. Real-time implementation on a GPU enabling 10 Hz scene flow estimation.

4. Extensive evaluation of our method against known ground truth from the KITTI dataset (Geiger et al., 2013).

3.2 Related Work

Estimating dynamic motion from sensor data has been studied extensively in various communities, including computer vision, self driving vehicles, and mobile robotics.

In the field of computer vision, optical flow or scene flow has been a popular research area in which motion in an image due to a moving platform or dynamic objects in the environment is estimated. While motion estimation in camera data and LIDAR have many similarities, it is important to note the unique advantages and challenges that each sensor modality provides. For example, unlike LIDAR sensors which provide a relatively sparse set of observations, cameras provide a rich view of the scene. In addition to denser measurements, cameras also provide a much more accurate estimate of the pixel appearance (such as color or pixel intensity). While some LIDAR sensors do report the intensity of the laser return, these intensity values are usually not very discriminative in general and unreliable, as they could vary significantly due to incidence angle or between laser sensors.

Optical flow is typically approached by solving for a two-dimensional (2D) motion field in the image plane that preserves some constancy metric (such as brightness constancy) and a regularization term to promote spatially smooth flow (Horn and Schunck, 1981; Lucas and Kanade, 1981). Feature based methods (Liu et al., 2008) or matching patches of images (Barnes et al., 2009; Hu, Song, and Li, 2016) has been proposed as well.

In scene flow, the 3D motion is estimated with the use of a stereo camera or some depth sensor (Jaimez et al., 2015; Menze and Geiger, 2015; Vogel, Schindler, and Roth, 2013, 2015). However, scene flow is generally not capable of real-time performance: at the time of this writing, the top nine submissions to the KITTI scene flow evaluation benchmark take five minutes or longer to process a single scene. In the current leading approach on Recently, Jaimez et al. (2015) presented a method to estimate scene flow in real-time for displacements up to 15 cm. However, due to the difference in sensing modalities described above, these methods do not directly translate from computer vision to LIDAR sensing.

While there are some key differences, obstacle tracking considers a similar problem. Whereas we are interested in sensing and detecting motion in general in a dynamic environment, in obstacle tracking, discrete objects are extracted from sensor data and detected over time. These observations of objects are associated temporally and used to compute trajectories or build appearance models. Many obstacle tracking methods rely on simple geometric models of obstacles, such as boxes or ellipses, which are fit to measurements over time (Baum and Hanebeck, 2014; Darms, Rybski, and Urmson, 2008; Kaestner et al., 2012; Petrovskaya and Thrun, 2009; Vu and Aycard, 2009). More recently, some methods do not make assumptions about the obstacle's appearance or structure. In our previous work, we framed obstacle tracking as a problem similar to that of SLAM, in which an obstacle's trajectory and "map" (i.e., point cloud model) are computed (Ushani et al., 2015). Held, Levinson, and Thrun (2013) provide a framework by which to efficiently register successive scans of an obstacle, providing an estimate of relative motion between these two scans and incrementally building up an obstacle model. Unlike our problem area, however, many approaches in obstacle tracking assume that the sensor data has been segmented into discrete objects and associated over time.

More recently, Dewan et al. (2016) consider a similar problem to ours, estimating rigid scene flow between LIDAR scans. Similar to our work, they formulate an energy minimization problem based on matching SHOT feature descriptors for a subset of keypoints. However, unlike our work, they rely on point correspondences, where we do not rely on any data association. Additionally, they do not temporally filter this result over successive scans.

Our filtering framework has some similarities to that of Tanzmeister et al. (2014) and Danescu, Oniga, and Nedevschi (2011), where particles with position and speed are spread throughout a 2D grid and can move from cell to cell. They rely on stereovision to produce positional observations that are used in the resampling step for the particle filter. Aside from the difference in sensing modality, we attempt to directly exploit perceived motion in the sensor data.

3.3 Problem Statement

Our goal is to compute scene flow from LIDAR scans in real-time. Our work has direct application for autonomous vehicles, so we make the assumption that the world is locally planar—as is common in many autonomous vehicle applications. Namely, we assume that object motion is restricted to this horizontal plane and that all objects in a vertical column tangent to this plane move together. It is important to note that we make this assumption for our application domain to help run in real-time, though our proposed method could be modified to compute non-horizontal motion if need be. Thus, our goal is to compute the temporal scene flow $s_{i,j}$ for every location (i, j) in the plane.

Many sensor observations in a LIDAR scan correspond to static background structure in which scene flow estimation is trivial (e.g., the ground plane). As autonomous vehicles are commonly instrumented to estimate odometry, scene flow of static structures can instead be estimated via the relative motion of the platform vehicle. Thus, our work is primarily interested in estimating the scene flow for dynamic objects or potentially dynamic objects in the environment.

In the following sections, we describe the stages of our framework. Beginning with a point cloud, $\mathbf{z}_{t,1:n} = \left\{[x_i, y_i, z_i]^\top\right\}_{i=1}^{n}$ that we receive from the LIDAR sensor at time t, we first construct a 3D occupancy grid G_t. We will consider both $\mathbf{z}_{t,1:n}$ and G_t in the reference frame of the vehicle platform at time t. We then use a learned background filter to extract the foreground from G_t. Next, we estimate the scene flow between two successive occupancy grids G_{t-1} and G_t using a learned approach. We compute this flow in the reference frame of the vehicle, although this could easily be converted to the world frame using the odometry estimate of the ego-motion of the vehicle platform. Finally, we accumulate these raw scene flow measurements into a filtering framework to provide an estimate of the temporal scene flow.

3.4 LIDAR Preprocessing

In this section, we describe the preprocessing performed on the LIDAR sensor data, $\mathbf{z}_{t,1:n}$, to create an occupancy grid G_t.

3.4.1 Occupancy Grid

To process the LIDAR data, we first build an occupancy grid (Hornung et al., 2013; Moravec and Elfes, 1985), composed of 3D voxels representing how likely it is that an object occupies the given space. For each voxel, the probability it is occupied is computed by

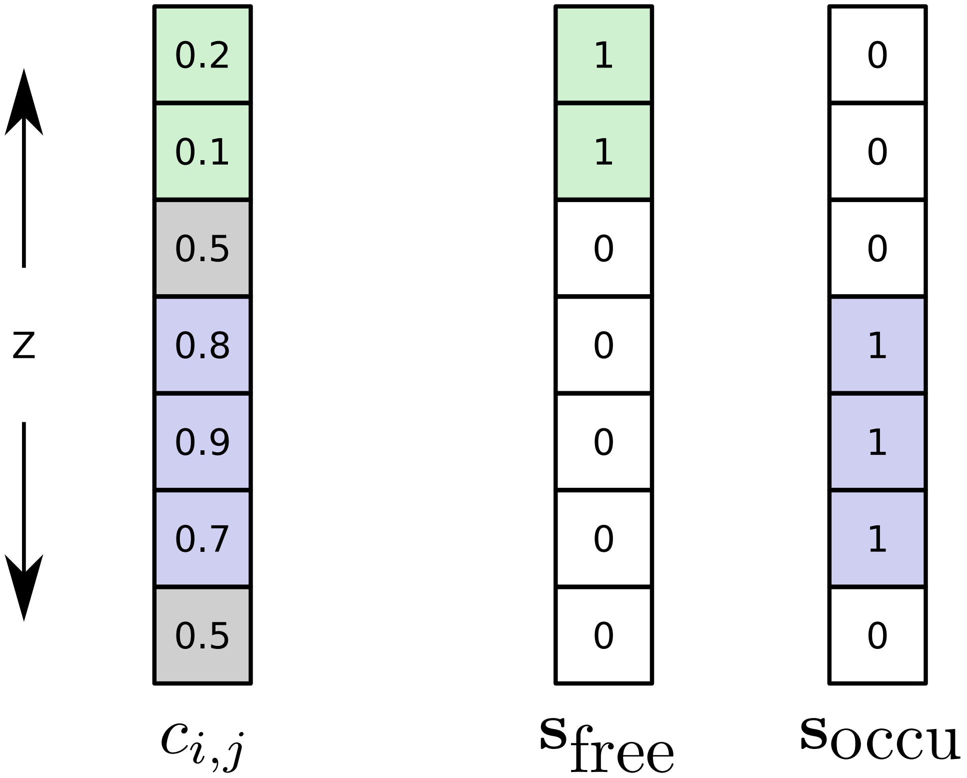
$$p(v|\mathbf{z}_{t,1:n}) = \left[1 + \frac{1 - p(v|\mathbf{z}_{t,n})}{p(v|\mathbf{z}_{t,n})} \frac{1 - p(v|\mathbf{z}_{t,1:n-1})}{p(v|\mathbf{z}_{t,1:n-1})} \beta \right]^{-1}, \tag{3.1}$$

where $\mathbf{z}_{t,1:n} = \{\mathbf{z}_{t,1}, \ldots, \mathbf{z}_{t,n}\}$ is the collection of laser returns from the LIDAR sensor, $\beta = \frac{p(v)}{1-p(v)}$, and $p(v)$ is a prior on the state of v. If we assume that our prior is $p(v) = 0.5$ and use log-odds (denoted L) (Hornung et al., 2013), we can represent the recursive formulation of (3.1) as

$$L(v|\mathbf{z}_{t,1:n}) = \sum_{i=1}^{n} L(v|\mathbf{z}_{t,i}) \tag{3.2}$$

$$L(v|\mathbf{z}_{t,i}) = \begin{cases} l_{\text{free}} & \text{the ray to } \mathbf{z}_{t,i} \text{ passes through } v \\ l_{\text{occupied}} & \text{the ray to } \mathbf{z}_{t,i} \text{ ends in } v \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

where $L(v|\mathbf{z}_{t,i})$ is the log-odds update given by the observation $\mathbf{z}_{t,i}$ and l_{free} and l_{occupied} are the log-odds updates given by an observation of free or occupied space, respectively. To compute all the of log-odds updates for all of our LIDAR observations, we use Bresenham's ray tracing algorithm (Bresenham, 1965), which can be efficiently implemented on the GPU. This is further discussed in Appendix B and yields a sparse mapping of voxel to occupancy probability. Note that the occupancy grid is sparse because the majority of voxels remain unknown.



(a) Input Column

(b) Output Feature Vectors

Figure 3.2: Background filter feature vectors.

3.4.2 Background Filter

In order to reduce the computational burden of computing flow for the entire scene, we first apply background subtraction to identify static structure. This preprocessing step identifies possible columns in the occupancy grid that could be dynamic and have some scene flow associated with them. While this filter need not be perfect, we find that it helps with runtime performance and errors due to perceptual aliasing.

Many techniques that use LIDAR sensors rely on various methods to eliminate static background (e.g., ground plane or buildings). One set of approaches rely on prior maps of the environment that are leveraged by localizing within these maps during runtime. These maps contain information about the ground plane or other static structure in the scene. For example, some obstacle trackers rely on these maps to identify only the sensor measurements from objects that need to be tracked (Ushani et al., 2015). However, these approaches fail if the map is not accurate (e.g., due to construction) or the localization system experiences any errors.

Other approaches leverage the appearance and structure of the data to discriminate between foreground and background, such as Wang, Posner, and Newman (2012, 2015). Features such as normal vectors and shape distributions are extracted from patches of points to create a handcrafted feature vector which is fed into a classifier. Wang, Posner, and Newman (2012) report a runtime of 5 s on 3D data, while Wang, Posner, and Newman (2015) report a runtime of 336 ms on 2D data. As our use requires a filter that is part of a full system that can run in under 100 ms, we propose a method that is simpler in nature but can run much faster than these previous techniques while still operating on our 3D occupancy grid.

We propose to solve this task via classification using a logistic classifier. Given a location (i, j) in our occupancy grid, we seek to find a label $y \in \{\text{Foreground}, \text{Background}\}$. This method requires no prior maps and allows us to learn from training data the structure of columns that appear to be dynamic.

For the occupancy column (i, j), we build a binary feature vector as follows. We extract 5×5 neighborhood patch of columns, $N_{i,j}$, from the occupancy grid around the location (i, j). We build two binary feature vectors $\mathbf{s}_{\text{free}}$ and $\mathbf{s}_{\text{occu}}$, each encoding the state of the full neighborhood patch. The binary elements of $\mathbf{s}_{\text{free}}$ and $\mathbf{s}_{\text{occu}}$ are given by

$$\mathbf{s}^n_{\text{free}} = p(v_n) < (0.5 - \epsilon) \tag{3.4}$$

$$\mathbf{s}^n_{\text{occu}} = p(v_n) > (0.5 + \epsilon), \tag{3.5}$$

for every voxel $v_n \in N_{i,j}$, where ϵ is a parameter controlling the certainty of free or occupied space. These vectors are concatenated to make one binary feature vector $\mathbf{s}_{i,j} = \left[{\mathbf{s}^n_{\text{free}}}^\top, {\mathbf{s}^n_{\text{occu}}}^\top\right]^\top$, which is then used in a logistic classifier,

$$x_{\text{filter}}(i, j) = \mathbf{w}^\top_{\text{filter}} \mathbf{s}_{i,j} + b_{\text{filter}} \tag{3.6}$$

$$P_{\text{filter}}(i, j) = \frac{1}{1 + \exp\bigl(-x_{\text{filter}}(i, j)\bigr)}. \tag{3.7}$$

Note that this feature vector implicitly captures the notion of unknown cell state where these two decision variables both evaluate to false.

To train this classifier and learn $\mathbf{w}_{\text{filter}}$ and b_{filter} , we extract training data from the KITTI dataset. For ten KITTI log sequences of data, we build feature vectors by sampling columns in the occupancy grid and extract labels by determining whether or not these columns are contained within the labeled KITTI tracklet data. We build a training set of 243,828 samples. We use TensorFlow (Abadi et al., 2015) to train our logistic classifier. Finally, we choose our decision threshold for this classifier to give us 95% accuracy on extracting the foreground; this allows us to reject much of the static background without catastrophically removing dynamic columns.

Our background filter can then be used online by running the classifier for every column of the occupancy grid. This step is embarrassingly parallel, and thus can be implemented very efficiently on the GPU. For any columns that are identified to be part of the background, we assign a raw scene flow measurement derived from the odometry estimate of the ego-motion of the vehicle platform.

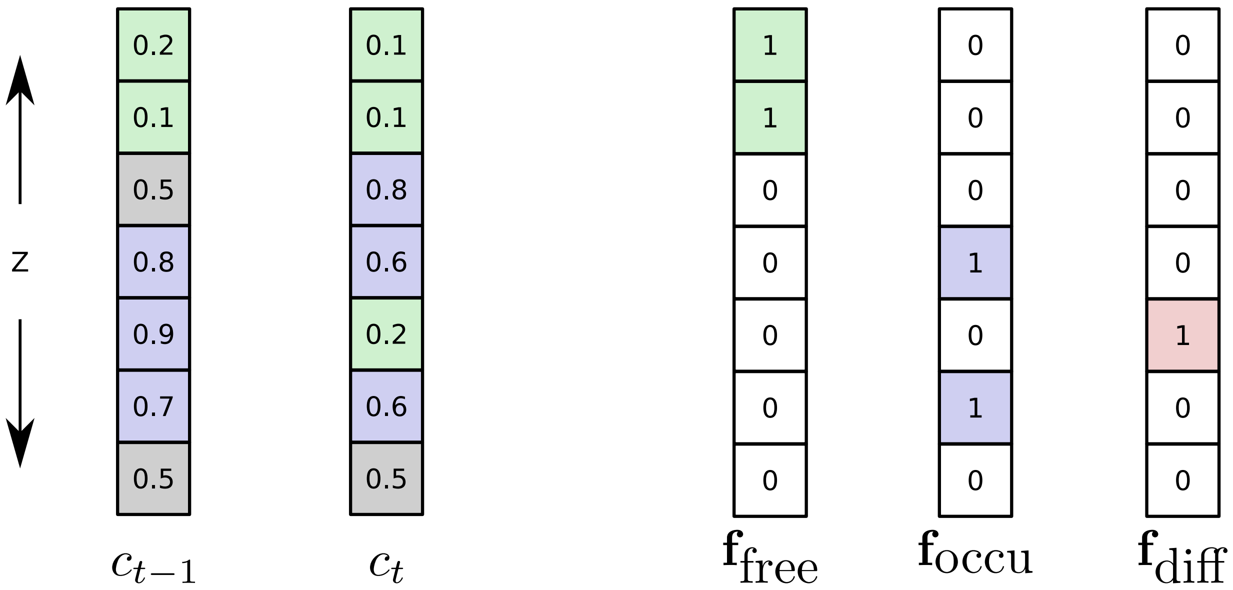
3.5 Raw Scene Flow Computation

In this section, we will describe the process by which we compute temporal scene flow between two occupancy grids G_{t-1} and G_t. We use the background filter described above to filter G_{t-1}, but importantly we do not filter G_t at this time. This helps mitigate errors in the background filter, as voxels that were not properly filtered in G_{t-1} can still be matched to the corresponding location in G_t.

3.5.1 Occupancy Constancy

To compute the scene flow between two occupancy grids, we need a method by which we can measure the consistency of the occupancy state of columns in successive occupancy grids. To do so, we will rely on *occupation constancy*: the occupation state of matching columns should not change between two successive scans. Here, we assume rigid, non-deforming motion, which is generally valid in our application. As we will demonstrate in the results, our method can still achieve good performance even for objects that can deform, such as cyclists and pedestrians.

We formulate occupation constancy as a learning problem, as we find that a learning approach works significantly better than a purely hand-designed metric. We take two candidate columns $c_{t-1} \in \mathrm{G}_{t-1}$ at (i_{t-1}, j_{t-1}) and $c_t \in \mathrm{G}_t$ at (i_t, j_t), each consisting of a vertical array of voxels $v_{i,j,k}$ in the occupancy grid at their respective location. We wish to determine whether or not they are consistent. We construct three binary feature vectors, $\mathbf{f}_{\text{free}}$, $\mathbf{f}_{\text{occu}}$, and $\mathbf{f}_{\text{diff}}$, encoding whether or not the two columns have similar occupation. Each


z
0.2
0.1
0.5
0.8
0.9
0.7
0.5
0.1
0.1
0.8
0.6
0.2
0.6
0.5
c_{t-1}
c_t
1
1
0
0
0
0
0
0
0
0
1
0
1
0
0
0
0
0
1
0
0
$\mathbf{f}_{\text{free}}$
$\mathbf{f}_{\text{occu}}$
$\mathbf{f}_{\text{diff}}$

(a) Input Columns

(b) Output Feature Vectors

Figure 3.3: Occupancy constancy feature vectors. In Fig. 3.3(a), we see two columns from successive occupancy grids, c_{t-1} and c_t. We wish to build a feature vector that describes the consistency of the occupancy states between the two. We build three feature vectors, shown in Fig. 3.3(b), to capture consistencies in free and occupied space and any inconsistencies. Note that unknown regions (in gray) are effectively ignored. These three vectors are then concatenated into the feature vector $\mathbf{f}_{c_{t-1},c_t}$.

element k of these feature vectors are given by

$$\begin{aligned}
\mathbf{f}_{\text{free}}^{(k)} &= \big(p(v_{c_{t-1},k}) < (0.5-\epsilon)\big) \wedge \big(p(v_{c_t,k}) < (0.5-\epsilon)\big) && (3.8)\\
\mathbf{f}_{\text{occu}}^{(k)} &= \big(p(v_{c_{t-1},k}) > (0.5+\epsilon)\big) \wedge \big(p(v_{c_t,k}) > (0.5+\epsilon)\big) && (3.9)\\
\mathbf{f}_{\text{diff}}^{(k)} &= \Big(\big(p(v_{c_{t-1},k}) > 0.5+\epsilon\big) \wedge \big(p(v_{c_t,k}) < 0.5-\epsilon\big)\Big)\\
&\quad \vee \Big(\big(p(v_{c_{t-1},k}) < 0.5-\epsilon\big) \wedge \big(p(v_{c_t,k}) > 0.5+\epsilon\big)\Big). && (3.10)
\end{aligned}$$

We concatenate these three binary feature vectors into one binary feature vector $\mathbf{f}_{c_{t-1},c_t}$. This is then used in a logistic classifier,

$$\begin{aligned}
x_{\text{match}}(c_{t-1}, c_t) &= \mathbf{w}_{\text{match}}^{\top} \mathbf{f}_{c_{t-1},c_t} + b_{\text{match}} && (3.11)\\
P_{\text{match}}(c_{t-1}, c_t) &= \frac{1}{1+\exp\big(-x_{\text{match}}(c_{t-1}, c_t)\big)}, && (3.12)
\end{aligned}$$

which yields the probability that the two given columns are consistent with each other given their occupation state.

To train this classifier and learn $\mathbf{w}_{\text{match}}$ and b_{match}, we again rely on the KITTI dataset to extract training data. We create a training dataset in the following manner. We sample a column $c_{t-1} \in \mathrm{G}_{t-1}$ at location (i, j). Using scan matching Segal, Haehnel, and Thrun (2009) for a ground truth relative pose estimate and the labeled KITTI tracklet data, we compute the true scene flow and find the true corresponding column $c_t \in \mathrm{G}_t$. Thus, c_{t-1} and c_t are used to construct a positive training sample. We then additionally sample from a $n_s \times n_s$ neighborhood about location (i, j) in G_t (denoted by $N_{c_{t-1}}$), excluding the true corresponding column, to construct negative training samples.

We use the same ten KITTI log sequences of data as we did before to build our training dataset, comprising of 1,028,381 training samples. We train our logistic classifier for occupancy constancy using TensorFlow.

