Leveraging Motion and Semantic Cues for 3D Scene Understanding

Ayush Dewan

Technische Fakultät
Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dissertation zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften

Betreuer: Prof. Dr. Wolfram Burgard

Leveraging Motion and Semantic Cues for 3D Scene Understanding

Ayush Dewan

Dissertation zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
Technische Fakultät, Albert-Ludwigs-Universität Freiburg

Dekan	Prof. Dr. Rolf Backofen
Erstgutachter	Prof. Dr. Wolfram Burgard
	Albert-Ludwigs-Universität Freiburg
Zweitgutachterin	Prof. Dr. Maren Bennewitz
	Rheinische Friedrich-Wilhelms-Universität Bonn
Tag der Disputation	14. Mai 2020

Zusammenfassung

Die Automatisierung von Fahrzeugen nimmt rasant zu, wobei das Hauptaugenmerk auf der Verbesserung der Verkehrssicherheit liegt. Generell wird von Robotern erwartet, dass sie sich in einer ständig verändernden Umgebung zurechtfinden. Das Verständnis der zugrunde liegenden Ursachen für diese Veränderungen ist ein entscheidender Faktor, um den gewünschten Automatisierungsgrad zu erreichen. Veränderungen sind eng mit der Bewegung und den semantischen Eigenschaften der verschiedenen Entitäten in der Umgebung verflochten. In dieser Arbeit schlagen wir verschiedene Methoden vor um diese Eigenschaften abzuleiten und, als finales Ziel, eines ganzheitliches Verständnis der Szene mithilfe von 3D-LiDAR-Daten zu erlangen.

Roboter arbeiten häufig in einer nicht statischen Umgebung und teilen den Raum mit verschiedenen dynamischen Objekten. Solche zu erkennen und ihre Bewegungen vorherzusagen ist für eine sichere und effiziente Navigation notwendig. Um diese dynamischen Eigenschaften zu verstehen, beschäftigen wir uns damit Hinweise auf Bewegungen zu interpretieren. Für das schätzen einer Bewegung ist es notwendig dieselben Teile der Umgebung mehr als einmal zu beobachten und eine Assoziation zwischen den Beobachtungen herzustellen. Wir adressieren dieses Problem indem wir einen lokalen Merkmalsdeskriptor für 3D-LiDAR-Scans vorschlagen, der unter Verwendung eines tiefen neuronalen Faltungsnetzwerkes gelernt wurde. Ein solcher Deskriptor ermöglicht das Auffinden von Assoziationen zwischen Schlüsselpunkten und ebnet den Weg für die Schätzung von Bewegungsmodellen.

In dieser Arbeit schlagen wir eine neue Methode zur Erkennung und Verfolgung dynamischer Objekte vor. In einer iterativen Weise schätzen wir Starrkörperbewegungsmodelle für verschiedene Objekte in der Szene und erkennen anschließend dynamische Objekt ausschließlich basierend auf diesen Modellen. Zum Verfolgen verwenden wir erneut die Bewegungsinformation, um eine Assoziation zwischen Objekten in aufeinander folgenden Scans zu finden. Diese Methode setzt implizit voraus, dass sich eine Szene aus einer Menge von Objekten zusammensetzt. Um dynamische Eigenschaften auf einer feineren Ebene abzuleiten, schlagen wir eine neue Methode zur Schätzung des dichten Starrkörperbewegungsfeldes vor. Diese Methode basiert auf der einzigen Annahme, dass Objekte lokal starr sind und kann beliebige unterschiedliche Bewegungen sowohl für starre als auch für nicht starre Objekte schätzen. Um den Bewegungszustand von Punkten in einem LiDAR-Scan abzuleiten, schlagen wir eine auf einem Hidden-Markov-Modell basierende Methode vor, die das Bewegungsfeld als Messquelle verwendet.

Die dynamischen Eigenschaften stehen in engem Zusammenhang mit den semantischen Eigenschaften verschiedener Objekte in der Umgebung. Um diese zu extrahieren, schlagen wir ein tiefes Faltungsnetz für die semantische Segmentierung von 3D-LiDAR-Scans vor. Die von einem LiDAR-Scanner oder anderem Sensor gesammelten Daten sind sequentiell, was wir mithilfe eines Bayes-Filter-Ansatzes ausnutzen, um diese semantischen Vorhersagen zeitlich konsistent zu machen. Der Filter nutzt die Vorhersage des Netzwerks aus den aktuellen und vorherigen Scans, wodurch das System robust gegenüber isolierten falschen Vorhersagen des Netzwerkes gemacht wird. Um die inhärente Beziehung zwischen Bewegung und semantischen Eigenschaften auszunutzen, schlagen wir einen neuen Ansatz vor der die Punkte in einem LiDAR-Scan als unbeweglich, beweglich oder dynamisch klassifiziert. Dieser Ansatz kombiniert nahtlos die Bewegung und die erlernten semantischen Hinweise und ermöglicht ein geeignetes Verständnis der Szene.

Abstract

There is a surge in the automation of vehicles, with the prime focus on improving road safety. In general, robots are expected to operate in a continuously evolving environment and understanding the underlying causes for the changes is a key enabler to achieve the desired level of automation. The changes are intertwined in the motion and semantic characteristics of various entities of the environment. In this thesis, we propose different methods to infer these characteristics, with the final objective of holistic scene understanding using 3D LiDAR data.

Robots are often operating in non-static environments, sharing the space with various dynamic objects. For safe and efficient navigation, it is necessary to detect such objects and furthermore predict their future state. To address these challenges, we delve into the problem of estimating motion models, with the objective of understanding the dynamic characteristics, solely based on the estimated motion. To estimate motion, it is necessary to observe the same parts of the environment more than once and find an association between them. Targeting this problem, we propose a local feature descriptor learned from 3D LiDAR scans using a deep convolutional neural network. Having such a descriptor enables finding correspondences between keypoints and paves the way for estimating motion models.

In this thesis, we propose a novel method for detection and tracking of dynamic objects. In an iterative fashion, we estimate rigid motion models for various objects in the scene and then detect dynamic objects solely based on the motion. For tracking, we again utilize the motion information for finding an association between objects in consecutive scans. This method implicitly assumes a scene can be decomposed into a set of objects and to infer dynamic characteristics at a finer granular level, we propose a novel method for estimating a dense rigid motion field. This method relies on the sole assumption that objects are locally rigid and is capable of estimating arbitrary different motions for both rigid and non-rigid objects. To infer the motion state of points in a LiDAR scan, we propose a hidden Markov model based method which uses the motion field as a measurement source.

The dynamic characteristics are closely related to the semantic properties of different objects in the environment. To extract those, we propose a deep convolutional neural network for semantic segmentation of 3D LiDAR scans. The data collected by a LiDAR scanner or an other sensor is sequential, which we leverage by using a Bayes filter approach to make these semantic predictions temporally consistent. The filter utilizes the prediction of the network from the current and previous scans, thereby making the system robust to isolated incorrect predictions from

the network. To exploit the inherent relationship between motion and semantic properties, we propose a novel approach to classify points in a LiDAR scan as non-movable, movable or dynamic. This approach seamlessly combines the motion and learned semantic cues, allowing proper scene understanding.

Acknowledgments

Throughout this journey, I have had immense support and assistance from a lot of wonderful people and I take this opportunity to thank them. I would like to thank everyone in my family. Their constant trust and support is invaluable and has helped me tremendously throughout this endeavor. I would like to thank Mona Samtleben for her support, bearing with my sporadic grumpy behavior and long working hours.

I owe an immense gratitude to my PhD advisor Wolfram Burgard. I thank him for giving me an opportunity to work at Autonomous Intelligent Systems (AIS) labratory. I have learned a lot from having both scientific and non-scientific discussions with him. I thank him for his regular support and encouragement, which has allowed me to become a better researcher. I would also like to thank my Master thesis advisor K Madhava Krishna. He introduced me to interesting fields of mobile robotics and computer vision. His support and encouragement during my time at RRC gave me confidence to pursue a PhD. I would like to thank Gian Diego Tipaldi for his support when I joined AIS. I thank him for the scientific discussions for my initial research work. I would like to thank Frank Moosmann and Andreas Geiger for providing the datasets that are extensively used in this thesis. I would like to thank Micheal Keser for providing continous IT support, and Susanne Bourjaillat and Evelyn Rusdea for all the adminsitrative help.