For efficient computation, we preprocess P_{match} for all pairs of columns we will consider. We take the log-probability and apply a window to promote spatial smoothness (e.g., $c_{i,j}$ and $c_{i+1,j+1}$ will likely experience similar scene flow). We store this result in a lookup table T_{match},

$$T_{\text{match}}(c_{t-1}, c_t) = \sum_{c_w \in W_{c_{t-1}}} \log \big(P_{\text{match}}(\mathbf{f}_{c_w, c_t}) \big), \tag{3.13}$$

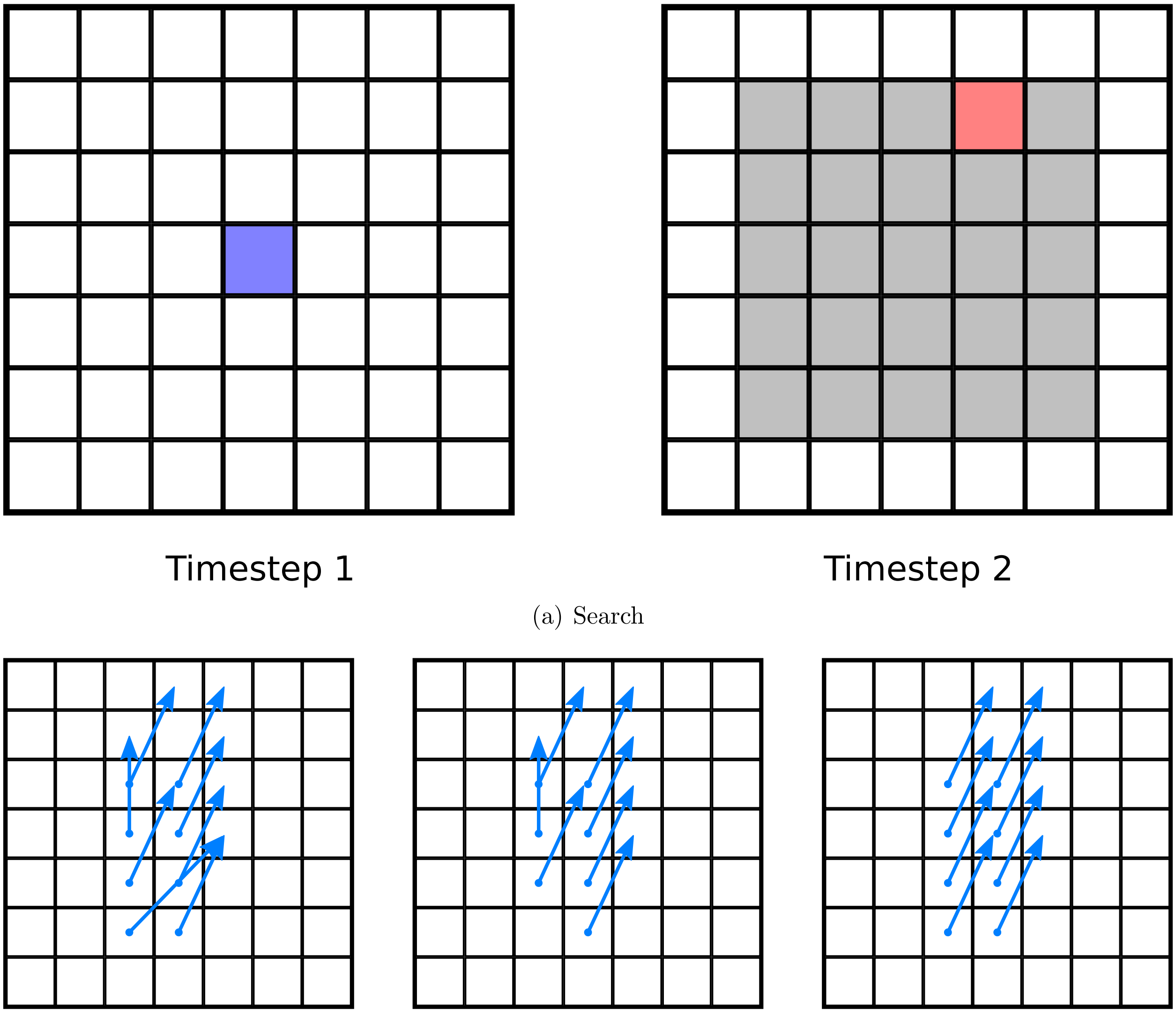
where $W_{c_{t-1}}$ is $n_w \times n_w$ window of columns $c_w \in \mathrm{G}_{t-1}$ centered on c_{t-1}.

3.5.2 Raw Scene Flow Computation

To solve for the scene flow, we formulate an energy minimization problem that leverages our learned occupancy constancy metric. Our energy minimization problem is equivalent to a Markov random field (MRF). Similar problems have been extensively studied in the field, with various methods for minimizing the energy (Szeliski et al., 2006, 2008). Methods such as graph cuts, loopy belief propagation (LBP), and tree-reweighted message passing have been applied, with various tradeoffs in the quality of the solution and the runtime.

In this work, we use an iterative EM algorithm to solve the energy minimization problem within our runtime constraints, which we run for a maximum of n_{em} iterations. By doing so, we estimate a locally rigid, non-deforming flow between successive occupancy grids. This computation process is shown in Fig. 3.4.

At each step of the EM algorithm, for every $c_{t-1} \in \mathrm{G}_{t-1}$ we maintain the current estimate of scene flow $s(c_{t-1})$ and matched column $m(c_{t-1}) \in \mathrm{G}_t$, both initially all flagged as invalid. We will compute an energy, $E(c_{t-1}, c_t)$, associated with a potential scene flow estimate that leads from $c_{t-1} \in \mathrm{G}_{t-1}$ to $c_t \in \mathrm{G}_t$. Additionally, for every $c_t \in \mathrm{G}_t$, we store the energy associated with the currently computed scene flow estimate that leads to $c_t \in \mathrm{G}_t$, $\hat{E}(c_t)$,



(a) Search

(b) Expectation

(c) Maximization

(d) Expectation

Figure 3.4: An overview of the raw scene flow computation process. In Fig. 3.4(a), for a given location at a timestep (shown in blue), we search a region in the next timestep (shown in gray) to find the scene flow estimate that minimizes the energy (shown in red). In the expectation step shown in Fig. 3.4(b), we perform this operation for all locations to find a scene flow estimate for each one (shown by the blue arrows). In the maximization step shown in Fig. 3.4(c), we enforce our assumption of rigid scene flow. In the following expectation step in Fig. 3.4(d), we recompute energies to reestimate scene flow, which may be updated due to the smoothing term or previous invalidation.

which is initialized to ∞.

This algorithm must be run for every column in the occupancy grid. However, we can process columns in parallel, and thus this can be efficiently implemented on the GPU.

3.5.2.1 Expectation

During the expectation step, we estimate the most likely scene flow for every column. For a given column $c_{t-1} \in \mathrm{G}_{t-1}$ at location (i, j), we search through a neighborhood $N_{c_{t-1}}$, which is a $n_s \times n_s$ neighborhood of columns $c_t \in \mathrm{G}_t$ centered around location (i, j). We compute an energy for each $c_t \in N_{c_{t-1}}$,

$$E(c_{t-1}, c_t) = \underbrace{-T_{\text{match}}(c_{t-1}, c_t)}_{\text{constancy metric}} + \underbrace{w_p \sum_{c_p \in P_{c_{t-1}}} \left\| s_{c_{t-1},c_t} - s(c_p) \right\|^2}_{\text{regularization}}, \tag{3.14}$$

where w_p is an L2 penalty weight to enforce spatial smoothness, $P_{c_{t-1}}$ contains all $c_p \in \mathrm{G}_{t-1}$ in a 5×5 neighborhood around c_{t-1} for which we have a valid scene flow estimate $s(c_p)$, and s_{c_{t-1},c_t} is the candidate scene flow vector which associates c_{t-1} and c_t.

For every $c_{t-1} \in \mathrm{G}_{t-1}$, we find the corresponding $c_t^* \in \mathrm{G}_t$ that minimizes $E(c_{t-1}, c_t)$ such that either $E(c_{t-1}, c_t^*) < \hat{E}(c_t^*)$ or $m(c_{t-1}) = c_t^*$ (i.e., the same scene flow was computed previously during an earlier iteration and we simply need to update the energy that may have changed due to the L2 penalty). We store the estimated scene flow at this iteration in $s(c_{t-1}) = s_{c_{t-1},c_t^*}$ and $m(c_{t-1}) = c_t^*$.

3.5.2.2 Maximization

During the maximization we step, we enforce our assumption of locally rigid, non-deforming flow. This means that only one column $c_{t-1} \in \mathrm{G}_{t-1}$ can lead to any column $c_t \in \mathrm{G}_t$.

For each $c_t \in \mathrm{G}_t$, we consider all $c_{t-1} \in \mathrm{G}_{t-1}$ such that $m(c_{t-1}) = c_t$. If there are any such c_{t-1}, we take the one with the lowest energy $E(c_{t-1}, c_t)$, denoted c_{t-1}^*, essentially picking the most likely column in the previous scan that leads to c_t. We set $\hat{E}_c(c_t) = E(c_{t-1}^*, c_t)$, and we flag all other $m(c_{t-1}) = c_t$ and $s(c_{t-1})$ where $c_{t-1} \neq c_{t-1}^*$ as being invalidated.

3.5.3 GPU Implementation

In this section, we describe the details of the GPU implementation for the energy minimization algorithm. While these details do not affect the result of the flow estimate, it is critical for real-time performance to take advantage of GPU hardware optimizations. Chief among these is making use of global memory coalescing and shared memory per block of Compute Unified Device Architecture (CUDA) threads. Further details on GPU programming are provided in Appendix A.

We preallocate several memory buffers in GPU global memory. These are reused for each flow computation, and allows us to avoid the overhead of memory allocation. These buffers include:

- A dense 3D copy of the occupancy grids G_{t-1} and G_t
- Logistic classifier outputs $P_{\text{match}}(c_{t-1}, c_t)$
- Occupancy constancy scores $T_{\text{match}}(c_{t-1}, c_t)$
- Energies $E(c_{t-1}, c_t)$ and $\hat{E}(c_t)$
- Scene flow $s(c_{t-1})$

Note that these memory buffers use 4 byte types (i.e., `float` or `int32_t`) even if they do not require 4 bytes of storage. This is due to GPU memory operations and global memory coalescing being optimized for data types of this size.

First, the sparse occupancy grids from Section 3.4.1 are used to populate dense occupancy grids G_{t-1} and G_t. Note that the background filter from Section 3.4.2 has been applied to create G_{t-1}. To take advantage of coalescing memory operations on the GPU, the memory for these dense grids is indexed so that the vertical dimension is the major dimension. The dense population can be achieved with the use of a CUDA kernel where each GPU thread handles one voxel in the grid.

Next, we compute occupancy constancy scores and populate P_{match} and T_{match}. We first construct a CUDA kernel to compute P_{match}. This kernel is configured so that each CUDA block handles a particular c_{t-1}, and each thread within that block handles some $c_t \in W_{c_{t-1}}$. This allows us to load c_{t-1} once per block and store it in shared memory, minimizing the need for slow global memory accesses. Each thread then computes P_{match} for its pair of c_{t-1} and c_t. A second CUDA kernel applies the windowing operation to compute T_{match}. To later take advantage of memory coalescing, T_{match} is indexed such that c_t is the major dimension.

Now, we are ready to perform the energy minimization. We have a CUDA kernel for each of the expectation and maximization steps.

For expectation, we configure the CUDA kernel such that each thread handles one c_{t-1}. Each block of threads handles a 2D array of nearby c_{t-1}'s. We first load into shared memory the current estimate of the scene flow for nearby locations (to help compute the L2 spatial smoothness penalty) and $\hat{E}$. By using shared memory, we minimize slow global GPU memory operations. Each thread then performs the expectation step according to Section 3.5.2.1. Note that if c_{t-1} has been identified as background by our filter, we can save on runtime by skipping this location (although care must be taken to still use the thread to help load data into shared memory).

For maximization, we use a similarly configured CUDA kernel. Each thread handles one c_{t-1}, with each block of threads handling a 2D neighborhood of c_{t-1}'s. We load current estimates of energies and scene flow into shared memory to again minimize the number of slow global GPU memory operations that are necessary. Once this is done, each CUDA kernel performs the maximization step according to Section 3.5.2.2.

Once we are finished, the scene flow estimate $s(c_{t-1})$ is copied from GPU memory to host memory. Note that this is only done once, at the end of the process, to minimize memory bottlenecks.

3.6 Temporal Scene Flow

We further refine this raw scene flow measurement by incorporating it into a filtering framework. We maintain a 2D array of *flow tracklets*, each maintaining a temporally filtered estimate of the scene flow at the given position (i, j). This step in our procedure is somewhat similar to the work presented in Danescu, Oniga, and Nedevschi (2011) and Tanzmeister et al. (2014). This filtering process is depicted in Fig. 3.5.

3.6.1 Flow Tracklets Filter

Each flow tracklet is essentially an extended Kalman filter (EKF) using a constant velocity model. We incorporate the raw flow measurements by treating them as x, y observations, with a covariance determined by the resolution of the occupancy grid. We use Mahalanobis gating to reject any flow measurements that are outliers.

We represent the state as:

$$\boldsymbol{\mu}_k = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \dot{\theta} \end{bmatrix}, \tag{3.15}$$

where (x, y) is the position of the flow tracklet, θ is its orientation, v is its forward speed, and $\dot{\theta}$ is its turning rate. The nonlinear process model is given by:

$$f(\boldsymbol{\mu}_{k-1}) = \boldsymbol{\mu}_{k-1} + \begin{bmatrix} v_{k-1}\cos\theta_{k-1} \\ v_{k-1}\sin\theta_{k-1} \\ \dot{\theta}_{k-1} \\ 0 \\ 0 \end{bmatrix} \Delta t. \tag{3.16}$$

The observation model is given by:

$$\mathrm{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.17}$$

$$\mathbf{z_k} = \mathrm{H}_k \boldsymbol{\mu}_k \tag{3.18}$$

$$= \begin{bmatrix} x \\ y \end{bmatrix}. \tag{3.19}$$

Thus, the prediction step in our EKF filter becomes:

$$\bar{\boldsymbol{\mu}}_k = f(\boldsymbol{\mu}_{k-1}) \tag{3.20}$$

$$\bar{\Sigma}_k = \mathrm{F}_k \Sigma_{k-1} \mathrm{F}_k^\top + \mathrm{Q}_k, \tag{3.21}$$

where F_k is the linearized Jacobian of $f(\,\cdot\,)$ with respect to $\boldsymbol{\mu}_{k-1}$, Σ_k and Σ_{k-1} is the covariance of our estimate for $\boldsymbol{\mu}_k$ and $\boldsymbol{\mu}_{k-1}$, respectively, and Q_k is our process model uncertainty.

The update step in our EKF filter is:

$$\mathrm{K}_k = \bar{\Sigma}_k \mathrm{H}_k^\top \big(\mathrm{H}_k \bar{\Sigma}_k \mathrm{H}_k^\top + R_k\big)^{-1} \tag{3.22}$$

$$\boldsymbol{\mu}_k = \bar{\mu}_k + \mathrm{K}_k \big(\mathbf{z}_k - \mathrm{H}_k \bar{\boldsymbol{\mu}}_k\big) \tag{3.23}$$

$$\Sigma_k = \big(\mathrm{I} - \mathrm{K}_k \mathrm{H}_k\big) \bar{\Sigma}_k \big(\mathrm{I} - \mathrm{K}_k \mathrm{H}_k\big)^\top + \mathrm{K}_k \mathrm{R}_k \mathrm{K}_k^\top, \tag{3.24}$$

where R_k is our observation uncertainty, K_k is the Kalman gain, and I is the identity matrix. We use a fixed measurement uncertainty R_k determined based on the resolution of our occupancy grid. However, this potentially could be computed using the computed energies when determined raw scene flow.

To apply Mahalanobis gating, we compute the Mahalanobis distance,

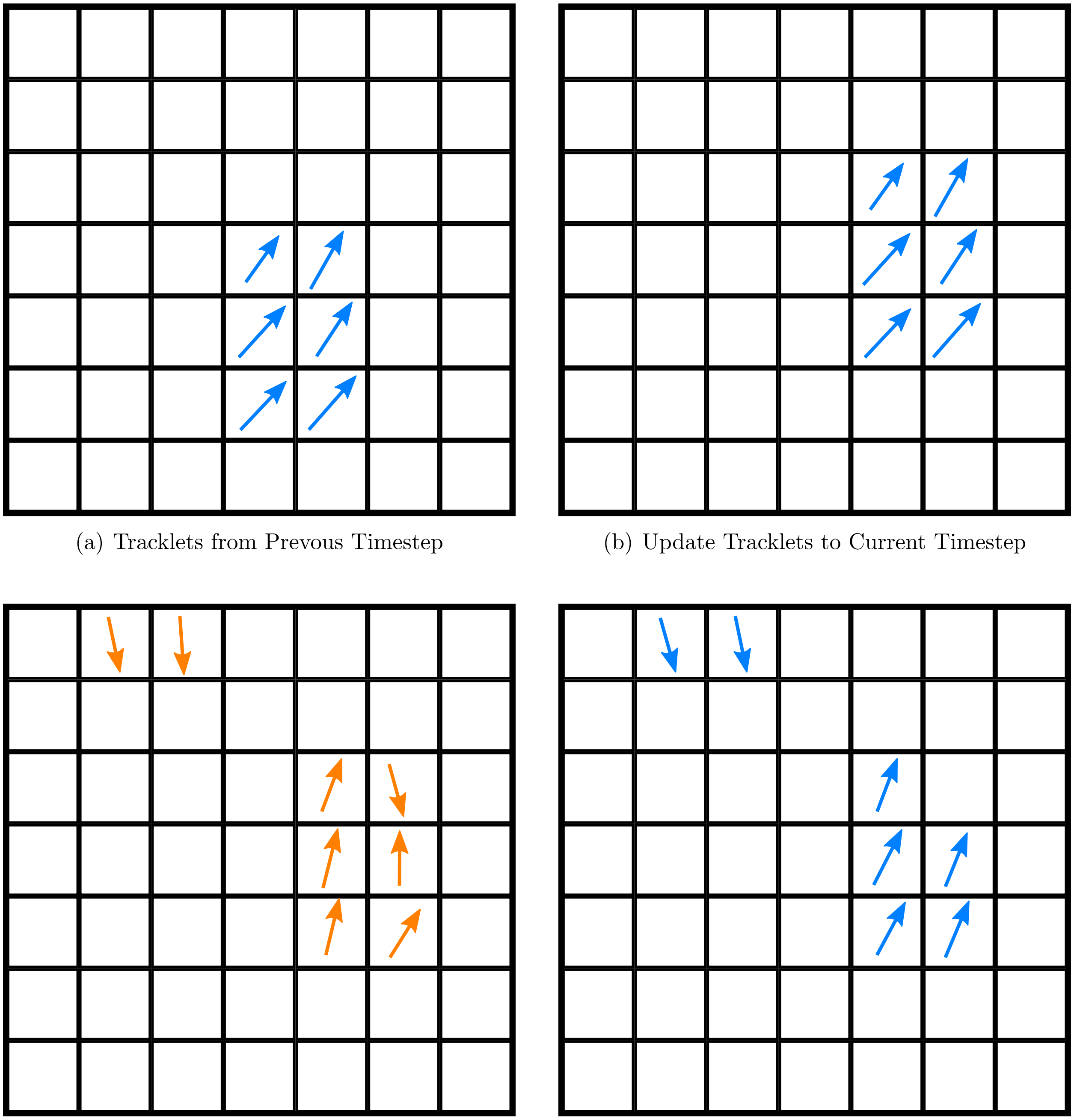
$$d_k = \sqrt{\big(\mathbf{z}_k - \mathrm{H}_k \bar{\mu}_k\big)^\top \big(\mathrm{H}_k \bar{\Sigma}_k \mathrm{H}_k^\top + R_k\big)^{-1} \big(\mathbf{z}_k - \mathrm{H}_k \bar{\mu}_k\big)}, \tag{3.25}$$

and compare this value to a chosen threshold. If the Mahalanobis distance exceeds our threshold, then the observation is flagged as being an outlier.

3.6.2 Flow Tracklets Array

After we compute the raw scene flow between G_{t-1} and G_t, we process this result with our 2D array of flow tracklets. First, we use the constant velocity process model to update all of our flow tracklets to the current time. Then, each raw scene flow measurement for each location (i, j) is assigned to a flow tracklet at that location and used for the EKF filter's observation update. If no such flow tracklet exists at the location (i, j), a new one is created. Flow tracklets without a valid raw scene flow measurement (for example, due to Mahalanobis gating or background filtering) are discarded. Finally, all flow tracklets are moved to their new location (i, j) in the 2D array as given by the scene flow.

Not only does maintaining this filter better estimate the scene flow, but it helps eliminate outliers, both due to the background filter missing static background and any erroneous raw scene flow values. Additionally, for each flow tracklet, we maintain a count of how many observations it has received, or its “age”. As flow tracklets increase in age and incorporate more raw observations, their estimate of the scene flow becomes more reliable.



(a) Tracklets from Prevous Timestep

(b) Update Tracklets to Current Timestep

(c) Raw Scene Flow Observation

(d) Update Tracklets with Current Observation

Figure 3.5: An overview of the temporal scene flow filtering process. In Fig. 3.5(a), we see the flow tracklets at the previous timestep. In Fig. 3.5(b), these are updated to the current timestep according to their process model, and then they are move to their new locations. Now, we wish to process a raw scene flow observation as shown in Fig. 3.5(c). In Fig. 3.5(d), tracklets have been updated with the current observation, initialized from new observations, or discarded due to Mahalanobis gating.

3.7 Motion Compensation

Note that the estimates of scene flow and temporal scene flow are in the reference frame of the ego-vehicle. If these motion vehicles are desired in the global frame, the appropriate coordinate frame transform must be applied. This can easily be estimated from odometry sensors that instrument the platform vehicle's motion.

3.8 Results

We evaluated our proposed method using the KITTI dataset (Geiger et al., 2013), using sequences of raw data from city driving. For computation, we ran all of our experiments on a machine with an Intel i7-4790K CPU with 16 GB of memory and a NVIDIA GeForce GTX 1080 GPU.

3.8.1 Parameter Selection

For the occupancy grid, we construct a 50 m × 50 m grid centered on the vehicle at a resolution of 30 cm.

To construct our binary feature vectors, we use $\epsilon = 0$. Note that this does not make our feature vectors complimentary. Any voxel v that represents unknown space will have $p(v) = 0.5$, and thus will not factor into any of the binary features due to the strict inequality in the decision variable. This is in fact what allows us to implicitly account for unknown space.

We use a search space size of $n_s = 31$, allowing for a relative motion estimation of up to 45 m/s. We use a window size of $n_w = 3$. We run our EM algorithm for $n_{em} = 20$ iterations. For the L2 penalty weight, we use a value of $w_p = 1$.

3.8.2 Background Filter

We first evaluate the performance of our background filter. We show the precision-recall curve by sweeping out the decision threshold for our classifier, as shown in Fig. 3.6; here the foreground is the positive class. As discussed in Section 3.4.2, we choose a decision threshold for our classifier to achieve a 95% accuracy rate on foreground (as indicated by the *red* dot). This results in a 74.5% accuracy on the background.

While our background filter may not achieve state-of-the-art performance, it still performs quite well with respect to other methods Wang, Posner, and Newman (2012, 2015). However,

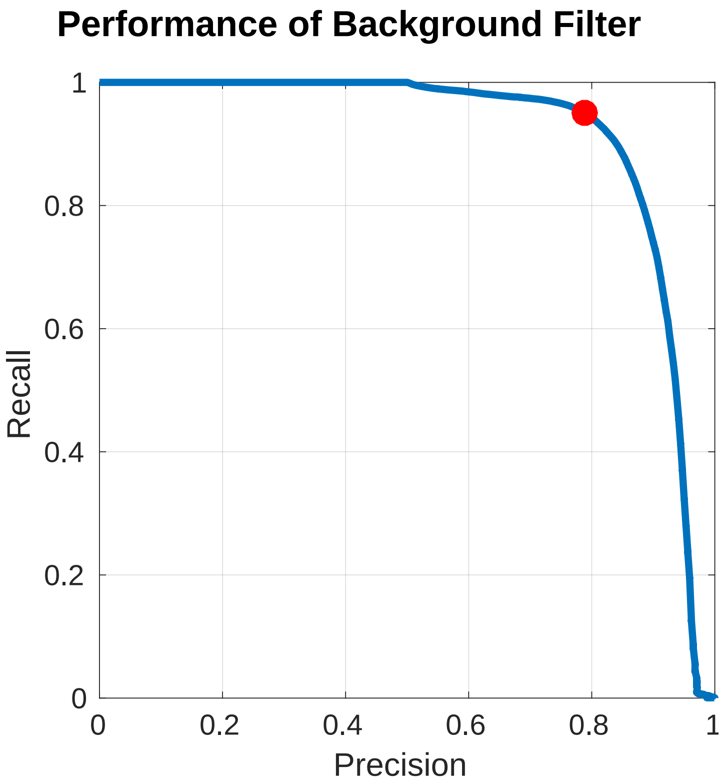


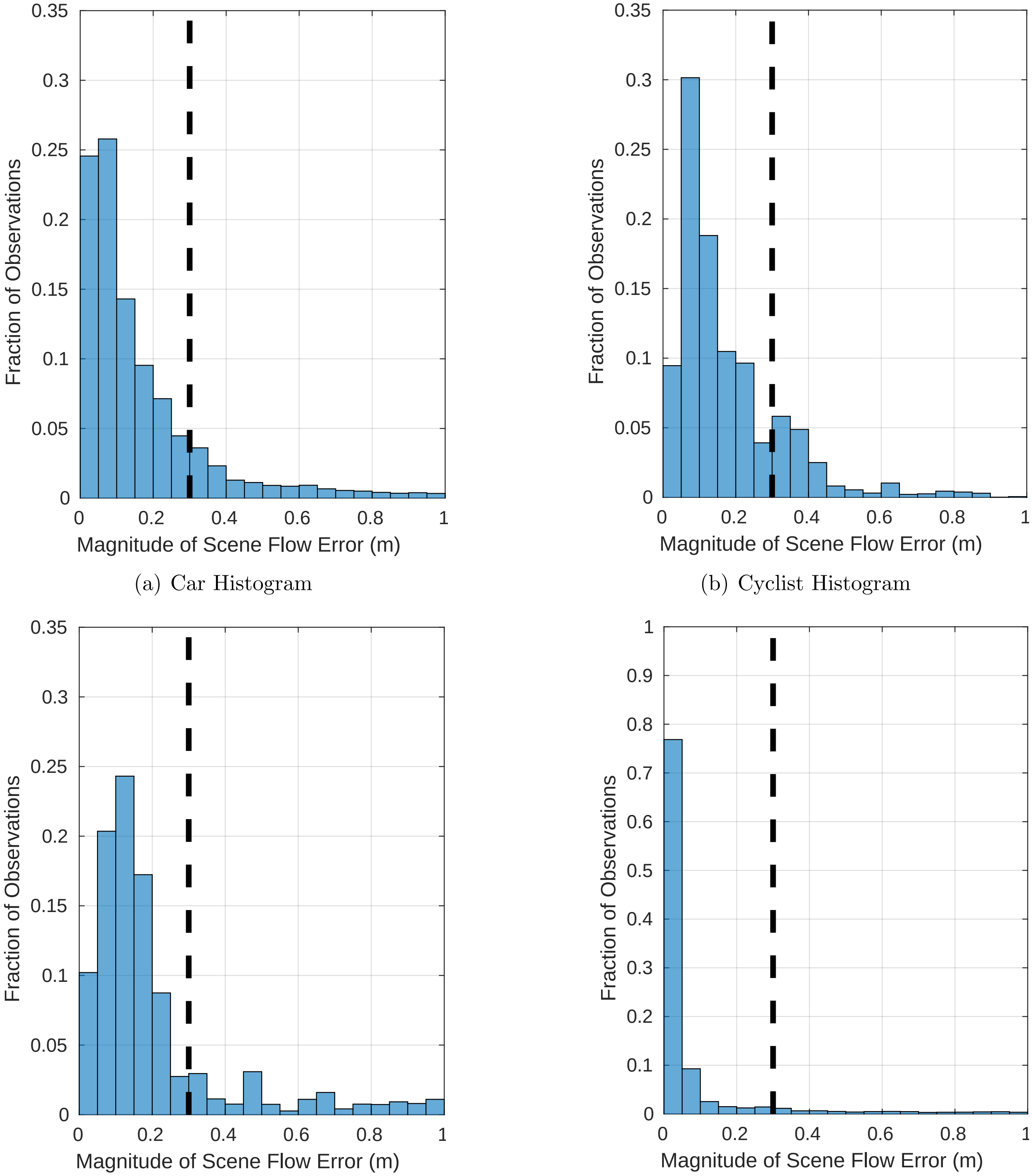
Figure 3.6: Precision-recall curve of the background filter. The red dot on the curve indicates the decision threshold chosen throughout our evaluation, where we detect 95% of the foreground.

a core strength of our method is in its efficiency, where we are able to execute extremely fast, taking only 1.2 ms to run.

3.8.3 Raw Scene Flow

Next, we evaluate our raw scene flow measurements. To generate the ground truth scene flow values, we combine the KITTI labeled tracklet data with the relative pose between the two successive LIDAR scans, as computed by scan matching Segal, Haehnel, and Thrun (2009). We then compute the norm of the error between the ground truth flow vector and our computed raw scene flow. Results are presented as error histograms by class in Fig. 3.7 and overall in Fig. 3.8. Computed error statistics are tabulated in Table 3.1.

We find that our raw scene flow measurements are generally quite accurate. While there are a number of outliers that arise from the aliasing of occupancy constancy, the error is frequently less than the resolution of the occupancy grid. Additionally, we find that we are still able to compute scene flow for objects that violate our assumption of rigid, non-deforming motion such as cyclists or pedestrians that move their arms and legs.



(a) Car Histogram

(b) Cyclist Histogram

(c) Pedestrian Histogram

(d) Background Histogram

Figure 3.7: Error histograms for raw scene flow measurements by class. The dashed vertical line in all plots indicates the resolution of our occupancy grid. Note the different vertical scale for Fig. 3.7(d).

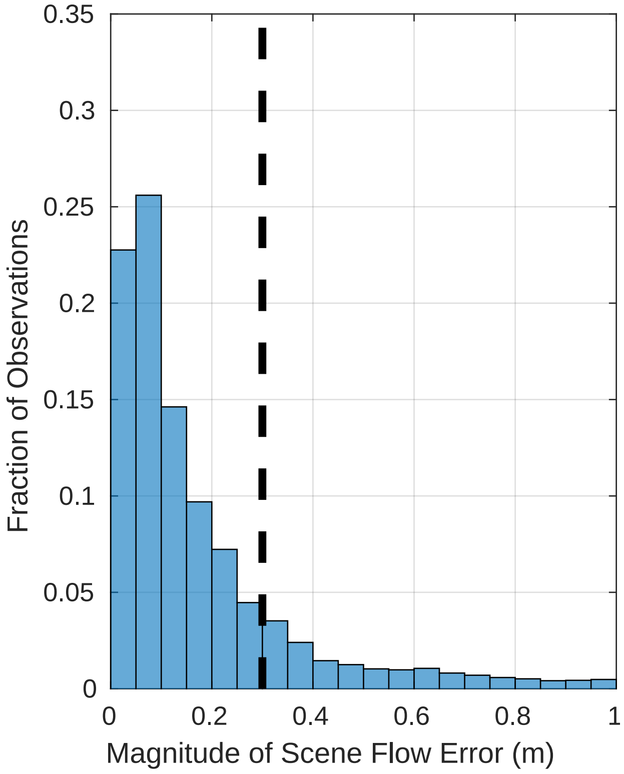


Figure 3.8: Error histogram for raw scene flow measurements for all foreground. The dashed vertical line indicates the resolution of our occupancy grid.

3.8.4 Temporal Scene Flow

Finally, we evaluate the temporal scene flow as computed by our flow tracklets against the same benchmark as discussed in Section 3.8.3. We evaluate this error as the tracklets age, which allows for the filtered scene flow estimate to become more and more accurate. The age of each tracklet as measured in these results is simply the number of scans that the tracklet has been active for (i.e., each age step corresponds to 0.1 s of sensor data). We compute the cumulative distribution function (CDF) of the norm of the velocity error vector for each flow tracklet for various ages and also for all tracklets that have been seen for at least a full second. We show these results in Fig. 3.9 and present error statistics in Table 3.2.

Unsurprisingly, the accuracy of the temporal scene flow steadily improves as tracklets get older and receive more observations. By the time a flow tracklet has been observed for at least a second, the median error is roughly 0.5 m/s, which corresponds to one sixth of the resolution of our occupancy grid.

A few sample temporal scene flow results are depicted in Fig. 3.10, from when the platform vehicle is stationary, and Fig. 3.11, from when the platform vehicle is moving. The flow

Class	Count	Median Error	Mean Error	Percent Within 30 cm
Car	873,947	10.2 cm	19.3 cm	83.8%
Cyclist	9,890	13.2 cm	24.5 cm	78.8%
Pedestrian	7,527	15.5 cm	38.9 cm	74.3%
Background	19,899,708	2.6 cm	14.9 cm	88.9%
All Labeled Tracklets	1,126,264	11.0 cm	22.1 cm	81.4%

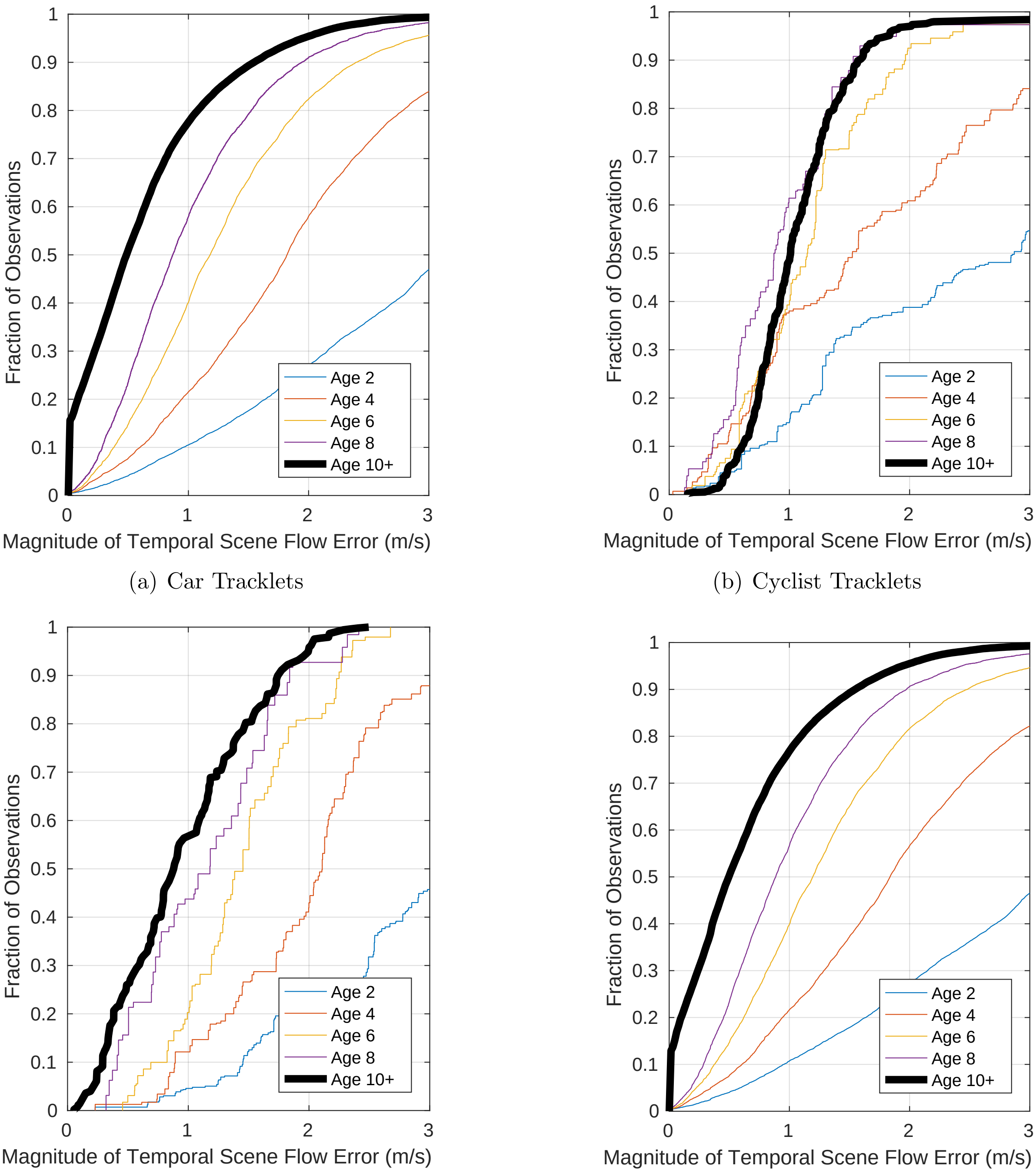
Table 3.1: Error statistics for raw scene flow measurements. We perform this analysis by class and overall labeled tracklets in the KITTI datasets. Note that the background can have non-zero error due to inaccuracies in the background filter, noisy odometry, or discretization effects.

Class	Count	Median Error	Mean Error
Car	216,026	0.49 m/s	0.65 m/s
Cyclist	1,507	1.01 m/s	1.09 m/s
Pedestrian	537	0.86 m/s	0.93 m/s
All Labeled Tracklets	266,237	0.50 m/s	0.66 m/s

Table 3.2: Error statistics for temporal scene flow measurements. We compute the error for all flow tracklets with an age of at least 10. We perform this analysis by class and over all labeled tracklets in the KITTI datasets.

estimate is shown in red and overlaid on top of the point cloud in blue. In both figures, the temporal scene flow is in the global world frame.

In Fig. 3.10(a), we see an vehicle turning in front of the platform vehicle. In Fig. 3.10(b), we see two vehicle passing from left to right at an intersection. In Fig. 3.11(a), the platform vehicle overtakes a car to its left while a bicycle travels towards the upcoming intersection on the right. In Fig. 3.11(b), the platform vehicle passes several cars parked on the side of the road. Two bicyclists are traveling on the sidewalk while a car slowly drives towards the platform vehicle.



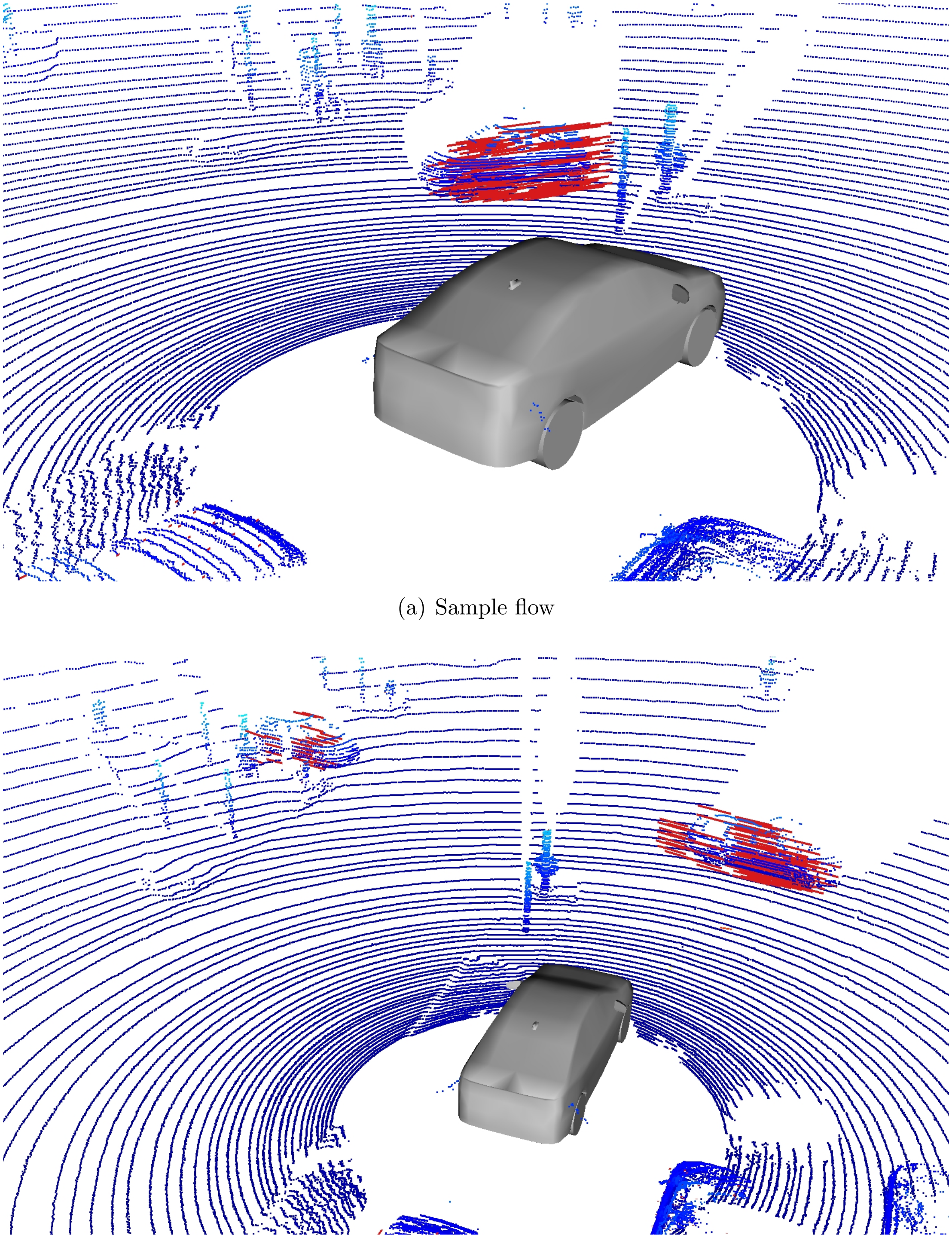
(a) Car Tracklets

(b) Cyclist Tracklets

(c) Pedestrian Tracklets

(d) Foreground Tracklets

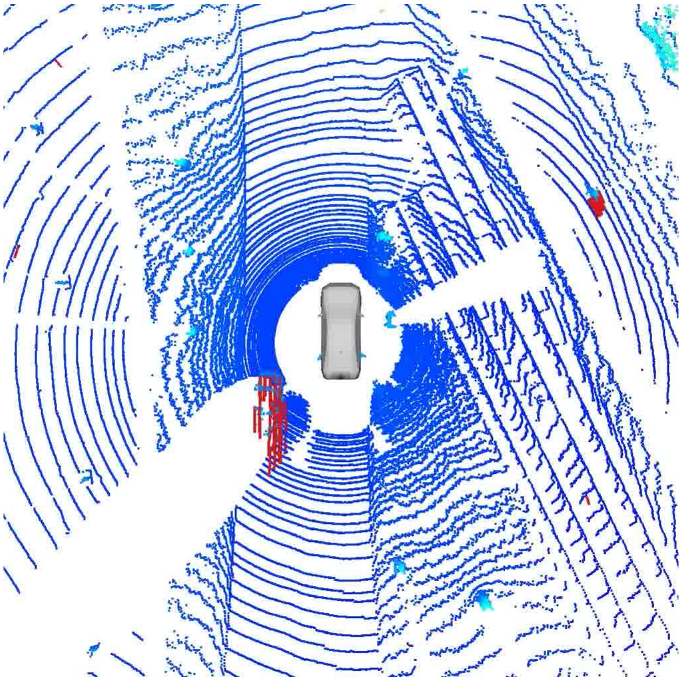
Figure 3.9: The CDF of the error of our temporal scene flow estimates. We present these results by class, and for all labeled KITTI tracklets. Note that these results are measured in terms of velocity (m/s) as opposed to the results in Fig. 3.7 and Fig. 3.8 which are measured in terms of position (m).



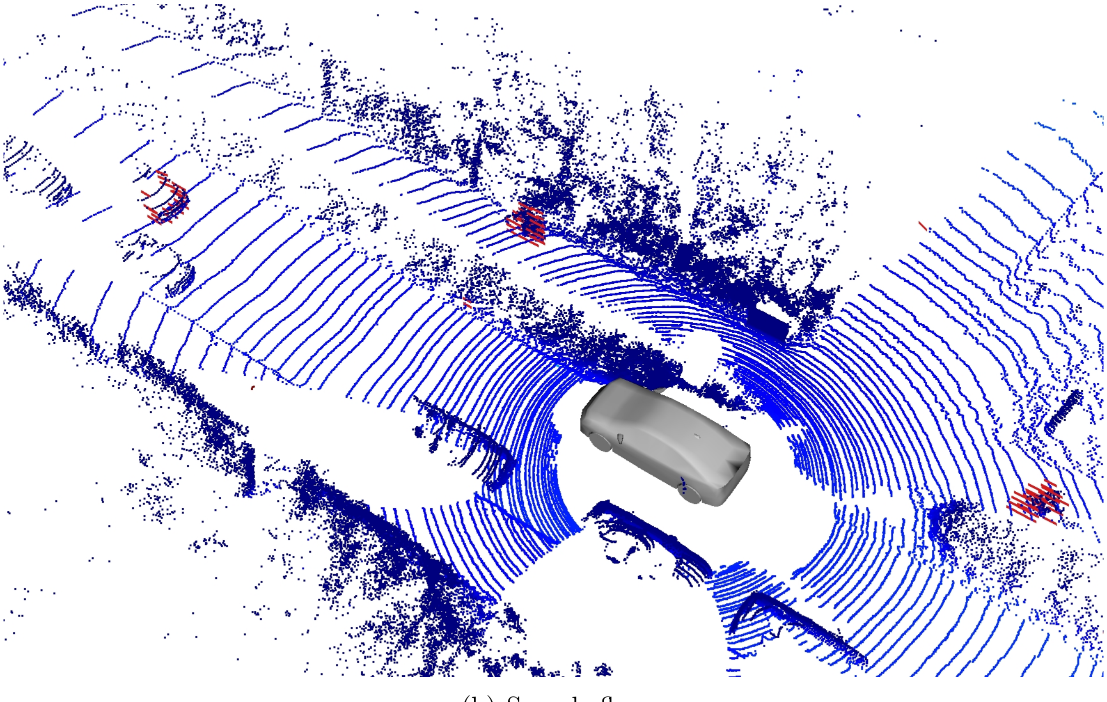
(a) Sample flow

(b) Sample flow

Figure 3.10: Sample temporal scene flow results when the platform is stationary. The temporal scene flow is depicted in red. Note that the platform vehicle is stationary in these examples.



(a) Sample flow



(b) Sample flow

Figure 3.11: Sample temporal scene flow results when the platform is moving. The temporal scene flow is depicted in red. Note that the platform vehicle is dynamic in these examples and flow is shown in the world frame.