I would like to thank Tim Caselitz for all his support during my time at AIS. I have had most interesting scientific and non-scientific discussions with him. His attention to detail is admirable. I thank him for his indispensable input to our work together and proof-reading parts of this thesis. I would like to thank Noha Radwan for being an amazing friend and office mate. I thank her for all the discussions we had and also keeping the drawer full with food for us. I would like to thanks Federico Boniardi for his help and support during our work together for SQUIRREL project. I would like to thank Tayyab Naseer and Abhinav Valada for their help, support and engaging discussions. I would like to thank Gabriel Oliveira for his contribution to our joint work. I would like to thank Daniel Büscher, Andreas Eitel, Lukas Luft, Jannik Zürn and Sudhanshu Mittal for proof-reading parts of this thesis. When I joined AIS, my German language skills was poor (it still is) and I thank everyone at AIS, who has helped me with all the bureaucratic stuff.

Working at AIS has been an amazing experience. I would like to thank everyone for providing such a supportive, encouraging and fun environment to work. I would like to thank especially Nichola Abdo, Alexander Schiotka, Andreas Wachaja,

Camilo Gordillo, Pratik Agarwal, Stefano Lucia, Benjamin Suger, Johan Vertens, Johannes Meyer, Henrich Kolkhorst, Nicolai Dorka, Niklas Wetzel, Tim Welschehold, Marina Kollmitz, Chau Do, Jingwei Zhang, Manuel Watter, Maria Hügle and Jan Wülfing.

Contents

Chapter 1

Introduction

In the last decade, the research towards self-driving cars has picked up a staggering pace. Between the years 2014 and 2017, an investment of $80 billion [33] has been made across different domains in this sector. This interest in self-driving technology has garnered equal attention from automotive giants like BMW [2], General Motors [1], and Toyota [5] and technology companies like Waymo [6] and Apple [3]. The main objective of this technology is to make our roads safer than ever before [7]. An estimated 37,461 fatalities due to crashes were recorded in 2016 on the United States roadways [80] and the goal is to decrease such numbers across the globe by introducing a high level of safe and reliable automation in the way we drive today. The extent of automation for driving a car has been divided into five different levels [2]. Lower levels of automation (level 1 and 2) has existed for quite some time, in the form of *driving assistance* like cruise control [16] or partly automated driving like lane keeping [4]. The current focus is to achieve higher levels of automation (level 3 and 4) where the driver is expected to take control in rare cases, with the final goal of having a driver-less car (level 5).

The main modules in the pipeline of achieving this automation of a mobile robotic system or can be defined as: perception, localization and mapping, planning, and control. The first module focuses on understanding the environment in which the robot is operating. The corresponding information is gathered through a variety of sensors that a robot is endowed with. Tasks like obstacle or lane detection, semantic segmentation, etc. are part of this module. Using this perceived knowledge a map of the environment can be built and the next module plans a set of actions that has to be executed. For instance, calculating a trajectory that a robot has to follow. This trajectory is then expected to comply with certain task-specific conditions like in the case of self-driving, the trajectory has to be obstacle free and assure that the vehicle adheres to the speed limit and stays within the bounds of the lane, etc. The objective of the last module is to execute the calculated plan or the trajectory. This execution happens at the actuator level, for instance, controlling the steering angle.

1.1 Scene Understanding

This thesis focuses on the first module i.e. perception, more concretely, we propose different methods for 3D scene understanding, where we primarily focus on urban outdoor environments. Scene understanding is a necessary prerequisite for a robot to operate autonomously, and uninterrupted for long periods of time. Various important tasks like localization, mapping, and obstacle avoidance, among many others, heavily rely on the perception and understanding of the environment. The objective of scene understanding is to extract meaningful compact representations from the environment and provide the necessary cues required to complete the above-mentioned tasks. Scene understanding becomes critical, especially when the robot is operating in a non-static environment, which is predominantly the case. To achieve the goal of uninterrupted autonomous operation of a robot, understanding the evolution of the environment is obligatory. The way the environment changes is intertwined in the motion and semantic cues, and also the time window in which the change is observed. If the time window is infinitesimal, the majority of the environment will be perceived unchanged and for a large time window, the opposite holds true.

If an autonomous vehicle is operating in complex scenarios, such as heavy traffic situations or crowded streets, the environment is evolving or changing at a similar rate as typical sensors are gathering data. Navigating in a safe and efficient fashion in such scenarios relies on a robust detection of dynamic objects and how reliably their future state can be predicted. Detection is challenging since such objects can belong to various semantic categories like cars, trucks, trams, bicyclists, pedestrians, etc. and the number of objects is not known apriori. In order to predict the future location of an object, an estimation of object's motion is required and, the accuracy of the estimate heavily relies on the data association between points on the object. Finding an accurate data association is challenging, especially if the same parts of the environment are not observed in two scans. This partial overlap is mainly attributed to dynamic objects and the motion of the robot itself.