3.8.5 Motion Compensation

Our proposed system computes raw scene flow and temporal scene flow in the frame of the platform vehicle. However, given an estimate for the movement of the ego-vehicle, for example from an odometry sensor, this flow estimate can be motion compensated and transformed into the global world frame. We present an example of this in Fig. 3.12, where the ego car is moving from the right to the left of the image.

Fig. 3.12(a) depicts temporal scene flow estimates in the ego frame. Note that the cars immediately in front of and behind the platform vehicle appear to have little to no flow, as they are traveling at approximately the same velocity. However, non-moving objects, such as stationary cars or static background structure that has not been filtered, appear to have motion opposite to the velocity of the platform vehicle.

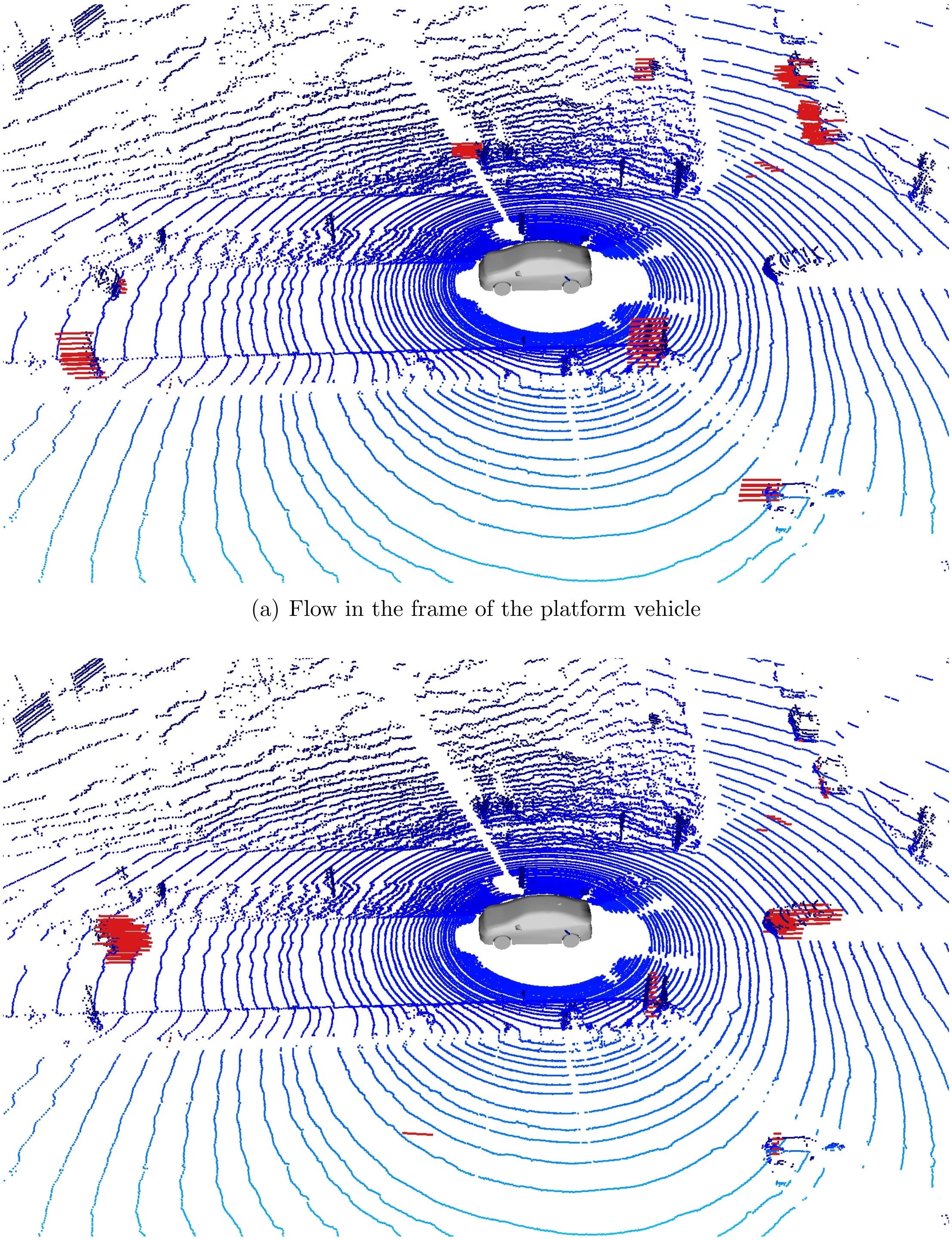
Fig. 3.12(b) depicts temporal scene flow estimates in the global world frame. Note that stationary objects now have little to no flow. The moving cars in front or behind the platform vehicle now have estimated temporal scene flow vectors that are approximately equal to the velocity of the platform vehicle.

3.8.6 Runtime Performance

A key goal is to enable real-time use of our proposed framework. We provide a thorough runtime analysis of our algorithm and all the core steps. As the LIDAR data is received at 10 Hz, our algorithm must run in under 100 ms to achieve real-time performance. The mean runtimes and standard deviations over the full set of KITTI city log sequences are provided in Table 3.3.

We find that most of the steps of the pipeline are fairly consistent in runtime with the exception of the iterative EM procedure. This can be attributed to scene variation, as we only perform the EM procedure for columns that have not been pruned by the background filter. If we were to perform the EM procedure for all columns in the scene, the runtime would increase by approximately 35 ms, putting us just beyond our allotted 100 ms for real-time performance without decreasing the number of iterations.

Nevertheless, we find that for over 99% of the input data, the total runtime is under 100 ms, achieving real-time performance.



(a) Flow in the frame of the platform vehicle

(b) Flow in the global world frame

Figure 3.12: An example of motion compensation for temporal scene flow. The flow estimate is rendered in red, overlaid on top of the point cloud in blue. These samples are from the same sensor data at the same time.

Step	**Runtime (ms)**	**Standard Deviation (ms)**
Occupancy Grid Generation	12.5	0.87
Background Filter	1.2	0.10
Occupancy Constancy	41.4	0.83
Iterative Expectation-Maximization	28.5	5.53
Filtered Temporal Flow	0.4	0.43
Total	85.7	6.38

Table 3.3: Runtime performance of our proposed temporal scene flow method. We present both the total runtime and the runtime for each key step.

3.9 Discussion

We find that our estimate of scene flow performs quite well. Between 74.3% and 83.8% of the KITTI tracklets, depending on the class, produce raw measurements of scene flow that are within 30 cm of the ground truth flow, which is the resolution of our occupancy grid. For static background structure, even though our background filter is conservative, we are able to extract scene flow accurately.

It is interesting to compare our results for raw scene flow with Dewan et al. (2016). Like our work, they evaluate their method on the KITTI dataset, though they only provide results for translation error for sequences with no moving objects. Although our method has some differences in formulation, most notably the use of an occupancy grid as opposed to directly dealing with the points, the results are comparable. However, we do not rely on point correspondences or any data association between scans.

Unsurprisingly, our results are even more impressive when filtered over time. Our temporal model of scene flow allows us to quickly improve the accuracy of the estimate over time by both rejecting outliers and filtering measurements. This quickly achieves accurate sub-voxel level estimates of temporal scene flow.

While our algorithm is not an obstacle tracker, it is interesting to compare the accuracy of our temporal scene flow estimate to the performance of full fledged obstacle tracking algorithms. State of the art obstacle trackers such as Ushani et al. (2015) and Held et al. (2014) report average velocity errors of between 0.314 m/s and 0.56 m/s, respectively. Despite the fact that we do not make similar assumptions as these trackers, such as accurate segmentation, our error metrics are close to these marks. Additionally, we are not prone to errors due to not correctly modeling free space as shown in Ushani et al. (2015). However, these obstacle trackers are able to provide refined trajectory estimates and crisp point cloud models of tracked obstacles, something our proposed method does not aim to achieve.

It is important to note that the results we have presented for temporal scene flow are for individual flow tracklets. A dynamic object in the scene, such as a car, nominally induces several flow tracklets moving together. If we were to assume accurate segmentation, we could take cluster groups of flow tracklets together and combine their temporal flow estimates to produce an even more accurate result.

One key feature of our proposed method is the extendibility to other sensor modalities or environments. Occupancy grids can easily handle measurements from sensors other than LIDAR, such as radar or camera systems. Adapting to different environments or sensor configurations can be handled by simply retraining the background filter and occupancy constancy metric with new data. This is unlike many other approaches for autonomous vehicle applications that have a heavy reliance on prior maps, such as Ushani et al. (2015).

Unlike many other methods which estimate scene flow, our timing results demonstrate the real-time capability of our algorithm. Our method is capable of consuming LIDAR data at 10 Hz, which is the nominal rate for such sensors, and thus is appropriate for online use with autonomous systems.

3.10 Conclusion

We have presented an end-to-end pipeline for consuming LIDAR data and producing estimates of temporal scene flow. We have demonstrated the performance of this algorithm on the KITTI dataset, presenting results that are competitive with or better than the current state of the art. We have shown that this algorithm can be run at 10 Hz, enabling real-time use for mobile robotic applications.

CHAPTER 4

Feature Learning for Scene Flow Estimation

To perform tasks in dynamic environments, many mobile robots must estimate the dynamic motion in the surrounding world. Recently, techniques have been developed to estimate scene flow directly from LIDAR scans, relying on hand-designed features. In this chapter, we instead build an encoding network to learn features from an occupancy grid. The network is trained so that these features are discriminative in finding matching or non-matching locations between successive timesteps. This learned feature space is then leveraged to estimate scene flow. We evaluate our method on the KITTI dataset and demonstrate performance that improves upon the current state-of-the-art. This work was published in Ushani and Eustice (2018). Source code for this work is available at https://github.com/aushani/flsf.

4.1 Introduction

Mobile robots often operate in environments that are inherently dynamic. To operate safely, it is necessary for such systems to be aware of the motion in the world around them. For example, self driving cars need to be aware of other cars on the road, and warehouse robots must be able to move in an area with many other agents. Estimating dynamic motion is a core competency for these autonomous systems.

Many approaches for detecting motion perform object tracking in a three step process (Feldman, Hybinette, and Balch, 2012; Held et al., 2014; Kaestner et al., 2012; Petrovskaya and Thrun, 2009; Ushani et al., 2015; Vu and Aycard, 2009). First, sensor data, such as a camera image or a light detection and ranging (LIDAR) point cloud, is segmented into distinct objects. Then, data association is performed across timesteps. From the location of an object at each timestep, its motion can be estimated. Finally, this estimate is used in a filtering or smoothing framework.

Another approach for detecting dynamic motion from a stream of sensor data is to solve for optical flow or scene flow (Behl et al., 2017; Horn and Schunck, 1981; Jaimez et al., 2015;

Lucas and Kanade, 1981; Menze and Geiger, 2015; Taniai, Sinha, and Sato, 2017; Vogel, Schindler, and Roth, 2013, 2015). Traditionally computed from camera data, optical flow and scene flow approaches seek to find a motion field between two successive images by leveraging some sort of consistency metric (such as brightness constancy) in an optimization problem.

More recently, similar ideas have been applied to point cloud data (Dewan et al., 2016; Ushani et al., 2017). However, these typically are reliant on a hand-designed feature or metric, such as SHOT features (Dewan et al., 2016) or occupancy constancy (Ushani et al., 2017). If a more discriminative feature space can be learned, leveraging it could lead to an improved estimate of the dynamic motion.

In this work, we propose an encoding network that learns features from an input occupancy grid. The network is trained so that features from corresponding locations across different timesteps will be similar, and features from different locations will be dissimilar. Thus, we can leverage the learned feature space to estimate scene flow. By evaluating our proposed method on the KITTI dataset (Geiger et al., 2013), we demonstrate that our approach yields results that beat the current state-of-the-art in the accuracy of the estimated scene flow.

4.2 Related Work

The work presented here is at the intersection of research done in the areas of dynamic object tracking, optical flow and scene flow, and feature learning. It is also related to work in object reconstruction and scene completion.

4.2.1 Dynamic Object Tracking

Dynamic object tracking from LIDAR sensors is a well studied problem. Early approaches would use a three-phased approached (Azim and Aycard, 2012; Kaestner et al., 2012; Leonard et al., 2008; Streller, Furstenberg, and Dietmayer, 2002; Wender and Dietmayer, 2008; Zhao and Thorpe, 1998), First, a LIDAR scan would be segmented into discrete objects using some sort of clustering or detection algorithm. Then, these segments would be associated through time, commonly using global nearest neighbor (GNN) or a variant thereof. Finally, a Bayesian filtering framework, such as a extended Kalman filter (EKF), Unscented Kalman filter (UKF), or particle would be used.

More recently, model-based methods were developed to improve the tracking result. The models ranged from bounding boxes (Darms, Rybski, and Urmson, 2008; Kaestner et al., 2012; Petrovskaya and Thrun, 2009; Vu and Aycard, 2009) to more expressive models such as ellipses or star-convex shapes (Baum and Hanebeck, 2014).

Later on, approaches would improve upon the motion estimate between successive timesteps. Feldman, Hybinette, and Balch (2012) aligned segmented snapshots of objects between scans using iterative closest point (ICP). Held et al. (2014) proposed annealing histograms to perform a bounded search to find the best relative motion that optimizes a measurement model. Ushani et al. (2015) used a continuous stream of the LIDAR observations from an object in a smoothing framework to estimate its trajectory.

Despite the impressive results achieved by these works, they are prone to catastrophic failure if there is an error in object segmentation or data association through time. Some methods were developed that did not rely on explicit segmentation or data association. Tanzmeister et al. (2014) and Danescu, Oniga, and Nedevschi (2011) propose grid based tracking systems, where particles move among cells in a grid and are updated according to the observations. Dequaire et al. (2017) propose a recurrent neural network that predicts a future occupancy grid from LIDAR input.

4.2.2 Optical Flow and Scene Flow

In optical flow, 2D motion in the image plane is solved for using successive sensor observations, such as images from a camera sensor (Horn and Schunck, 1981; Lucas and Kanade, 1981). Typically, a constancy metric that enforces consistency between images (e.g., brightness constancy) is leveraged. With three-dimensional (3D) information, such as from a depth sensor or a stereocamera, 3D motion can be estimated, known as scene flow. Many methods exist to estimate scene flow from camera data. Some methods leverage an initial oversegmentation followed by a CRF (Menze and Geiger, 2015) or an energy minization framework (Vogel, Schindler, and Roth, 2013, 2015). Jaimez et al. (2015) rely on a primal-dual algorithm. Taniai, Sinha, and Sato (2017) identify regions that are inconsistent with the ego-motion of the platform and then build an energy minimization problem consisting of appearance, flow, prior, color, and smoothing terms. Behl et al. (2017) leverage object recognition in a conditional random field (CRF) model.

However, estimating scene flow from camera images is typically very slow. In the KITTI Scene Flow Challenge, the current top nine submissions take longer than 5 minutes to run, with three reporting runtimes of nearly an hour (Geiger et al., 2013).

Recently, similar techniques have been applied to LIDAR point clouds. Dewan et al. (2016) use an energy minimization problem to estimate rigid scene flow between LIDAR scans that leverages SHOT feature descriptors. Ushani et al. (2017) also frame an energy minimization problem, but instead leverage “occupancy constancy”, measuring the consistency of the occupancy states between successive occupancy grids built from the LIDAR point clouds.

While these methods are similar to ours, we seek to find a learned feature representation rather than relying on a hand-designed feature.

4.2.3 Object and Scene Understanding

Many related tasks in object and scene understanding must similarly rely on building a feature representation. These tasks include object reconstruction, inpainting, and scene completion. These types of problems have been extensively researched in the literature in the context of both two-dimensional (2D) images and 3D objects.

Many approaches, especially in recent years, have relied on the use of an autoencoder (Hinton and Salakhutdinov, 2006; Vincent et al., 2010), Generative Adversarial Network (GAN) (Goodfellow et al., 2014), or similar methods. They all usually rely on building a representation of the object or image that is being considered in some kind of embedded or latent space, similar to our method. This representation is then leveraged for the desired task.

For 3D objects, Wu et al. (2015) were one of the first to build 3D deep learning models that can be leveraged for recognition, reconstruction, or classification of 3D objects. They design a convolutional deep belief network (CDBN) that models the relationship and dependencies in a 3D voxel grid and object labels, similar in spirit to the early work of Hinton, Osindero, and Teh (2006).

Girdhar et al. (2016) train an autoencoder for 3D object reconstruction. In addition to voxel-wise cross entropy loss, they also train a network to map from image data to the same latent representation and use a Euclidean loss between the two. They show their learned latent representation, with some feature augmentation, is somewhat class-discriminative, despite not explicitly being trained to be so.

Guizilini and Ramos (2017) consider 3D reconstruction using autoencoders in the context of building large scale maps from LIDAR. Their autoencoder learns to reconstruct shape primitives found in the world. However, they do not consider distinct objects, but rather segments of the map that they cluster to estimate the occupancy state of unknown space.

Wu et al. (2016) leverage a GAN for object reconstruction, generation, and classification. They report better results when they train a separate GAN for each semantic object class. This method is improved upon by Smith and Meger (2017), but the results are still better when separately trained.

Pathak et al. (2016) use an autoencoder to recover missing portions of an image by semantic inpainting. They use a combination of L2 pixel-wise reconstruction loss and an adversarial loss over the whole reconstructed image to ensure it appears realistic after inpainting.

Wang et al. (2017) consider shape inpainting by using an autoencoder as the generator in a GAN. The network is trained on a combination of voxel-wise reconstruction loss and the GAN objective function.

Yeh et al. (2017) consider semantic inpainting of face images using a GAN. A latent representation is found using the available image data. Then, this latent representation is used with the generator from the GAN to recreate the full image. Qualitative results show an improvement of this method as compared to autoencoder approaches.

Dai, Qi, and Nießner (2017) use an autoencoder to perform shape completion. The latent representation is augmented by the output of a separate independently trained object classification network (Qi et al., 2016). The output of the autoencoder is fine-tuned by finding nearest neighbors in an object database to build the final reconstruction.

Choy et al. (2016) build a network that is an encoder followed by a long short-term memory (LSTM) followed by a decoder. This network takes images as input and produces 3D volumetric grids and is trained without relying on semantic object class labels. Fan, Su, and Guibas (2017) use a similar approach to instead generate point clouds. They explore how different loss functions capture shape properties differently. Lin, Kong, and Lucey (2018) also use an image encoder and structure decoder to predict 3D structure of objects from one or more images to build a dense point cloud. They train their network with both a single object class at a time and multiple classes at once.

In a different application, Liu, Yu, and Funkhouser (2017) train a 3D GAN to aid users in 3D modeling. A loss function that combines realism (as measured by a discriminator) and semantic dissimilarity (using intermediate activations from a classifier) is used.

The task of scene completion becomes more difficult when dealing when a complex environment rather than a single distinct object. However, in certain applications with environments that exhibit repeated structure, a few methods have demonstrated success. Song et al. (2017) perform scene completion from a single depth image of an indoor environment using an end-to-end 3D convolutional neural network (CNN). Dai et al. (2018) use a similar approach for scene completion, but additionally predict semantic labels as well.

While different deep learning tools are used, these methods are all similar in that they leverage some kind of learned latent representation for the data to be considered. This is similar to our proposed method in which a feature representation is learned and then exploited.

4.2.4 Feature Learning

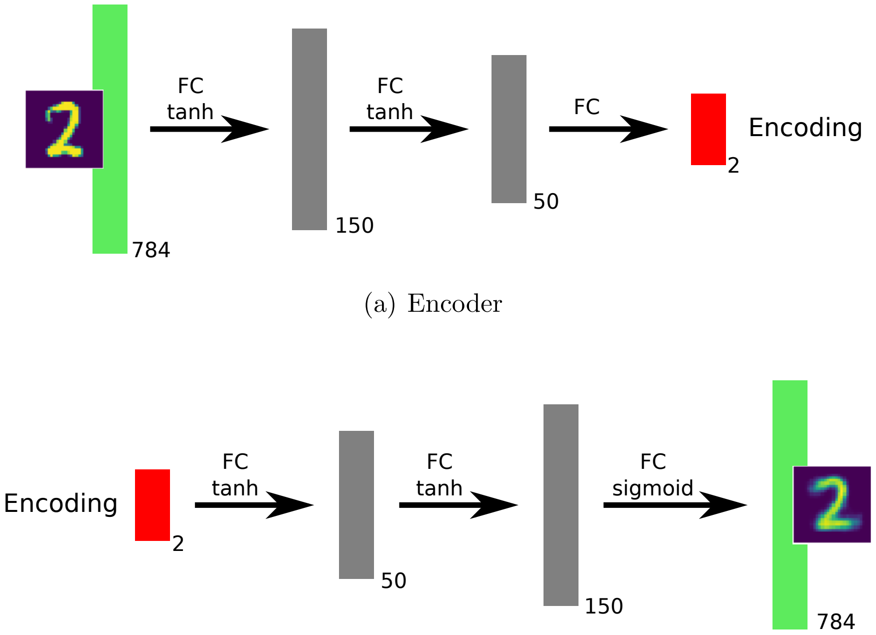
Feature learning approaches have found success in many areas of robotics. This includes face recognition (Wen et al., 2016), long term image matching (Carlevaris-Bianco and Eustice, 2014), and point cloud segmentation and classification (Qi et al., 2017a,b). Generally, these approaches construct a network from the input data to the feature space, and use a loss function to promote the separability of the features for the desired task. For example, Wen et al. (2016) proposed center loss, where features from the same class are pulled towards the same center, and centers from different classes are forced to stay apart. Carlevaris-Bianco and Eustice (2014) used a loss function that increases as the Euclidean distance between matching feature pairs grows, and decreases as the Euclidean distance between non-matching feature pairs grows.

More recently, PointNet and PointNet++ are two deep learning techniques for point cloud feature representation (Qi et al., 2017a,b). Given an input point cloud, these methods find a feature representation that is invariant to the order of the points and invariant to a transformation, such as a rotation or translation, that is applied to all points. This feature representation is demonstrated for tasks including segmentation and classification.

More similar to our method, Zeng et al. (2017) present 3DMatch. Patches are extracted from red, green, blue, and depth (RGB-D) reconstructions. Correspondences are collected from different views. A CNN is then trained to learn a geometric descriptor that outperforms existing methods in determining correspondences. However, unlike our work, the dynamics of the scene are not considered.

4.3 Motivation and Toy Example

There are several choices we must make to choose how to learn our feature representation. For example, we can chose an autoencoder or GAN approach to find low-dimensional feature representations that can adequately represent our high-dimensional input data, similar to approaches for object or scene understanding. However, we choose to find a encoded feature using more of a feature learning approach for two reasons. First, while autoencoder and GAN approaches have found much success for object reconstruction, the applications of these method for more complex scenes has proved to be more challenging, with a few methods showing success in environments with much repeated structure (Dai et al., 2018; Song et al., 2017). Secondly, we find that a feature learning approach that is specifically designed for our task allows us to learn a more useful feature representation. This feature space is more discriminative for determining whether or not two samples (such as locations in an occupancy



(a) Encoder

(b) Decoder

Figure 4.1: The encoding and decoding networks used for the MNIST toy example. In Fig. 4.1(a), we see the encoding network. An image is flattened to a 784-vector, which is then passed through two fully connected layers with tanh activation. This is then passed through a final fully connected layer to produce the encoded representation of the original image. In Fig. 4.1(b), we see the decoding network. This is similar to the encoding network, but in reverse. The encoding is passed through two fully connected layers with tanh activation. This is finally passed through a fully connected layer with sigmoid activation and reshaped to produce the reconstructed image.

grid) match.

We motivate this choice by applying a toy example problem using the MNIST dataset (LeCun et al., 1998). This fairly simple dataset allows us to build some intuition for how to best learn the feature space we are interested in.

4.3.1 Network

We build a simple encoding network for MNIST images. The network is shown in Fig. 4.1(a). The encoder takes as input a flattened image as a 784 dimensional vector. To encode, this is passed through two fully connected layers with tanh activation and then an additional fully connected layer, yielding a 2D feature encoding.

We train this encoding network in two ways. First, we use the standard autoencoder approach. We construct a decoding network, as shown in Fig. 4.1(b). This network takes this feature encoding and treats it as a latent representation, Through two fully connected layers with tanh activation and a final fully connected layer with sigmoid activation, the original image is reconstructed. The entire network is trained with the standard reconstruction loss based on mean squared error:

$$\mathcal{L}_r = \frac{1}{n} \left\| \hat{x} - x \right\|_2^2 , \tag{4.1}$$

where $\hat{x}$ is the input image and x is the reconstruction.

For our second approach, we use a loss function based directly on the learned feature space. We promote features for images of the same number to be similar, and features for images of different numbers to be dissimilar. Thus, we have:

$$L_{fl}\big(\mathbf{f}_1, \mathbf{f}_2\big) = \begin{cases} \|\mathbf{f}_1 - \mathbf{f}_2\| & x_1 \text{ and } x_2 \text{ are images of the same number} \\ d_{max} - \|\mathbf{f}_1 - \mathbf{f}_2\| & x_1 \text{ and } x_2 \text{ are images of different numbers,} \\ & \qquad \|\mathbf{f}_1 - \mathbf{f}_2\| < d_{max} \\ 0 & \text{otherwise} \end{cases} , \tag{4.2}$$

where x_1 and x_2 are the input images, and $\mathbf{f}_1$ and $\mathbf{f}_2$ are their feature representations, respectively. For this experiment, we use $d_{max} = 5$.

We train both networks using standard L2 regularization. Additionally, for each encoding that we learn, we use the feature representation in a simple, one-layer classifier. When we train this classifier, the encoding network is held constant.

We recognize that the dataset and these networks are fairly simple. However, they serve to motivate our approach and illustrate the reasoning behind our choices.

4.3.2 Motivating Results

h First, we show the learned feature representations in Fig. 4.2 and Fig. 4.3. We see that the feature learning approach produces a feature space that is much more discriminative. The traditional autoencoder approach produces a latent representation that, while somewhat separable, has much overlap between different classes.

Next, we further demonstrate this by evaluating the simple, one-layer classifier for each learned feature space. Classification results are shown in Table 4.1. We find a improvement in the classification of all classes and a significant improvement overall.

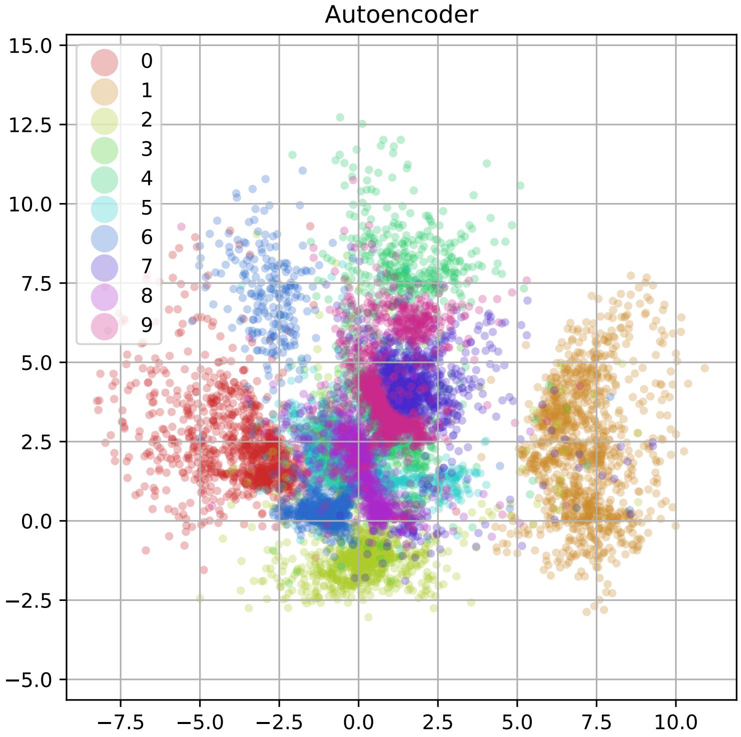


Figure 4.2: Learned latent encoding for MNIST using the autoencoder approach. Points are colored by class.

4.4 Method

In this section, we describe our proposed method. We will make similar assumptions as other works in this area. Chiefly, as we target autonomous vehicle applications with the KITTI dataset in this work, we will assume that all dynamic motion is in the horizontal plane and that the motion field is consistent for everything at the same (x, y) location. We seek to estimate the motion (u, v) for every 2D location $\mathbf{x} = (x, y)$ from one timestep to the next. That is to say, for a location $\mathbf{x}_t$ at time t, we seek to find the corresponding location at the next timestep, $\mathbf{x}_{t+1} = \mathbf{x}_t + (u, v)$ such that whatever was at $\mathbf{x}_t$ at time t is now at $\mathbf{x}_{t+1}$ at time $t + 1$. We term these $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ to be matching locations (and non-matching locations otherwise).

Furthermore, we assume that the motion corresponds to locally rigid, non-deforming flow. Finally, since much of the environment is background structure that is static, we focus our attention on the dynamic objects in the scene and assume that the flow of static objects can

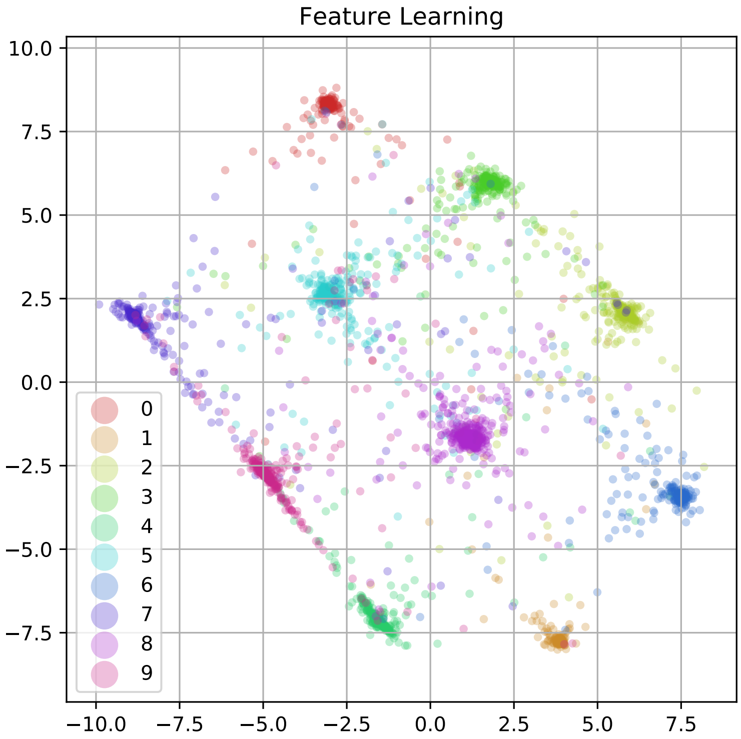


Figure 4.3: Learned feature representations for MNIST using the feature learning approach. Points are colored by class.

be accurately estimated from odometry.

4.4.1 Input

Our system takes as input a point cloud $\mathrm{P}_t = \{\mathbf{z}_{t,1:n}\}$. We seek to construct an occupancy grid G_t that maps each (x, y, z) location to a probability that the given location is occupied (Hornung et al., 2013; Moravec and Elfes, 1985).

For each voxel $\mathbf{v}$ at location (x, y, z), the occupancy probability given the set of observations P_t is given by:

$$p(\mathbf{v}|\mathbf{z}_{t,1:n}) = \left[1 + \frac{1 - p(\mathbf{v}|\mathbf{z}_{t,n})}{p(\mathbf{v}|\mathbf{z}_{t,n})} \frac{1 - p(\mathbf{v}|\mathbf{z}_{t,1:n-1})}{p(\mathbf{v}|\mathbf{z}_{t,1:n-1})} \frac{p(\mathbf{v})}{1 - p(\mathbf{v})}\right]^{-1}. \tag{4.3}$$

We assume an uninformative prior $p(\mathbf{v}) = 0.5$. Using log-odds notation, denoted by $LO(\,\cdot\,)$,

Table 4.1: MNIST classification results. We show results for each class for both approaches.

	Autoencoder Approach	**Feature Learning Approach**
0	87.7 %	98.2 %
1	97.7 %	98.6 %
2	80.8 %	95.8 %
3	74.1 %	97.0 %
4	39.3 %	96.5 %
5	1.9 %	94.8 %
6	52.8 %	96.7 %
7	58.3 %	95.1 %
8	47.3 %	95.3 %
9	40.7 %	94.9 %
Overall	59.3 %	96.3 %

we can rewrite (4.3) as

$$LO(\mathbf{v}|\mathrm{P}_t) = \sum_{i=1}^{n} L(\mathbf{v}|\mathbf{z}_{t,i}), \tag{4.4}$$

where

$$LO(\mathbf{v}|\mathbf{z}_{t,i}) = \begin{cases} l_{\text{free}} & \text{the ray from the sensor to } \mathbf{z}_{t,i} \text{ passes through } \mathbf{v} \\ l_{\text{occupied}} & \text{the ray from the sensor to } \mathbf{z}_{t,i} \text{ ends inside } \mathbf{v} \\ 0 & \text{otherwise} \end{cases} . \tag{4.5}$$

To compute G_t, this ray tracing is achieved using Bresenham's ray tracing algorithm (Bresenham, 1965). Each observation $\mathbf{z}_{t,i}$ results in a ray tracing operation to find its log-odds updates for the corresponding voxels. After processing all observations, these updates are summed for all voxels to produce G_t. This algorithm is implemented on the GPU for efficient computation, as discussed in Appendix B. A sparse representation of the occupancy grid is created, which is then used to build a dense 3D occupancy grid using a custom Compute Unified Device Architecture (CUDA) kernel.

4.4.2 Network

The probabilities of the constructed occupancy grid G_t are first rescaled between -0.5 and 0.5 to produce $\tilde{\mathrm{G}}_t$. Note that any unknown voxels (i.e., $p(\mathbf{v}) = 0.5$) are scaled to 0 in $\tilde{\mathrm{G}}_t$. $\tilde{\mathrm{G}}_t$ is passed through a series of 2D convolution layers with leaky RELU activations, with kernel sizes and output layers as depicted in Fig. 4.4. This network has two outputs. First, we have an encoding output, F_t, that maps each (x, y) location to a N_f dimensional feature vector $\mathbf{f}_{(x,y)}$. Second, we have a classification output, C_t, that yields softmax scores for the foreground/background classifier for each location (x, y). Notably, we intentionally use only a single layer between the encoding output and the filter output to help promote learning a discriminative feature space.

The loss function for the encoding output is as follows. For locations $\mathbf{x}_1$ and $\mathbf{x}_2$ with feature vectors $\mathbf{f}_1$ and $\mathbf{f}_2$, respectively, the loss is given by

$$L_{\mathbf{f}_1,\mathbf{f}_2} = \begin{cases} \|\mathbf{f}_1 - \mathbf{f}_2\| & \mathbf{x}_1 \text{ and } \mathbf{x}_2 \text{ are matching locations} \\ d_{max} - \|\mathbf{f}_1 - \mathbf{f}_2\| & \mathbf{x}_1 \text{ and } \mathbf{x}_2 \text{ are non-matching locations,} \\ & \qquad \|\mathbf{f}_1 - \mathbf{f}_2\| < d_{max} \\ 0 & \text{otherwise} \end{cases}, \tag{4.6}$$

where $\|\cdot\|$ denotes the L2 norm. We choose $d_{max} = 10$. This loss function is visualized in Fig. 4.5.

The loss function for the classification output, L_{C}, is simply the mean weighted softmax cross entropy loss. As the training data contains far more instances of background than foreground, we weight the loss of each class by the inverse of their respective frequency. Thus, we have:

$$L_{\mathrm{C}} = \frac{1}{n_{\text{samples}}} \left(\frac{1}{f_{\text{fg}}} \sum_{\mathbf{x} \in \text{foreground}} L_{\text{S.C.E.}}(\mathrm{C}_{t,\mathbf{x}}) + \frac{1}{f_{\text{bg}}} \sum_{\mathbf{x} \in \text{background}} L_{\text{S.C.E.}}(\mathrm{C}_{t,\mathbf{x}}) \right) \tag{4.7}$$

where n_{samples} is the number of labeled foreground or background locations in the training sample, f_{fg} and f_{bg} are the frequencies of foreground and background locations respectively in the training set, and $L_{\text{S.C.E.}}$ is the softmax cross entropy loss.

These loss functions are combined to form the total loss function,

$$L_{total} = L_{\mathbf{f}_1,\mathbf{f}_2} + L_{\mathrm{C}}, \tag{4.8}$$

which is used to train the network.

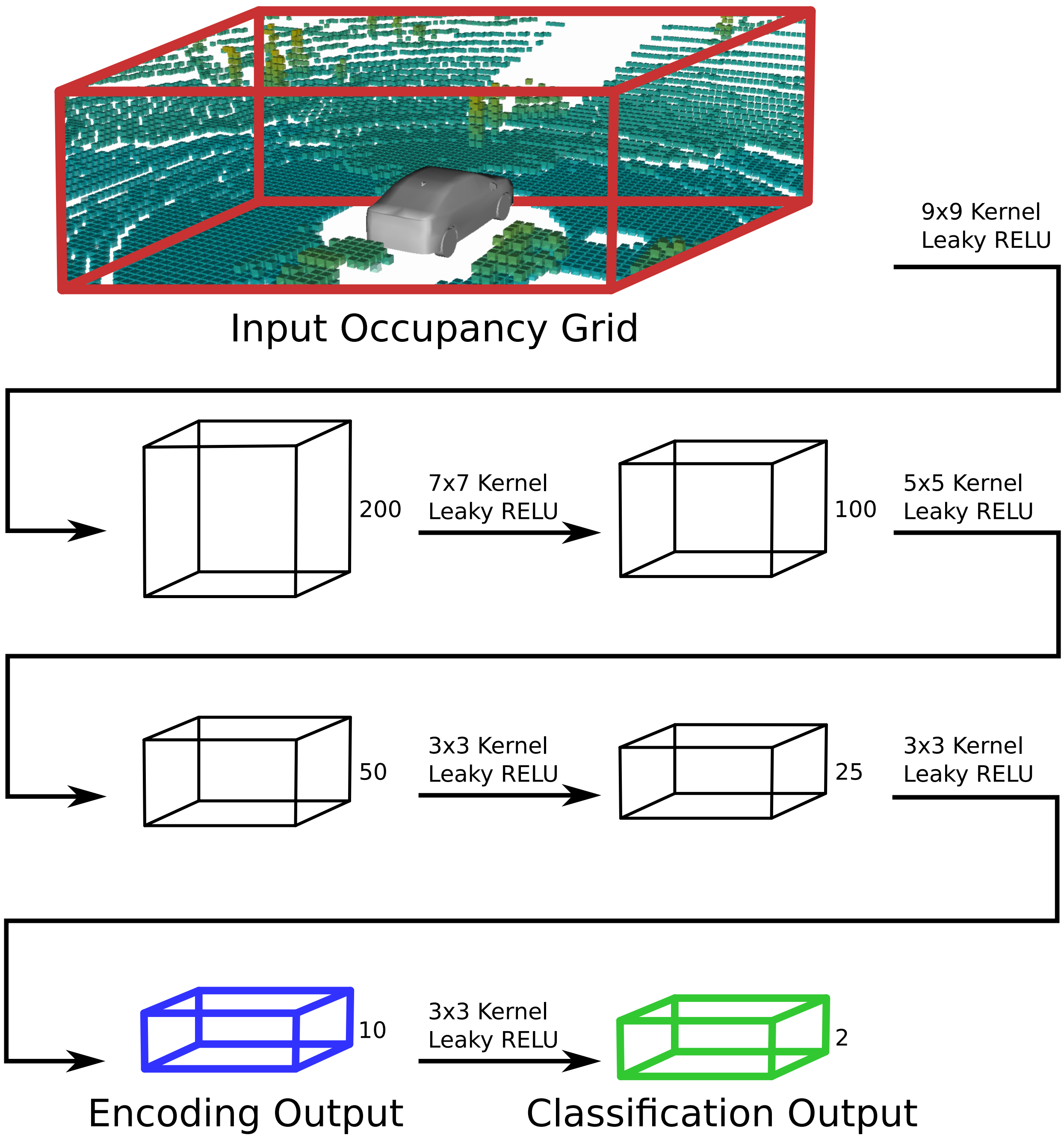


Figure 4.4: The encoder network architecture. The input occupancy grid is passed through a series of convolution layers to produce a grid of feature vectors. This encoding is further passed through a single convolutional layer to produce softmax scores for the foreground/background classifier.

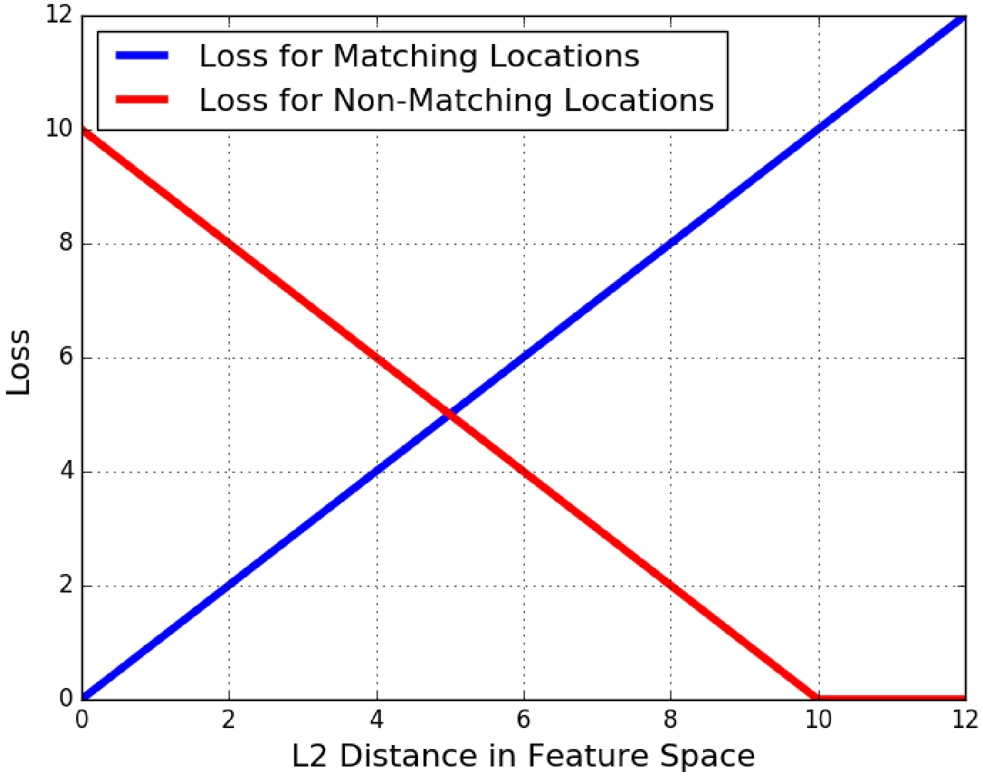


Figure 4.5: The loss function $L_{\mathbf{f}_1,\mathbf{f}_2}$ for matching and non-matching locations.

4.4.3 Training

At each iteration, we train the network using one sample of locations $\mathbf{x}_1$ and $\mathbf{x}_2$ (which could be matching or non-matching) and one sample of a classification map C_t, simultaneously.

4.4.3.1 Training Data

We construct a training data set using the first ten KITTI city sequences. We step through each log and construct an occupancy grid from each velodyne point cloud, as described in Section 4.4.1.

Using the labeled KITTI tracklet data, we construct a 2D classification map for ground truth foreground/background classification. For any location $\mathbf{x}$ that is within the bounding box of a object labeled in the KITTI tracklets, we mark the location as foreground. Otherwise, we mark the location as background.

For the encoding output, we take successive occupancy grids G_1 and G_2. We first iterate through all foreground locations $\mathbf{x}_1$ from G_1 that are part of a labeled KITTI tracklet (i.e., the foreground). We then sample from a location $\mathbf{x}_2$ from G_2, chosen from an $n_s \times n_s$ neighborhood of locations centered on $\mathbf{x}_1$. Using the labeled KITTI tracklet data, we record whether $\mathbf{x}_1$ and $\mathbf{x}_2$ are matching or non-matching locations in the successive occupancy grids. $\mathbf{x}_1$ and $\mathbf{x}_2$ are sampled such that we have approximately an equal number of matches and non-matches. Additionally, we repeat this procedure for 1% of all background locations.

Note that the labeled KITTI tracklets are only valid in the field of view of the camera. Accordingly, we take care as to only include such locations in our training data set. To help mitigate this, we augment our data with random rotations and reflections of the occupancy grids.

4.4.3.2 Training Procedure

We use TensorFlow's AdamOptimizer with a learning rate of 10^{-6} (Abadi et al., 2015) to train the network using the loss function described in (4.8). During training, we use dropout at each layer in the encoding network with a dropout probability of 20%. Note that at the input, due to the rescaling of the occupancy grid, dropout essentially amounts to setting voxels from free or occupied (i.e., $p(\mathbf{v}) < 0$ or $p(\mathbf{v}) > 0$, respectively) to unknown (i.e., $p(\mathbf{v}) = 0$). These parameters were empirically tuned.

We trained the network for approximately two days using a NVIDIA GeForce GTX TITAN X GPU.

4.4.4 Flow Computation

We take two successive point clouds, P_1 and P_2. From these, we construct occupancy grids, G_1 and G_2. Each is rescaled and passed through the network described in Section 4.4.2 using NVIDIA's cudNN library to produce encoding ouptuts F_1 and F_2 and classification outputs C_1 and C_2. Note that as we deal with a stream of data, we can cache P_2, G_2, F_2, and C_2 for use at the next timestep for faster runtime performance.