Another key ingredient for successful autonomous operation is accurate localization of the robot and the accuracy largely depends on how well the sensor measurements can be associated with a pre-existing map. In the case of a non-static environment, an important challenge is to make the localization method robust towards changes in the environment. Besides enabling safe navigation, knowledge of dynamic objects can also improve the localization accuracy, as sensor measurements corresponding to dynamic objects can simply be discarded. The accuracy can potentially be further improved if more semantic knowledge can be incorporated in both mapping [93] and localization [78] methods. Parts of the environment like buildings or other man-made structures have higher chances of being useful for localization since their location does not change as frequently as a parked vehicle. To understand the distinction between non-movable and movable structures requires

extraction of semantic representations of the environment.

To extract representations for scene understanding requires perception of the environment. Currently, autonomous vehicles like self-driving cars are equipped with an array of sensors, that can include cameras (monocular or stereo), LiDARs (2D or 3D), RADARs, etc. Different sensors have their set of merits and demerits. Monocular cameras are affordable and provide rich color and texture information but do not provide any direct information about the geometrical structure of the scene. To estimate the geometrical structure using cameras, algorithms like structure-from-motion [105] are required. Furthermore, the perception ability of cameras decreases in adverse weather conditions. LiDAR scanners provide highly accurate geometrical information in the form of depth, along with surface reflectance information. Due to highly accurate measurements, state-of-the-art 2D localization system [17] and 3D perception systems [85, 127, 128, 132] heavily rely on LiDAR data. Similar to a camera, the accuracy of LiDAR measurements decreases in challenging weather conditions like fog or dust. With the surge in development in the autonomous driving sector, the cost of LiDAR scanners has decreased significantly [76] but is still not as affordable as cameras. In contrast to LiDARs, RADARs are affordable and also provide geometrical information, and unlike cameras and LiDARs, they work reliably in adverse weather conditions. Using Doppler frequency shift, location and speed of moving objects can be tracked accurately using RADARs [19], but the inferred geometrical information is not as accurate as LiDARs [126], and therefore it is not actively used for the task of semantic scene understanding. Among these sensors, 3D LiDAR data is most reliable for the task of scene understanding and therefore in this thesis, we use it as the primary data source.

1.2 Contributions

In this thesis, we propose various novel methods for 3D scene understanding to address discussed challenges. In the following, we outline the scientific contributions made through these methods and in Figure 1.1, we visually illustrate the same.

Local feature descriptor for 3D LiDAR scans In Chapter 3, we propose deep convolutional neural network architectures for learning a local feature descriptor together with a metric for matching the descriptors, and for learning a descriptor to be matched using a predefined metric. Furthermore, we tackle different problems in the descriptor learning pipeline, namely generating local surface patches around the keypoints and estimating ground-truth correspondences. We explore two different representations for surface patches. The first representation quantifies the local surface information using a 2D image and the second representation is based on 3D voxel grids. Both these representations encode depth and surface reflectance information. For our experiments, we compare the matching accuracy of our learned

feature descriptor with other handcrafted descriptors, feature descriptors learned using other network architectures. We perform an experiment for aligning multiple objects in consecutive LiDAR scans since one of the motivations for learning a descriptor is to have accurate data association between keypoints, required for aligning scans. To further analyze the discriminative power of different descriptors, we visualize the descriptors in low dimensional space and report related metrics. The learned descriptors, the dataset for learning the descriptors and the software is made publicly available.

Detection and tracking of dynamic objects In Chapter 4, we focus on inferring dynamic parts of the environment solely based on motion cues. We propose a novel model-free method for detection and tracking of dynamic objects in 3D LiDAR scans. The main contribution of this method is that it can estimate the motion of arbitrary different objects as it does not rely on prior semantic information about the objects. For detection, we utilize an iterative process, wherein every iteration we estimate the most dominant rigid motion model using RANSAC and propose a Bayesian segmentation approach to segment the points which agree with the estimated motion model. This process is repeated until all the points have a corresponding motion model. For tracking, we use the estimated motion model to associate corresponding objects. We evaluate our method on a publicly available dataset from Moosmann and Stiller [74] and compare it with their method.