For each location $\mathbf{x}_1$ from G_1, we consider the feature vector $\mathbf{f}_1$. We then consider a $n_s \times n_s$ window of locations $\mathbf{x}_2$ from G_2 around $\mathbf{x}_1$. For each $\mathbf{x}_2$ and $\mathbf{f}_2$, we compute the L2 norm between the two feature vectors and store this result in a lookup table,

$$T_{\text{distance}}(\mathbf{x}_1, \mathbf{x}_2) = \left\|\mathbf{f}_1 - \mathbf{f}_2\right\|. \tag{4.9}$$

Note that T_{distance} is similar to T_{match} from Chapter 3. The key difference is that T_{match} is rooted in a hand-designed notion of occupancy constancy, whereas T_{distance} takes a fully learned approach to measure distances between feature vectors in our learned feature space.

From this point, we compute scene flow by using an iterative expectation-maximization (EM) algorithm similar to the one proposed in Chapter 3, which we briefly summarize here.

We construct an energy minimization problem to compute the flow (u, v) for every location $\mathbf{x}_1$ from G_1. If, according to the classification output C_1, the given location is more likely to be static background structure, we flag $\mathbf{x}$ as such and assume that the flow can be estimated from odometry sensors that are measuring the relative motion of the platform. Thus, we focus our attention on foreground locations $\mathbf{x}$ that are dynamic (or could be dynamic).

We construct an iterative EM algorithm to estimate the flow. We use $\mathbf{x}_1$ to denote a location in G_1 and $\mathbf{x}_2$ for locations in G_2. $s(\mathbf{x}_1) = (u, v)$ denotes the current estimate of the scene flow at location $\mathbf{x}_1$. $m(\mathbf{x}_1) = \mathbf{x}_1 + s(\mathbf{x}_1)$ denotes the estimated matching location $\mathbf{x}_2$ in G_2 for $\mathbf{x}_1$. At the start of the algorithm, all $s(\mathbf{x}_1)$ and $m(\mathbf{x}_1)$ are marked as being invalid.

$E(\mathbf{x}_1, \mathbf{x}_2)$ denotes the energy of the flow from $\mathbf{x}_1$ to $\mathbf{x}_2$. $\hat{E}(\mathbf{x}_2)$ denotes the energy of the current flow estimate that leads to $\mathbf{x}_2$, initialized with ∞.

4.4.4.1 Expectation

During the expectation step, we seek the most likely flow (u, v) for every $\mathbf{x}_1$. We search through a $n_s \times n_s$ window of locations centered on $\mathbf{x}_1$, $N_{\mathbf{x}_1}$, in G_2. For each $\mathbf{x}_1$, $\mathbf{x}_2$, we compute an energy,

$$E(\mathbf{x}_1, \mathbf{x}_2) = T_{\text{distance}} + w_p \sum_{\mathbf{x} \in P_{\mathbf{x}_1}} \left\| (\mathbf{x}_2 - \mathbf{x}_1) - s(\mathbf{x}) \right\|^2, \tag{4.10}$$

where w_p is a smoothness penalty weight and $P_{\mathbf{x}_1}$ is a 5×5 window around $\mathbf{x}_1$ of locations for which we currently have a valid estimate for scene flow.

We seek:

$$\mathbf{x}_2^\star = \operatorname*{argmin}_{\mathbf{x}_2} E(\mathbf{x}_1, \mathbf{x}_2), \tag{4.11}$$

subject to the conditions:

$$E(\mathbf{x}_1, \mathbf{x}_2) < \hat{E}(\mathbf{x}_2) \tag{4.12}$$

$$\text{or}$$

$$m(\mathbf{x}_1) = \mathbf{x}_2. \tag{4.13}$$

We then update $s(\mathbf{x}_1)$ and $m(\mathbf{x}_1)$ with $\mathbf{x}_2^\star - \mathbf{x}_1$ and $\mathbf{x}_2^\star$, respectively. If no such $\mathbf{x}_2^\star$ is found, the estimated flow is marked as being invalid.

4.4.4.2 Maximization

For each $\mathbf{x}_2$, we consider all $\mathbf{x}_1$ with $m(\mathbf{x}_1) = \mathbf{x}_2$. If there is at least one, we find

$$\mathbf{x}_1^\star = \underset{\mathbf{x}_1}{\operatorname{argmin}}\, E(\mathbf{x}_1, \mathbf{x}_2), \tag{4.14}$$

in essence selecting the $\mathbf{x}_1$ that best matches $\mathbf{x}_2$. We update $E(\mathbf{x}_1) = E(\mathbf{x}_1^\star, \mathbf{x}_2)$. We also invalidate all $s(\mathbf{x}_1)$ and $m(\mathbf{x}_1)$ for $\mathbf{x}_1 \neq \mathbf{x}_1^\star$.

4.5 Results

We evaluate our proposed method in a number of experiments using the KITTI dataset (Geiger et al., 2013). For all results presented here, we build a training set from the first ten KITTI city sequences and a test set from the remaining sequences.

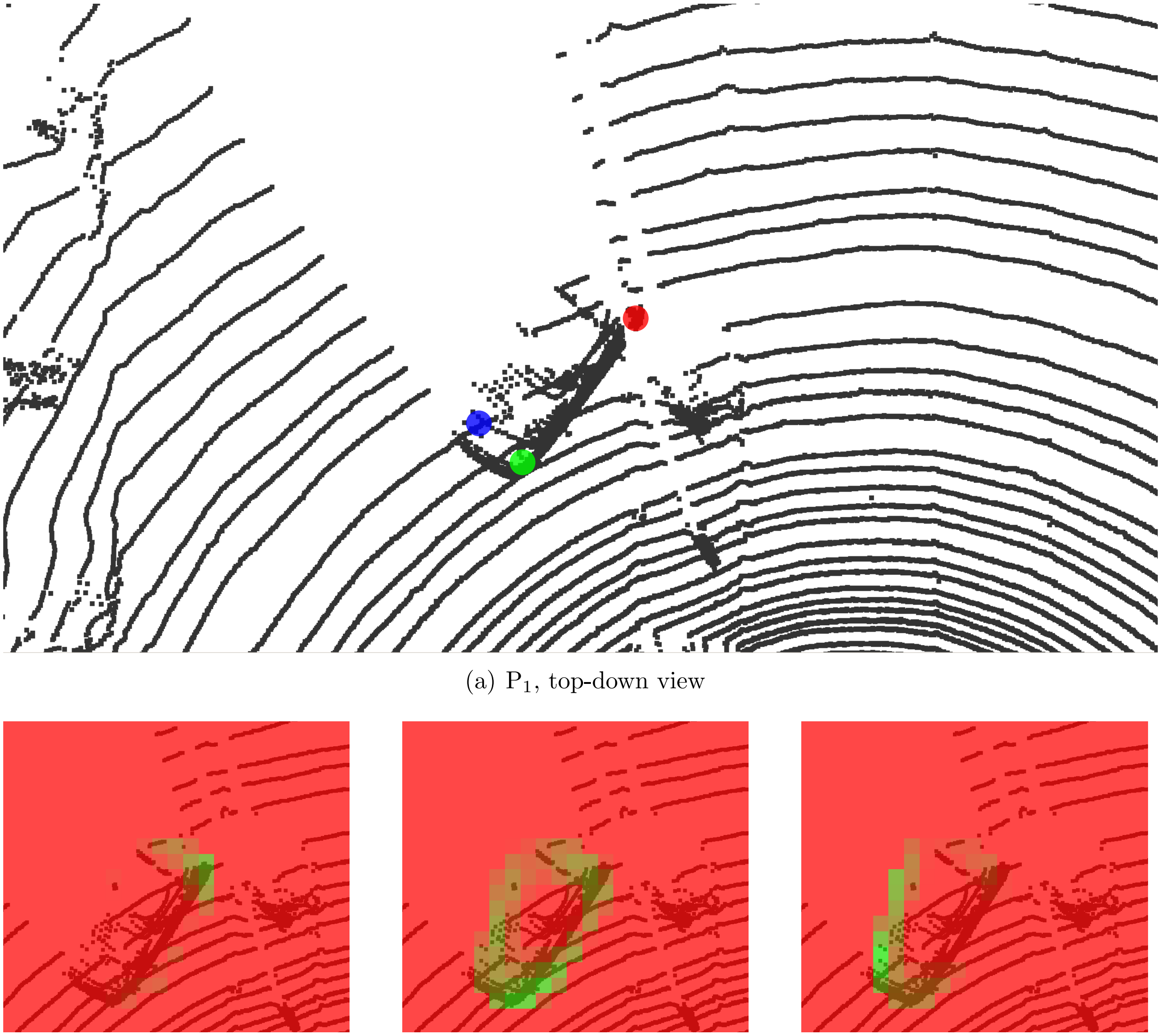
We used $n_s = 31$, 20 EM iterations, and an occupancy grid with a resolution of 30 cm that extends over a 50 m by 50 m area. We set our smoothing parameter $w_p = 0.07$ by inspecting a validation set constructed from the first ten KITTI city log sequences.

4.5.1 Visualizing the Learned Feature Space

We first present a qualitative evaluation of our learned feature space. In Fig. 4.6, we have a scene where a car is moving from behind some traffic poles towards the bottom of the image. We inspect the feature vector at three locations from P_1, indicated by the colored dots in Fig. 4.6(a) at various places on the cars. For each, we evaluate the distance in feature space, T_{distance}, to locations in the following scan P_2, shown in Fig. 4.6(b), Fig. 4.6(c), and Fig. 4.6(d).

We can see that our learned features are discriminative. Each chosen location show in Fig. 4.6(a) is closest in feature space to its matching location in the following timestep. Unsurprisingly, we see that similar locations sometimes yield similar feature vectors, such as different corners of the car in Fig. 4.6(c). However, even in this example, we see that the most similar feature vectors (i.e., the locations that are most green) are clustered around the matching location. Additionally, we find that feature vectors from the car, which are foreground locations, are a large distance from the background locations (shown in red, beyond range of the colormap).

We present a similar analysis of a different scene in Fig. 4.7. We focus on two pedestrians walking along the sidewalk. They can be seen in the camera image shown in Fig. 4.7(b). In Fig. 4.7(c) and Fig. 4.7(d), we see distances in feature space to locations at the following



(a) P_1, top-down view

(b) Red

(c) Green

(d) Blue

Figure 4.6: A visual look at the distances in the feature space for a turning car. For the three locations from P_1 in Fig. 4.6(a), we evaluate the distance in feature space, T_{distance}, to locations in the successive scan P_2. These are shown in Fig. 4.6(b) for the red point, Fig. 4.6(c) for the green point, and Fig. 4.6(d) for the blue point. Green indicates small distances and red indicates large distances. The colormap ranges from a distance of 0 to 2 in the feature space. Best viewed in color.

timestep for each pedestrian. Using a colormap with a large range, we see that the pedestrians' feature vectors are similar to their new position, but they are also somewhat similar to some of the background structure, such as a traffic sign. However, when we look at the distances using a tighter colormap in Fig. 4.7(f) and Fig. 4.7(e), we find that the closest locations in feature space are indeed those from the appropriate matching locations.

4.5.2 Learned Feature Space vs. Occupancy Constancy

In additional to the qualitative results in Section 4.5.1, we also perform a quantitative analysis to demonstrate the performance of our learned feature space. We compare our learned feature space with the occupancy constancy metric proposed in Chapter 3 in terms of how discriminative they are in determining matching or non-matching locations.

We build a simple binary classifier for each method that determines if two locations, $\mathbf{x}_1$ from G_t and $\mathbf{x}_2$ from G_{t+1}, are matching or non-matching. For our learned feature space, we take the distance in feature space, T_{distance}, and compare this against a threshold τ. If the distance is small, then our classifier predicts that $\mathbf{x}_1$ and $\mathbf{x}_2$ are matching. Otherwise, it predicts that they are non-matching. For occupancy constancy, we perform a similar procedure with the occupancy constancy metric T_{match} from Chapter 3.

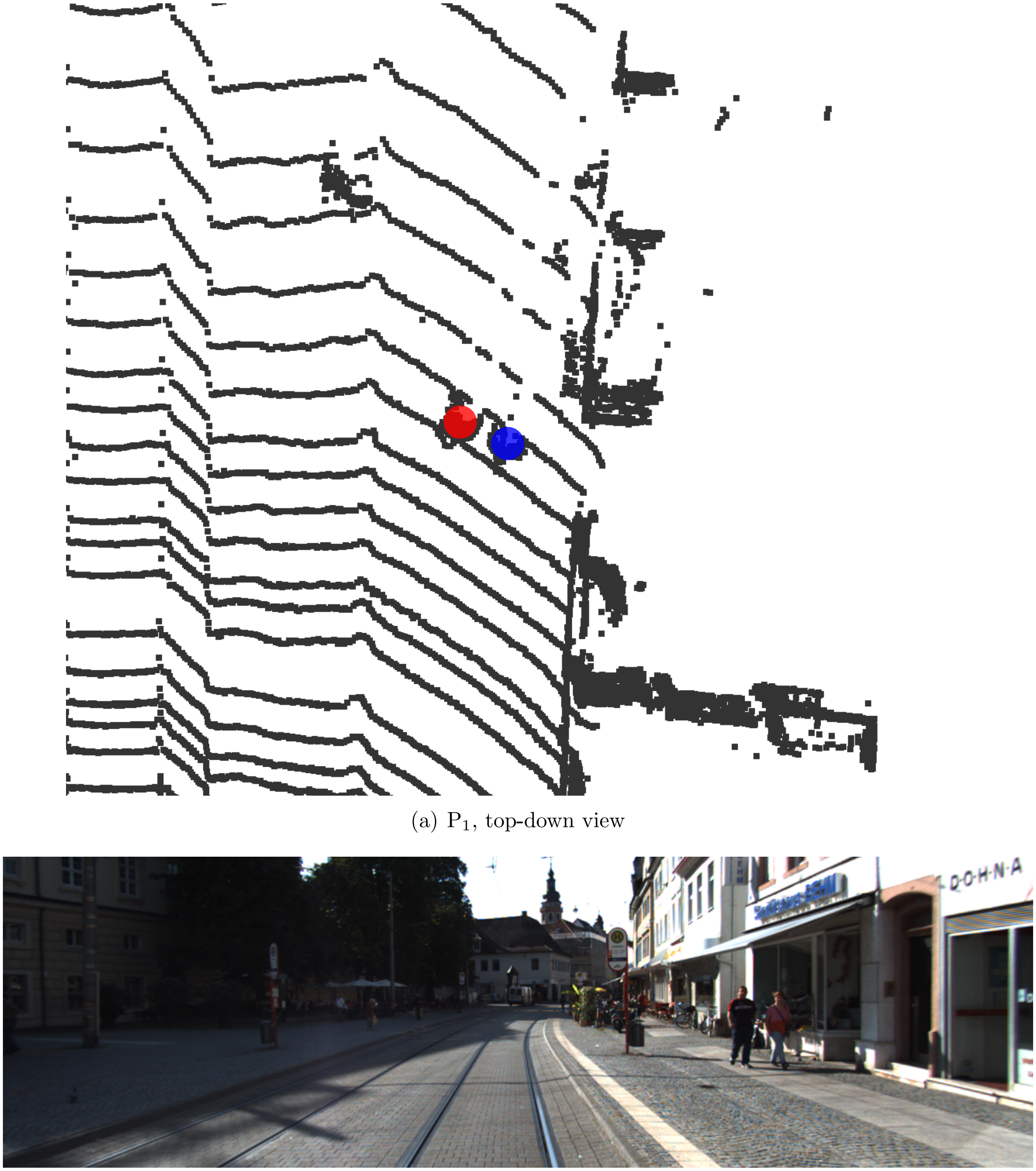
We sample 5000 locations $\mathbf{x}_1$ from the foreground of some occupancy grid G_t from the test set. For each $\mathbf{x}_1$, we sample another location $\mathbf{x}_2$ from the following occupancy grid G_{t+1}. We evenly sample $\mathbf{x}_2$ such that it is equally likely to match or not match $\mathbf{x}_1$.

We present precision recall curves for each described classifier in Fig. 4.8. As we can see, our learned feature space is significantly better at distinguishing between matching and non-matching locations than occupancy constancy, demonstrating that it is more discriminative for the task at hand.

4.5.3 Scene Flow Results

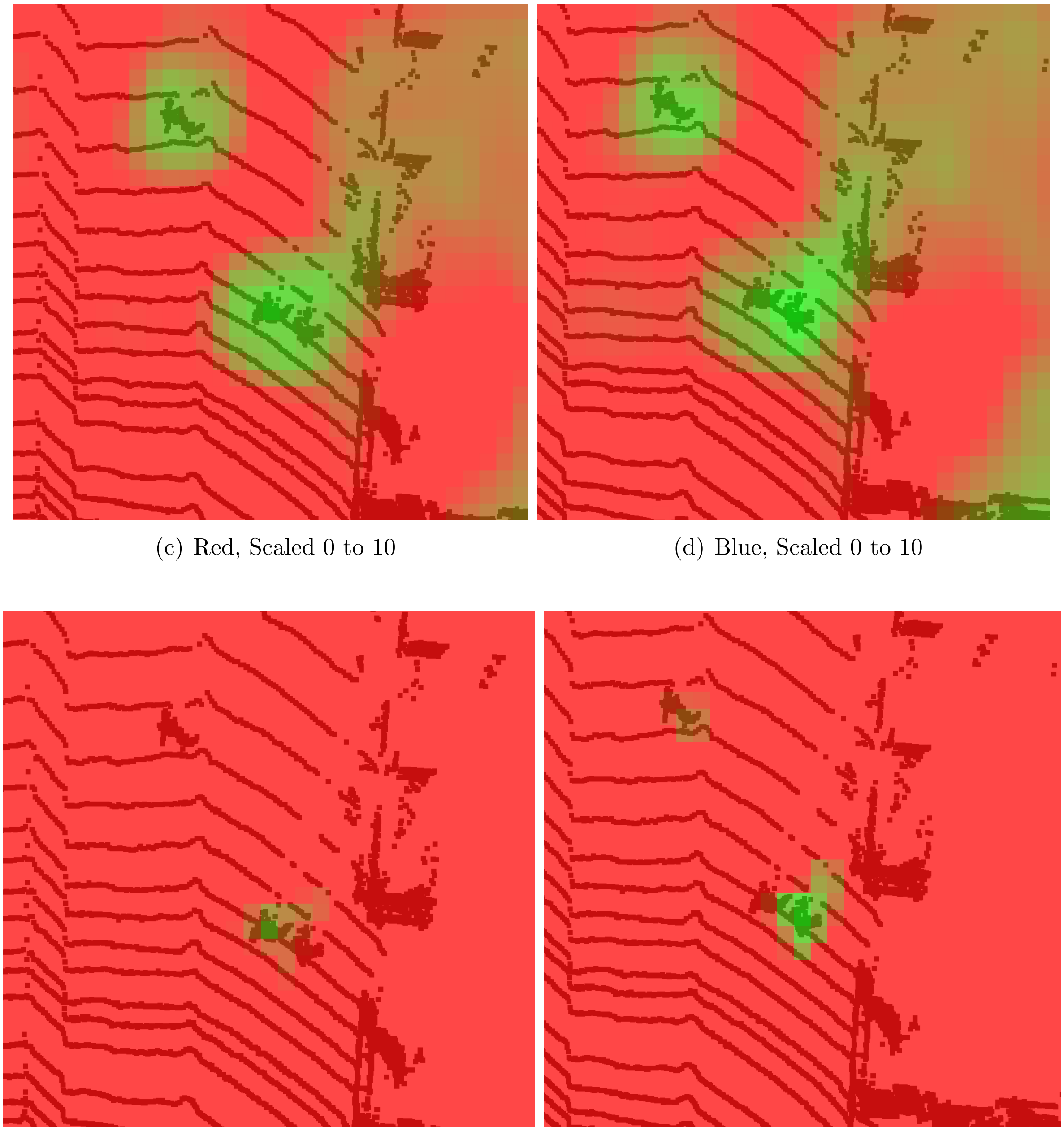
Finally, we evaluate the performance of the scene flow estimate. We perform this evaluation for cars, cyclists, and pedestrians, and also over all of the foreground classes. Results can be found in Table 4.2. We find a significant improvement in the error statistics of the scene flow estimate across all classes for our proposed feature learning approach over the occupancy constancy approach proposed in Chapter 3.

One downside of our method however is increased runtime. While the work presented in Chapter 3 had real-time performance (i.e., runtime of under 100 ms for 10 Hz data), our method takes about 188 ms on average using a machine with an Intel i7 CPU and an NVIDIA



(a) P_1, top-down view

(b) Camera Image of the Scene



(c) Red, Scaled 0 to 10

(d) Blue, Scaled 0 to 10

(e) Red, Scaled 0 to 3

(f) Blue, Scaled 0 to 3

Figure 4.7: A visual look at the distances in the feature space for two pedestrians. For the three locations from P_1 in Fig. 4.7(a), we evaluate the distance in feature space, T_{distance}, to locations in the successive scan P_2. A image of the scene is shown in Fig. 4.7(b), where the pedestrians can be seen on the right. Distances to the red point are shown in Fig. 4.7(c) and Fig. 4.7(f), and distances to the blue point are shown in Fig. 4.7(d) and Fig. 4.7(e). Green indicates small distances and red indicates large distances. The colormap ranges from a feature space distance of 0 to 10 for Fig. 4.7(c) and Fig. 4.7(d) and 0 to 3 for Fig. 4.7(f) and Fig. 4.7(e). Best viewed in color.

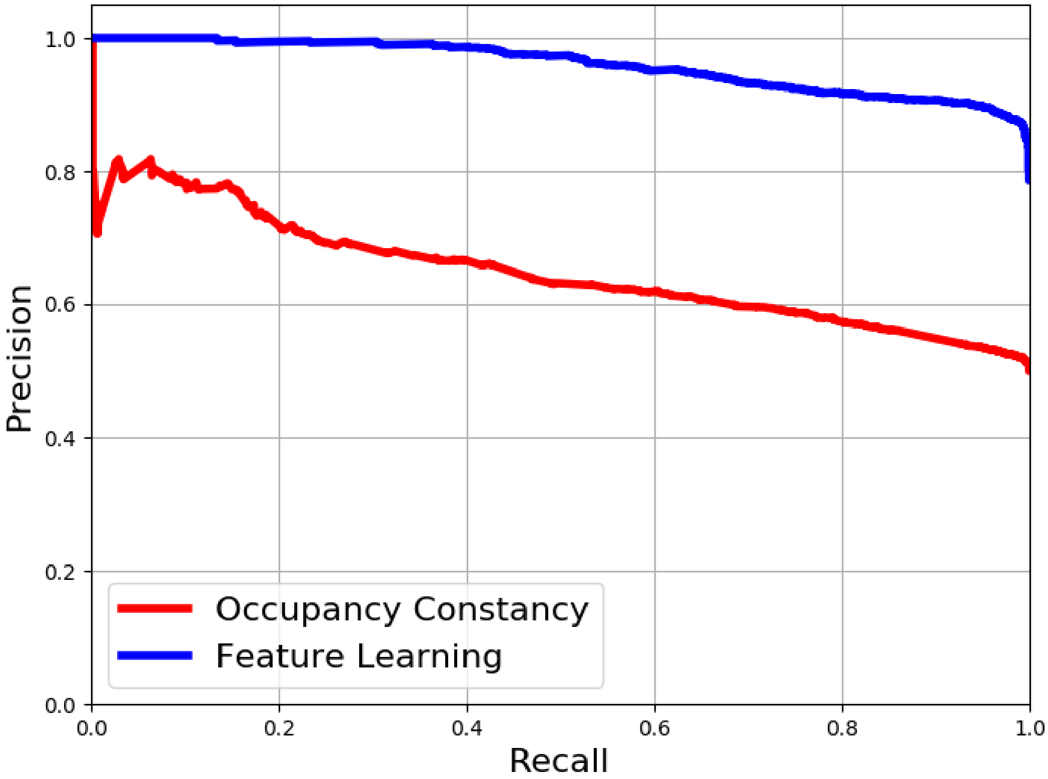


Figure 4.8: Performance of classifiers based on occupancy constancy and feature learning. We compare precision-recall curves for each classifier. We find that feature learning results in a more discriminative classifier.

		Occupancy Constancy	Feature Learning
Car	Mean Error	19.3 cm	15.6 cm
	Within 30 cm	83.8%	89.2 %
Cyclist	Mean Error	24.5 cm	15.9 cm
	Within 30 cm	78.8%	93.1 %
Pedestrian	Mean Error	38.9 cm	22.9 cm
	Within 30 cm	74.3%	89.0 %
All Dynamic Objects	Mean Error	22.1 cm	16.4 cm
	Within 30 cm	81.4%	88.2 %

Table 4.2: Error statistics for the scene flow estimate. We present results by class and over the entire foreground.

GeForce GTX 1080 GPU. We attribute this added runtime mainly to the time it takes to pass the input data through the network and evaluate pairwise distances in feature space.

4.6 Conclusion

In this work, we have presented a feature learning method for scene flow estimation from LIDAR data. We train an encoding network to extract features from an occupancy grid. This learned feature space is then leveraged in an energy minimization problem to solve for scene flow between successive scans. This approach yields improved results, both directly in the feature space itself and in the improved scene flow estimate, that beats the current state-of-the-art.

CHAPTER 5

Conclusion

To safely and smartly operate in their environment, autonomous vehicles must be able to understand the dynamic world around them. To do so involves problems in dynamic motion estimation. Accordingly, in this thesis, we have presented three novel methods for dynamic motion estimation that improve upon the performance of current methods. We focus on the application of autonomous vehicles using light detection and ranging (LIDAR) sensors, a prevalent sensing modality used by such platforms.

5.1 Contributions

The contributions of this thesis include:

- In Chapter 2, we proposed a framework for simultaneously estimating the trajectory and shape of a dynamic object in the environment that is being tracked. Unlike other work in this area, we used continuous-time estimation tools to properly model the rolling-shutter nature of multiple unsynchronized LIDAR sensors. We evaluated this method on a real-world dataset and demonstrated improved performance over a baseline tracker. We present results that show lower tracking error and higher fidelity dynamic object models.

- In Chapter 3, we proposed a framework for estimating temporal scene flow from a stream of LIDAR data. Inspired by brightness constancy from optical flow, we proposed occupancy constancy to measure the consistency in geometric structure between locations in occupancy grids from successive timestamps. Notably, this pipeline does not rely on any segmentation or data association result. By designing our algorithms to exploit a graphics processing unit (GPU), this method is able to run in real time. We evaluated our proposed method using the KITTI dataset and presented results that rival state-of-the-art obstacle trackers in the accuracy of the motion estimate, despite the fact that our method is independent of accurate segmentation or data association.

- In Chapter 4, we proposed a feature learning algorithm for LIDAR point clouds using deep learning. The network was trained so that distances in the learned feature space can be used to compare the similarity of different locations in the environment. These distances were used to replaced the occupancy constancy metric from Chapter 3. We demonstrated how feature learning can significantly improve the flow estimate. We presented results that show reduced error in the dynamic motion estimate for all classes of objects.

5.2 Future Work

While this thesis has made several contributions in the area of dynamic motion estimation, there is much room for improvement of these methods. In this section, we briefly discuss some avenues for future work.

5.2.1 Improved Object Modeling

While the point cloud model in Chapter 2 is general and expressive, it has several limitations. For example, there is no representation of free, occluded, and unknown space, which can lead to tracking errors. Also, dependencies between the points in the point cloud are not modeled. We might, for example expect the point cloud of an object to have certain smoothness or planar properties.

As seen in the history of object tracking, improved object models often to lead to improved tracking results. For instance, a mesh model or a signed distance function could be used rather than a point cloud model. This would allow for explicitly modeling the surface of the object, which is what the sensor is really observing. These types of representations have found great success in related areas. In Newcombe, Fox, and Seitz (2015), a signed distance function representation is used to reconstruct and track non-rigid objects such as a person standing in front of a camera, in real-time. Furthermore, these object models can then be used to improve upon the segmentation or data association techniques that are currently used. The application of such representations to object tracking methods warrants further investigation.

5.2.2 Reasoning about Occlusions and Temporal Features

One key challenge of any tracking or motion estimation framework is dealing with appearance changes over time, such as occlusions. While the work presented in this thesis can handle short occlusions, some information, such as the flow tracklet filters in Chapter 3 or feature representations in Chapter 4, may be lost. Indeed, several systems in this area attempt to explicitly handle occlusions (Galceran, Olson, and Eustice, 2015). Dramatic appearance changes could hinder any motion estimation framework as well.

The techniques presented in this thesis could be adapted to better handle these situations, especially with the gridded filtering framework used to compute flow. As a first step, the filtering framework could easily be adapted to allow motion estimates to persist while propagating the uncertainty accordingly. More interestingly however, the features themselves could be better modeled over time and through occlusions as well. The feature learning framework from Chapter 4 could be extended to predict a feature uncertainty, either by directly outputting a feature covariance or through indirect methods (Gal, 2016). Such an approach would then naturally lead to incorporating the feature representation in the state filter. By doing so, it could allow for better reasoning in occlusions and better modeling of object appearance changes (for example, a cyclist leaving behind a bicycle and becoming a pedestrian). Additionally, a better representation of the dynamic environment using these improvements could not only answer questions such as "what will be at this location from the sensor data I see?", but also "what could emerge from this occluded region?". This depth of understanding would be instrumental in properly navigating through environments with difficult occlusions, such as tight turns or heavy traffic.

5.2.3 Incorporating Semantics

While this thesis focuses on autonomous driving, we have not explored the semantics present in typical urban driving environments. The information that could be leveraged includes not only the semantic labels of the environment, such as what areas belong to a road or a sidewalk, but also cues such as lane markings, traffic lights, and traffic signs. These semantics heavily influence how dynamic objects move and thus are critical in estimating their motion.

For example, imagine a curved road where two vehicles are driving towards each other in opposite lanes. The bend of the road could cause the velocity vectors of the vehicles to be pointing directly at each other. Normally, this situation would be a very dangerous scenario. However, the semantics present, such as the lane markings, allow us to be reasonably certain that there is no pending collision. In situations like this one, knowledge of the semantics is essential to fully understanding the dynamics of the environment.

While incorporating the semantic information of the environment can be critical, it can also be quite challenging. Some approaches consider semantics only after performing dynamic motion estimation. For example, Galceran et al. (2017) predicts high-level behaviors for other agents on the road, such as staying in a lane or performing a lane change maneuver, from tracking estimates. The predicted behaviors are then used to aid in planning a safe route for the autonomous vehicle. Incorporating the semantic information at an earlier stage would allow for more accurate dynamic motion estimates. Additionally, it could be used to aid in related tasks such as segmentation or data association.

APPENDICES

APPENDIX A

GPU Programming

A graphics processing unit (GPU) is a specialized processor that can dramatically improve the run-time performance of parallelizeable algorithms. It is capable of running many threads of execution in parallel with a single instruction multiple data (SIMD) architecture. Originally developed for computer graphics, especially for video games, they are now used in many applications, including computer vision and deep learning. With significant speed-up in run-time, they have enabled real-time performance of many tasks that otherwise would be intractable.

While several different manufacturers develop GPUs, in our work we focus on NVIDIA GPUs that support Compute Unified Device Architecture (CUDA) (NVIDIA, 2018a). Additionally, NVIDIA provides a library that provides many commonly used parallel algorithms (such as reductions or parallel sorting), known as thrust (NVIDIA, 2018b). There are numerous features and capabilities available for an NVIDIA GPU. In this appendix, we briefly review functionality and features that are required for the run-time performance presented in Chapter 3 and Chapter 4. An overview is shown in Fig. A.1.

A.1 CUDA

CUDA is a platform provided by NVIDIA that extends `C/C++` (NVIDIA, 2018a). It provides the capability to design and create applications that can take advantage of a GPU's high performance. The application programming interface (API) provides the capability to write custom routines that are compiled to run on GPU hardware. These routines are referred to as CUDA kernels.

A CUDA kernel can be run in parallel on a GPU. The number and layout of GPU threads that run the kernel are determined when the kernel is called, referred to as configuring the kernel. A block of threads can share certain resources, such as shared memory or registers, and also be synchronized if necessary. A kernel can be configured to run using many blocks of

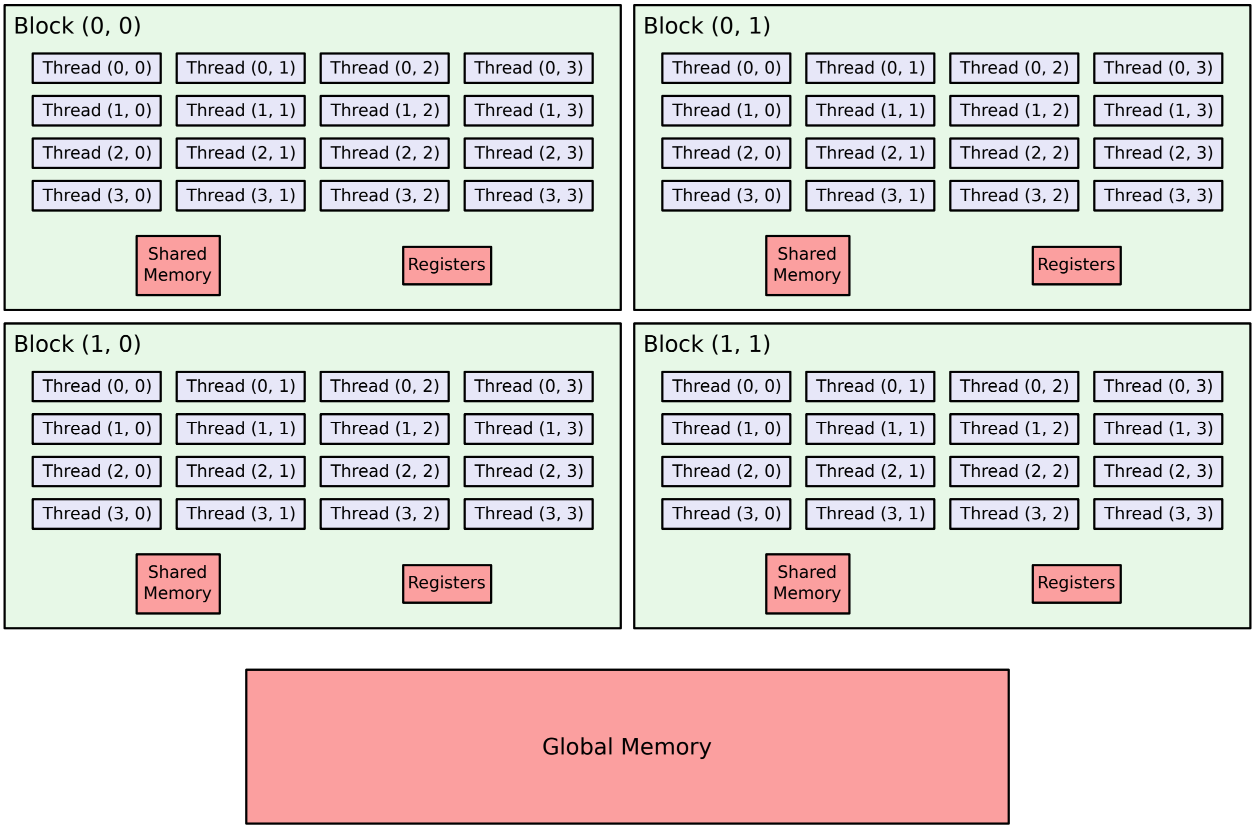


Figure A.1: An overview of GPU functionality and features. Each block, depicted in green, has many threads associated with it, depicted in blue. In each block, certain resources are shared amongst threads, including shared memory and registers, shown in red for every block. The GPU can be configured to run a number of blocks for a given CUDA kernel call. These blocks must share certain global resources, such as global memory, shown in red. In practice, the number of blocks and threads per block is often much higher. Threads and blocks are shown here as both indexed in two dimensions, but one, two, or three dimensions may be used, and may be different for blocks and threads.

threads, but different blocks are not necessarily synchronous and may run at slightly different times. Depending on the configuration, each block or thread may be indexed in one, two, or three dimensions.

A CUDA kernel is typically called using the following syntax:

```
<<<block_dim, thread_dim>>>MyCUDAKernel(data);
```

`block_dim` is the size (in one, two, or three dimensions) of the block of threads that are run. `thread_dim` is the size (in one, two, or three dimensions) of the threads that each block runs. Typically, the number of threads used is a multiple of 32 for optimal performance.

A.2 Memory Management

GPU global memory, also referred to as device memory, is managed separately from host memory (i.e., memory that is used on the central processing unit (CPU)). Thus, a CUDA kernel cannot directly access any memory that is not stored on the GPU. Typically, data that is needed by a CUDA kernel is first copied from host memory to device memory. Any resulting data must also first be copied from device memory to host memory before it can be accessed from the CPU. For optimal run-time performance. the number of memory allocations and transfers is kept at a minimum.

While global memory can be accessed at any time from inside a CUDA kernel, the access time can quite high. Access time is particularly slow if each thread must access a seemingly random index of memory. Indeed, many CUDA kernels are limited by the global memory bandwidth.

However, in two special cases, global memory access can be made more efficient. In the first, all threads access the same location in global memory. In the second case, known as memory coalescing, each thread accesses a sequential index in memory, according to a given stride (i.e., thread 1 accesses memory location 16004, thread 2 accesses memory location 16008, thread 3 accesses memory location 16012, and so on). While the specifics may vary between different models of GPUs, designing a CUDA kernel and memory layout to take advantage of memory coalescing can significantly improve the performance of kernels that are bottlenecked by global memory operations.

Another strategy for dealing with limitations on global memory bandwidth is the use of shared memory. In each block of threads, a small amount of memory, typically about tens of kilobytes, can be used to share data between the threads of the block. To make use of the shared memory, each thread in the CUDA kernel loads some data from global memory and stores it in shared memory. Then, each thread can use the cached data stored in shared memory rather than having to access the much slower global memory. Thus, each index in global memory will only be accessed once. Shared memory is especially for useful for algorithms where similarly located threads will need to access data from a similar neighborhood (e.g., convolution in a two-dimensional (2D) image).

A CUDA kernel that uses shared memory is typically called using the following syntax:

```
<<<block_dim, thread_dim, shared_memory_size>>>MyCUDAKernelSM(data);
```

shared_memory_size is the number of bytes of shared memory required by the kernel.

	NVIDIA GeForce GTX 1080	**NVIDIA GeForce GTX TITAN X**
CUDA Capability Version	6.1	6.1
Total Global Memory	8114 MB	12196 MB
Memory Clock Rate	5005 MHz	5005 MHz
Maximum Threads per Block	1024	1024
Shared Memory Per Block	49152 bytes	49152 bytes
Maximum Threads per Multiprocessor	2048	2048
Multiprocessors	20	28
CUDA Cores per Multiprocessor	128	128
CUDA Cores	2560	3594
Maximum Clock Rate	1734 MHz	1531 MHz

Table A.1: Key GPU specifications. We show specifications for the NVIDIA GeForce GTX 1080 and the NVIDIA GeForce GTX TITAN X.

A.3 Thrust

Thrust is a library provided by NVIDIA that implements many commonly used parallel algorithms (NVIDIA, 2018b), similar to how Standard Template Library (STL) provides library `C++` functionality. These algorithms are efficiently implement to run on the GPU and include reduction operations (such as summing an array of numbers), sorting operations, and many more. Conveniently, thrust routines can be called directly without use of a CUDA kernel. The number of blocks, threads, and any shared memory is automatically managed. However, data must be properly stored in GPU global memory.

A.4 GPU Specifications

NVIDIA has provided a wide array of GPU products over the years. While their functionality is similar, their exact specifications and set of available features vary. In the work presented in Chapter 3 and Chapter 4, the NVIDIA GTX 1080 and the NVIDIA GTX TITAN X were used. Some key specifications from these two devices are shown in Table A.1.

APPENDIX B

Efficient Construction of Occupancy Grids using a GPU

In Chapter 3 and Chapter 4, we compute a probabilistic occupancy grid constructed from a point cloud collected by a light detection and ranging (LIDAR) sensor. As runtime performance is important for these systems, we must take care to efficiently implement this occupancy grid construction. We do so by writing custom CUDA code (Nickolls et al., 2008) to run on an NVIDIA GPU, such as a TITAN X or a GTX 1080. In this appendix, we describe our GPU occupancy construction method. A brief overview of GPU programming methods used here is provided in Appendix A.

B.1 Occupancy Grids

In constructing an occupancy grid, our goal is to represent the observed LIDAR point-cloud with an occupancy grid (Hornung et al., 2013; Moravec and Elfes, 1985) composed of 3D voxels. Each voxel represents the probability of being occupied given a LIDAR scan composed of n points $\mathbf{z}_{1:n} = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ in the vehicle reference frame. The position of the LIDAR sensor when each point was observed is given by $\mathbf{s}_{1:n}$. Depending on the application and setup, these can all be the same (for example, in the case of a stationary sensor), all different (in the case of a moving platform), or one of a few discrete values (such as with a sensor array consisting of multiple LIDAR sensors).

For each voxel v, the probability of it being occupied can be expressed by

$$p(v|\mathbf{z}_{1:n}) = \left[1 + \frac{1-p(v|\mathbf{z}_n)}{p(v|\mathbf{z}_n)} \frac{1-p(v|\mathbf{z}_{1:n-1})}{p(v|\mathbf{z}_{1:n-1})} \frac{p(v)}{1-p(v)}\right]^{-1}, \tag{B.1}$$

where $p(v)$ is a prior on v. We assume a non-informative prior $p(v) = 0.5$.

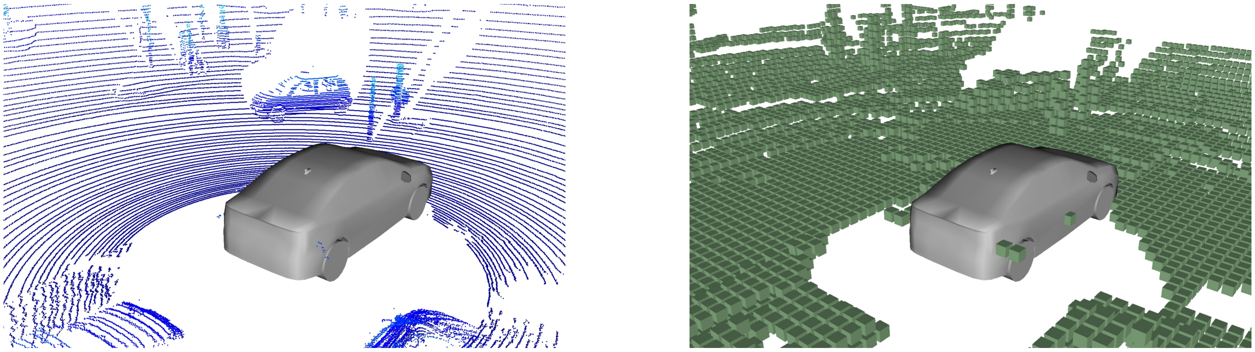

(a) LIDAR scan (b) Occupancy Grid

Figure B.1: Sample point cloud and occupancy grid from the KITTI dataset. Note the occupied space (shown in green boxes).

We replace probabilities with log-odds notation (L) (Hornung et al., 2013), where:

$$\begin{aligned} L(x) &= \log \frac{p(x)}{p(\neg x)} && \text{(B.2)} \\ &= \log \frac{p(x)}{1 - p(x)}. && \text{(B.3)} \end{aligned}$$

Using log-odds notation, we can rewrite (B.1) as:

$$L(v|z_{1:n}) = \sum_{i=1}^{n} L(v|z_i) \tag{B.4}$$

$$L(v|z_i) = \begin{cases} l_{\text{free}} & \text{the ray from } s_i \text{ to } z_i \text{ passes through } v \\ l_{\text{occupied}} & \text{the ray from } s_i \text{ to } z_i \text{ ends in } v \\ 0 & \text{otherwise} \end{cases}, \tag{B.5}$$

where each beam is treated as independent and $L(v|z_i)$ represents the update due to the ith beam.

Optionally, we can avoid over-confidence by clipping the log-odds once the probability is sufficiently confident one way or another so that they are always in the range $[l_{\text{min}}, l_{\text{max}}]$. Clipping is a common technique when dealing with log-odds.

We can see an example of a point cloud and its corresponding occupancy grid in Fig. B.1, taken from the KITTI dataset.

B.2 GPU Algorithm

To efficiently compute the occupancy grid, we use the following GPU algorithm.

Let m_r be the maximum range of the sensor. Let $n_{\max}$ be the maximum number of LIDAR observations n we ever expect to receive. Let r be the resolution of the occupancy grid we wish to construct.

B.2.1 Preallocation

In preprocessing, we preallocate several buffers in GPU memory. These buffers include a list of voxel locations and log-odds updates. The size of each is given by $n_{\max}\lceil\frac{m_r}{r}\rceil$. These buffers are reused each time we construct an occupancy grid. By preallocating them, we avoid the latency of GPU memory allocation operations at runtime.

B.2.2 Computing a List of Updates

We are given a point cloud $\mathbf{z}_{1:n}$, from which we wish to construct an occupancy grid. First, we compute a list of log-odds updates. To do so, we build a custom CUDA kernel. The CUDA kernel is configured so that each CUDA thread processes a single observation.

Consider an observation $\mathbf{z}_i$. The position of the sensor is given by $\mathbf{s}_i$. These two locations must first be transformed into the reference frame of interest, which could be a global coordinate frame, a reference frame with an origin on the vehicle body, or in the reference frame of the LIDAR sensor itself. For our work in Chapter 3 and Chapter 4, we processed the data in the sensor frame. Once the appropriate transformations have been applied, let $\mathbf{p}_0$ and $\mathbf{p}_1$ be the sensor origin and the LIDAR observation point, respectively. $\mathbf{p}_0$ may be constant for the point cloud, but in the general case it may vary for each observation.

We compute the starting and terminating voxels, v_s and v_m, for the observation according to $\mathbf{p}_0$, $\mathbf{p}_1$, and r.

Now, we employ Bresenham's algorithm (Bresenham, 1965) to trace out a path from v_s to v_m. For each voxel v in this path, we add both v and the log-odds update, l_{free}, to the location and log-odds update buffer we have preallocated. When we reach v_m, we write the log-odds update l_{occupied} to the buffer. Any unused portion of the buffer is set to log-odds updates of 0. A 2D analog of Bresenham's algorithm is illustrated in Fig. B.2.

For optimal performance, we take care to index these buffers in order to take advantage of memory coalescing on the GPU. By having each CUDA thread access a sequential address in memory, the memory access times are greatly reduced due to GPU hardware optimization.

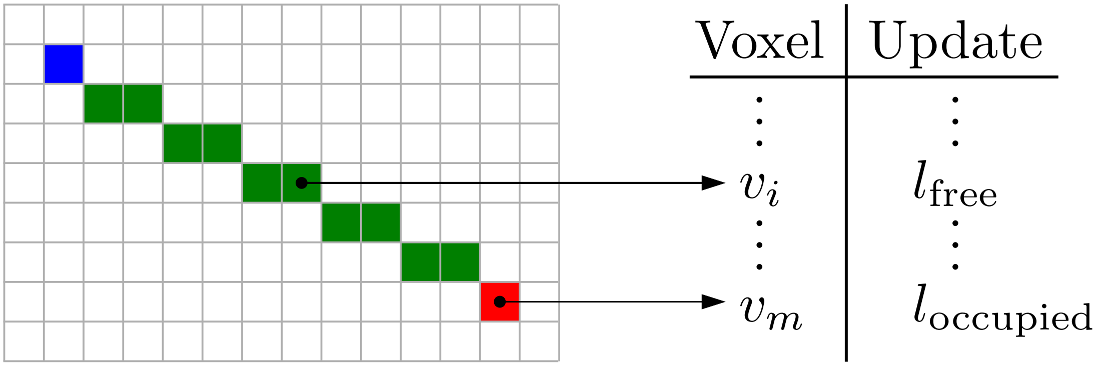


Figure B.2: Cartoon example of ray tracing. Starting from the blue voxel containing the beam origin $\mathbf{p}_0$, we trace a path to the red voxel, containing $\mathbf{p}_1$, using Bresenham's algorithm. Each voxel that is intersected by the ray is added to the list with an l_{free} update. The terminating voxel, v_m, is marked with an l_{occupied} update.

Once the CUDA kernel call is complete, we have filled our preallocated buffers with a list of log-odds updates and the corresponding voxel locations.

B.2.3 Reductions

To finish building the occupancy grid, we must finish processing our list of log-odds updates. We can do so with a set of operation that are efficiently implemented in NVIDIA's thrust library (NVIDIA, 2018b).

First, we use a `remove_if` operation to remove all log-odds updates that are set to zero (i.e., there is no update to process). Then, we sort the updates by the voxel location so that updates for the same voxel location are always contiguous. We do so by using a `sort_by_key` operation. Lastly, we use a `reduce_by_key` reduction to sum the log-odds updates for every voxel location together.

Once these thrust operations have been completed, we have constructed a mapping from each voxel v to $L(v|\mathbf{z}_{1:n})$. Note that this mapping is inherently sparse due to the nature of the algorithm; voxels that are not updated by an observation are not included in the map. This mapping can be used to compute a dense occupancy grid if desired.

Paramter	**Description**	**Value**
m_r	Maximum sensor range	100 m
r	Occupancy grid resolution	30 cm
n_{max}	Maximum number of observations	150000
l_{free}	Log-odds for free space	-0.1
l_{occupied}	Log-odds for occupied space	1.0
l_{min}	Minimum log-odds	-3.0
l_{max}	Maximum log-odds	3.0

Table B.1: Parameter values for occupancy grid construction.

B.3 Parameter Selection

See Table B.1 for a list of parameters and the values we used in our work.

B.4 Runtime

Using a NVIDIA GTX 1080 GPU, creating an occupancy grid from a point cloud from the KITTI dataset takes about 12.5 ms on average. The runtime can vary with the size of the point cloud, distribution of the observations, and the choice of GPU.

BIBLIOGRAPHY

BIBLIOGRAPHY

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

S. Anderson and T. Barfoot. Towards relative continuous-time SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1033–1040, Karlsruhe, Germany, May 2013.

S. Anderson, F. Dellaert, and T. D. Barfoot. A hierarchical wavelet decomposition for continuous-time SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 373–380, Hong Kong, China, May/June 2014.

M. Aubry, U. Schlickewei, and D. Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1626–1633, Barcelona, Spain, June 2011.

O. Aycard, A. Spalanzani, J. Burlet, C. Fulgenzi, T.-D. Vu, D. Raulo, and M. Yguel. Pedestrian tracking using offboard cameras. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 513 – 518, Beijing, China, October 2006.

A. Azim and O. Aycard. Detection, classification and tracking of moving objects in a 3d environment. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 802–807, Madrid, Spain, June 2012.

H. Badino, D. Huber, and T. Kanade. The CMU visual localization data set, 2011.

C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 28 (3):24, 2009.

T. Basha, Y. Moses, and N. Kiryati. Multi-view scene flow estimation: A view centered variational approach. *International Journal of Robotics Research*, 101(1):6–21, 2013.

M. Baum and U. D. Hanebeck. Extended Object Tracking with Random Hypersurface Models. *IEEE Transactions on Aerospace and Electronic Systems*, 50:149–159, 2014.

A. Behl, O. H. Jafari, S. K. Mustikovela, H. A. Alhaija, C. Rother, and A. Geiger. Bounding boxes, segmentations and object coordinates: How important is recognition for 3d scene flow estimation in autonomous driving scenarios? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2583, Honolulu, HI, USA, June 2017.

A. Behl, D. Paschalidou, S. Donné, and A. Geiger. Pointflownet: Learning representations for 3d scene flow estimation from point clouds. *arXiv preprint arXiv:1806.02170*, 2018.

S. Beningo, M. Chambers, C. Ford, K. Notis, M. Liu, and S. Smith-Pickel. National transportation statistics. *United States Bureau of Transportation Statistics*, 2018.

J.-L. Blanco-Claraco, F.-A. Moreno-Duenas, and J. Gonzalez-Jimenez. The Malaga urban dataset: High-rate stereo and LiDAR in a realistic urban scenario. *International Journal of Robotics Research*, 33(2):207–214, 2014.

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

M. M. Bronstein and I. Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1704–1711, San Francisco, CA, USA, June 2010.

A. Byravan and D. Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 173–180, Singapore, May/June 2017.

N. Carlevaris-Bianco and R. M. Eustice. Learning visual feature descriptors for dynamic lighting conditions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2769–2776, Chicago, IL, USA, September 2014.

N. Carlevaris-Bianco, A. K. Ushani, and R. M. Eustice. University of Michigan North Campus long-term vision and lidar dataset. *International Journal of Robotics Research*, 35(9): 1023–1035, 2015.

S. Ceriani, G. Fontana, A. Giusti, D. Marzorati, M. Matteucci, D. Migliore, D. Rizzi, D. G. Sorrenti, and P. Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27(4):353–371, 2009.

S. Choi, K. Lee, and S. Oh. Robust modeling and prediction in dynamic environments using recurrent flow networks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1737–1742, Daejeon, South Korea, October 2016.

C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision*, pages 628–644, Amsterdam, Netherlands, October 2016.

A. G. Cunningham, E. Galceran, R. M. Eustice, and E. Olson. MPDM: Multipolicy decision-making in dynamic, uncertain environments for autonomous driving. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1670–1677, Seattle, WA, USA, May 2015.

A. Dai, C. R. Qi, and M. Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5868–5877, Honolulu, HI, USA, July 2017.

A. Dai, D. Ritchie, M. Bokeloh, S. Reed, J. Sturm, and M. Nießner. Scancomplete: Large-scale scene completion and semantic segmentation for 3d scans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, Salt Lake City, UT, USA, June 2018.

R. Danescu, F. Oniga, and S. Nedevschi. Modeling and tracking the driving environment with a particle-based occupancy grid. *IEEE Transactions on Intelligent Transportation Systems*, 12(4):1331–1342, 2011.

M. Darms, P. Rybski, and C. Urmson. Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 1197–1202, Eindhoven, Netherlands, June 2008.

C. de Boor. *A Practical Guide to Splines*. Springer Verlag (New York), 1978.

J. Dequaire, P. Ondrúška, D. Rao, D. Wang, and I. Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *International Journal of Robotics Research*, 37(4-5):492–512, 2017.

A. Dewan, T. Caselitz, G. D. Tipaldi, and W. Burgard. Rigid scene flow for 3d lidar scans. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1765–1770, Daejeon, South Korea, October 2016.

B. Douillard, D. Fox, F. Ramos, and H. Durrant-Whyte. Classification and semantic mapping of urban environments. *International Journal of Robotics Research*, 30(1):5–32, 2010.

M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1355–1361, Singapore, May/June 2017.

M. Fallon, H. Johannsson, M. Kaess, and J. J. Leonard. The MIT Stata Center dataset. *International Journal of Robotics Research*, 32(14):1695–1699, 2013.

H. Fan, H. Su, and L. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 605–613, Honolulu, HI, USA, July 2017.

A. Feldman, M. Hybinette, and T. Balch. The multi-iterative closest point tracker: An online algorithm for tracking multiple interacting targets. *Journal of Field Robotics*, 29(2): 258–276, 2012.

D. Ferstl, G. Riegler, M. Ruether, and H. Bischof. Cp-census: A novel model for dense variational scene flow from rgb-d data. In *Proceedings of the British Machine Vision Conference*, Nottingham, UK, September 2014.

T. N. Firestine, K. Notis, and S. Randrianarivelo. Transportation economic trends 2017. 2018.

P. Furgale, T. D. Barfoot, and G. Sibley. Continuous-time batch estimation using temporal basis functions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2088–2095, St. Paul, MN, USA, May 2012.

Y. Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction. In *Proceedings of the Robotics: Science & Systems Conference*, Rome, Italy, July 2015.

E. Galceran, E. Olson, and R. M. Eustice. Augmented vehicle tracking under occlusions for decision-making in autonomous driving. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3559–3565, Hamburg, Germany, September 2015.

E. Galceran, A. G. Cunningham, R. M. Eustice, and E. Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction: Theory and experiment. *Autonomous Robots*, 41(6):1367–1382, 2017.

A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, 2013.

R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *Proceedings of the European Conference on Computer Vision*, pages 484–499, Amsterdam, Netherlands, October 2016.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the Advances in Neural Information Processing Systems Conference*, pages 2672–2680, Montreal, Canada, 2014.

V. Guizilini and F. Ramos. Learning to reconstruct 3d structures for occupancy mapping. In *Proceedings of the Robotics: Science & Systems Conference*, Cambridge, MA, USA, July 2017.

S. Hadfield and R. Bowden. Kinecting the dots: Particle based scene flow from depth sensors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2290–2295, Barcelona, Spain, November 2011.

D. Held, J. Levinson, and S. Thrun. Precision tracking with sparse 3d and dense color 2d data. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1138–1145, Karlsruhe, Germany, May 2013.

D. Held, J. Levinson, S. Thrun, and S. Savarese. Combining 3d shape, color, and motion for robust anytime tracking. In *Proceedings of the Robotics: Science & Systems Conference*, Berkeley, CA, USA, July 2014.

D. Held, J. Levinson, S. Thrun, and S. Savarese. Robust real-time tracking combining 3d shape, color, and motion. *International Journal of Robotics Research*, 35(1-3):30–49, 2016.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1): 185–203, 1981.

M. Hornacek, A. Fitzgibbon, and C. Rother. Sphereflow: 6 dof scene flow from rgb-d pairs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3526–3533, Columbus, OH, USA, 2014.

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

Y. Hu, R. Song, and Y. Li. Efficient coarse-to-fine patchmatch for large displacement optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5704–5712, Las Vegas, NV, USA, June/July 2016.

F. Huguet and F. Devernay. A variational method for scene flow estimation from stereo sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–7, Rio de Janeiro, Brazil, October 2007.

M. Isard and J. MacCormick. Dense motion and disparity estimation via loopy belief propagation. In *Proceedings of the Asian Conference on Computer Vision*, pages 32–41, Hyderabad, India, 2006.

M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers. A primal-dual framework for real-time dense rgb-d scene flow. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 98–104, Chicago, IL, USA, May 2015.

A. H. Jazwinski. *Stochastic Processes and Filtering Theory.* Academic Press, Inc., 1970.

R. Kaestner, J. Maye, Y. Pilat, and R. Siegwart. Generative object detection and tracking in 3d range data. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3075–3081, St. Paul, MN, USA, May 2012.

R. Lanctot. Accelerating the future: The economic impact of the emerging passenger economy. 2017.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

J. Leonard et al. Team MIT Urban Challenge technical report. Technical Report MIT-CSAIL-TR-2007-058, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, USA, December 2007.

J. Levinson and S. Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4372–4378, Anchorage, AK, USA, May 2010.

J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 163–168, Baden-Baden, Germany, June 2011.

C.-H. Lin, C. Kong, and S. Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, New Orleans, LA, USA, February 2018.

C. Liu, J. Yuen, A. Torralba, J. Sivic, and W. T. Freeman. SIFT flow: Dense correspondence across different scenes. In *Proceedings of the European Conference on Computer Vision*, pages 28–42, Marseille, France, October 2008.

J. Liu, F. Yu, and T. Funkhouser. Interactive 3d modeling with a generative adversarial network. In *Proceedings of the International Conference on 3D Vision*, pages 2278–2324, Quingdao, China, October 2017.

X. Liu, C. R. Qi, and L. J. Guibas. Learning scene flow in 3d point clouds. *arXiv preprint arXiv:1806.01411*, 2018.

B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, Vancouver, Canada, August 1981.

A. Masatoshi, C. Yuuto, T. Kanji, and Y. Kentaro. Leveraging image-based prior in cross-season place recognition. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5455–5461, Seattle, WA, USA, May 2015.

M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, Boston, MA, USA, June 2015.

M. Milford and G. Wyeth. SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1643–1649, Saint Paul, MN, USA, May 2012.

F. Moosmann and T. Fraichard. Motion estimation from range images in dynamic outdoor scenes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 142–147, Anchorage, AK, USA, May 2010.

F. Moosmann and C. Stiller. Joint self-localization and tracking of generic objects in 3d range data. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1146–1152, Karlsruhe, Germany, May 2013.

H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, St. Louis, MO, USA, March 1985.

National Highway Traffic Safety Administration. Traffic safety facts 2016: Motor vehicle crash data from the fatality analysis reporting system (FARS) and the general estimates system (GES). *United States Department of Transportation*, 2018.

R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 343–352, Boston, MA, USA, June 2015.

J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. In *ACM SIGGRAPH*, page 16, 2008.

Norwegian Broadcasting Corporation. Nordlandsbanen: Minute by minute, season by season, 2013. URL https://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season/.

NVIDIA. CUDA Toolkit — NVIDIA Developer. https://developer.nvidia.com/cuda-toolkit, 2018a. Accessed: 2018-07-29.

NVIDIA. Thrust quick start guide, 2018b.

P. Ondruska and I. Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Phoenix, Arizona USA, February 2016.

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

D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, Las Vegas, NV, USA, June/July 2016.

A. Petrovskaya and S. Thrun. Model based vehicle tracking for autonomous driving in urban environments. In *Proceedings of the Robotics: Science & Systems Conference*, Zurich, Switzerland, June 2008.

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

C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5648–5656, Las Vegas, NV, USA, June/July 2016.

C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, Honolulu, HI, USA, July 2017a.

C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the Advances in Neural Information Processing Systems Conference*, pages 5105–5114, Long Beach, CA, USA, December 2017b.

F. Ramos and L. Ott. Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent. *International Journal of Robotics Research*, 35(14):1717–1730, 2016.

R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (FPFH) for 3d registration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3212–3217, Kobe, Japan, 2009.

S. Särkkä, A. Vehtari, and J. Lampinen. Rao-blackwellized particle filter for multiple target tracking. *Information Fusion*, 8(1):2–15, 2007.

A. V. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proceedings of the Robotics: Science & Systems Conference*, pages 435–442, Seattle, WA, USA, June/July 2009.

R. Senanayake, L. Ott, S. O'Callaghan, and F. T. Ramos. Spatio-temporal hilbert maps for continuous occupancy representation in dynamic environments. In *Proceedings of the Advances in Neural Information Processing Systems Conference*, pages 3925–3933, Barcelona, Spain, December 2016.

M. Sheehan, A. Harrison, and P. Newman. Self-calibration for a 3d laser. *International Journal of Robotics Research*, 31(5):6891–6900, 2012.

E. Smith and D. Meger. Improved adversarial systems for 3d object generation and reconstruction. In *Proceedings of the Conference on Robot Learning*, pages 87–96, Mountain View, CA, USA, November 2017.

M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman. The new college vision and laser data set. *International Journal of Robotics Research*, 28(5):595–599, 2009.

S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6891–6900, Honolulu, HI, USA, July 2017.

D. Streller, K. Furstenberg, and K. Dietmayer. Vehicle and object models for robust tracking in traffic scenes using laser range images. In *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, pages 118–123, Singapore, September 2002.

N. Suenderhauf. The VPRiCE challenge 2015 — visual place recognition in changing environments. https://roboticvision.atlassian.net/wiki/pages/viewpage.action?pageId=14188617, 2015.

N. Sunderhauf, P. Neubert, and P. Protzel. Are we there yet? Challenging SeqSLAM on a 3000 km journey across all four seasons. In *ICRA Workshop on Long-Term Autonomy*, pages 1–3, Karlsruhe, Germany, May 2013.

R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields. In *Proceedings of the European Conference on Computer Vision*, pages 16–29, Graz, Austria, 2006.

R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080, 2008.

T. Taniai, S. N. Sinha, and Y. Sato. Fast multi-frame stereo scene flow with motion segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6891–6900, Honolulu, HI, USA, June 2017.

G. Tanzmeister, J. Thomas, D. Wollherr, and M. Buss. Grid-based mapping and tracking in dynamic environments using a uniform evidential environment representation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 6090–6095, Hong Kong, China, May/June 2014.

A. Teichman and S. Thrun. Tracking-based semi-supervised learning. *International Journal of Robotics Research*, 31(7):804–818, 2012.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics.* MIT press, 2005.

F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *Proceedings of the European Conference on Computer Vision*, pages 356–369, Crete, Greece, September 2010.

A. K. Ushani and R. M. Eustice. Feature learning for scene flow estimation from lidar data. In *Proceedings of the Conference on Robot Learning*, Zurich, Switzerland, October 2018.

A. K. Ushani, N. Carlevaris-Bianco, A. G. Cunningham, E. Galceran, and R. M. Eustice. Continuous-time estimation for dynamic obstacle tracking. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1137–1143, Hamburg, Germany, September 2015.

A. K. Ushani, R. W. Wolcott, J. M. Walls, and R. M. Eustice. A learning approach for real-time temporal scene flow estimation from lidar data. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 5666–5673, Singapore, May/June 2017.

S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 722–729, Kerkyra, Greece, September 1999.

P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Machine Learning*, 11(Dec):3371–3408, 2010.

C. Vogel, K. Schindler, and S. Roth. Piecewise rigid scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1384, December 2013.

C. Vogel, S. Roth, and K. Schindler. View-consistent 3d scene flow estimation over multiple frames. In *Proceedings of the European Conference on Computer Vision*, pages 263–278, Zurich, Switzerland, September 2014.

C. Vogel, K. Schindler, and S. Roth. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1):1–28, 2015.

T.-D. Vu and O. Aycard. Laser-based detection and tracking moving objects using data-driven markov chain monte carlo. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3800–3806, Kobe, Japan, May 2009.

T.-D. Vu, O. Aycard, and N. Appenrodt. Online localization and mapping with moving object tracking in dynamic outdoor environments. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 190–195, Istanbul, Turkey, June 2007.

C. Wang. *Simultaneous Localization, Mapping and Moving Object Tracking.* PhD thesis, Carnegie Mellon University, 2004.

C.-C. Wang, C. Thorpe, and A. Suppe. Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 416–421, Columbus, OH, USA, June 2003.

C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *International Journal of Robotics Research*, 26(9):889–916, 2007.

D. Z. Wang, I. Posner, and P. Newman. What could move? Finding cars, pedestrians and bicyclists in 3d laser data. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4038–4044, St. Paul, MN, USA, May 2012.

D. Z. Wang, I. Posner, and P. Newman. Model-free detection and tracking of dynamic objects with 2D lidar. *International Journal of Robotics Research*, 34(7):1039–1063, 2015.

W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann. Shape inpainting using 3d generative adversarial network and recurrent convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2298–2306, Venice, Italy, 2017.

A. Wedel, C. Rabe, T. Vaudrey, T. Brox, U. Franke, and D. Cremers. Efficient dense scene flow from sparse or dense stereo data. In *Proceedings of the European Conference on Computer Vision*, pages 739–751, Marseille, France, October 2008.

Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *Proceedings of the European Conference on Computer Vision*, pages 499–515, Amsterdam, Netherlands, October 2016.

S. Wender and K. Dietmayer. 3d vehicle detection using a laser scanner and a video camera. *Intelligent Transportation Systems, IET*, 2(2):105–112, 2008.

R. W. Wolcott and R. M. Eustice. Visual localization within LIDAR maps for automated urban driving. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 176–183, Chicago, IL, USA, September 2014.

R. W. Wolcott and R. M. Eustice. Fast LIDAR localization using multiresolution Gaussian mixture maps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2814–2821, Seattle, WA, USA, May 2015.

D. Wolf and G. S. Sukhatme. Online simultaneous localization and mapping in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1301–1307, New Orleans, LA, USA, April/May 2004.

J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Proceedings of the Advances in Neural Information Processing Systems Conference*, pages 82–90, Barcelona, Spain, December 2016.

Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, Boston, MA, USA, June 2015.

W. Xu, J. Pan, J. Wei, and J. Dolan. Motion planning under uncertainty for on-road autonomous driving. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2507–2512, Hong Kong, China, May 2014.

Z. Yan and X. Xiang. Scene flow estimation: A survey. *arXiv preprint arXiv:1612.02590*, 2016.

R. A. Yeh, C. Chen, T. Y. Lim, S. A. G., M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5485–5493, Honolulu, HI, USA, July 2017.

A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1802–1811, Honolulu, HI, USA, July 2017.

L. Zhao and C. Thorpe. Qualitative and quantitative car tracking from a range image sequence. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 496–501, Santa Barbara, CA, USA, June 1998.