Estimating dense rigid motion field Shifting our focus from object level inference to point level, in Chapter 4, we also propose a novel method to estimate dense motion field, called RigidFlow. The main contribution of this method is that it can estimate arbitrary motion models in a dynamic environment and since it only assumes objects are locally rigid it can be used for estimating the motion of non-rigid objects. We pose it as an energy minimization problem. In contrast to our previously discussed RANSAC-based method of detection and tracking of objects that requires multiple iterations, we can estimate different motion models simultaneously using this approach. We evaluate our approach on the KITTI odometry benchmark [36], a simulated dataset [129] and a dataset collected by us, consisting of pedestrians. The last dataset is used for evaluating motion for non-rigid bodies. We also incorporate the learned descriptor from the previous chapter and report results for that. The output of this method is a dense motion field and to infer the pointwise motion state using the motion field, we propose a hidden Markov model based method. This method classifies points in a scan as static or dynamic and we use the method of detection and tracking on dynamic objects as a baseline for comparison.

Semantic segmentation of a 3D LiDAR scan For proper scene understanding, inferring both motion and semantic information is necessary and so far we primarily focused on the former. In Chapter 5, we focus on extracting semantic information from the environment and combining this information with the motion cues. We

propose a deep convolutional neural network for segmentation of the LiDAR scan into the classes *car*, *pedestrian* or *bicyclist*. Our architecture is based on dense blocks [55] and to limit the parameters we use depth separable convolutions [24]. To evaluate, we use the dataset from Wu et al. [127] and compare our architecture with other state-of-the-art architectures [125, 127, 128]. To further analyze different components of our architecture, we present an ablation study. In the case of perception, the environment is often perceived through sequentially collected data and the network predicts the segmentation mask for each scan independently. To utilize the sequential nature of the data and make predictions temporally consistent we propose a Bayes filter based approach. For each class, we use a binary Bayes filter with *static* state, and static in this context means that the transition between states is unlikely, which is true for semantic classes. The filtering approach, along with current prediction from the neural network, uses the predictions from previous scans and helps in mitigating sporadic erroneous neural network predictions. Incorporating information from previous scans requires data association between points in consecutive scans and for this task we use our method of estimating pointwise motion from the previous chapter. We evaluate our approach on sequences from the KITTI tracking benchmark and use our neural network as the baseline method.

Combining motion and semantic cues for scene understanding To exploit the inherent relationship between an object's semantic class and its motion state, in Chapter 5, we also propose a hidden Markov model approach to classify points in a LiDAR scan as *non-movable*, *movable* or *dynamic*. This method neatly combines the learned semantic information with the motion cues, with the objective of a holistic understanding of the 3D environment. The distinction between non-movable and movable is learned by our proposed neural network architecture and our pointwise motion method supplies the motion cues required for separating dynamic from static. We evaluate our approach on the KITTI tracking benchmark and use our pointwise classification method from Chapter 4 as a baseline.

Publications

For the research presented in the thesis, in the following we list the corresponding publications.

- **Ayush Dewan**, Wolfram Burgard. DeepTemporalSeg: Temporally Consistent Semantic Segmentation of 3D LiDAR Scans *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2020*

- **Ayush Dewan**, Tim Caselitz, Wolfram Burgard. Learning a Local Feature Descriptor for 3D LiDAR Scans *Proceedings of the IEEE/RSJ International Conference on Robots and Systems (IROS), 2018*

- **Ayush Dewan**, Gabriel L. Oliveira, Wolfram Burgard. Deep Semantic Classification for 3D LiDAR Data *Proceedings of the IEEE/RSJ International Conference on Robots and Systems (IROS), 2017*
- **Ayush Dewan**, Tim Caselitz, Gian Diego Tipaldi, Wolfram Burgard. Rigid Scene Flow for 3D LiDAR Scans *Proceedings of the IEEE/RSJ International Conference on Robots and Systems (IROS), 2016*
- **Ayush Dewan**, Tim Caselitz, Gian Diego Tipaldi, Wolfram Burgard. Motion-based Detection and Tracking in 3D LiDAR Scans *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2016*

1.3 Collaborations

The scientific work presented in this thesis involves collaboration with other researchers. Wolfram Burgard, as the thesis supervisor has contributed through scientific discussions for all of the work. The work presented in Chapter 3 & 4 was supported by discussions with Gian Diego Tipaldi and Tim Caselitz. The research presented in Chapter 5 is based on the conference paper with Gabriel L. Oliveira in IROS, 2017. Gabriel's contribution in this paper was the neural network architecture for semantic segmentation. In Chapter 5, this architecture is replaced by an architecture proposed by the author of this thesis.

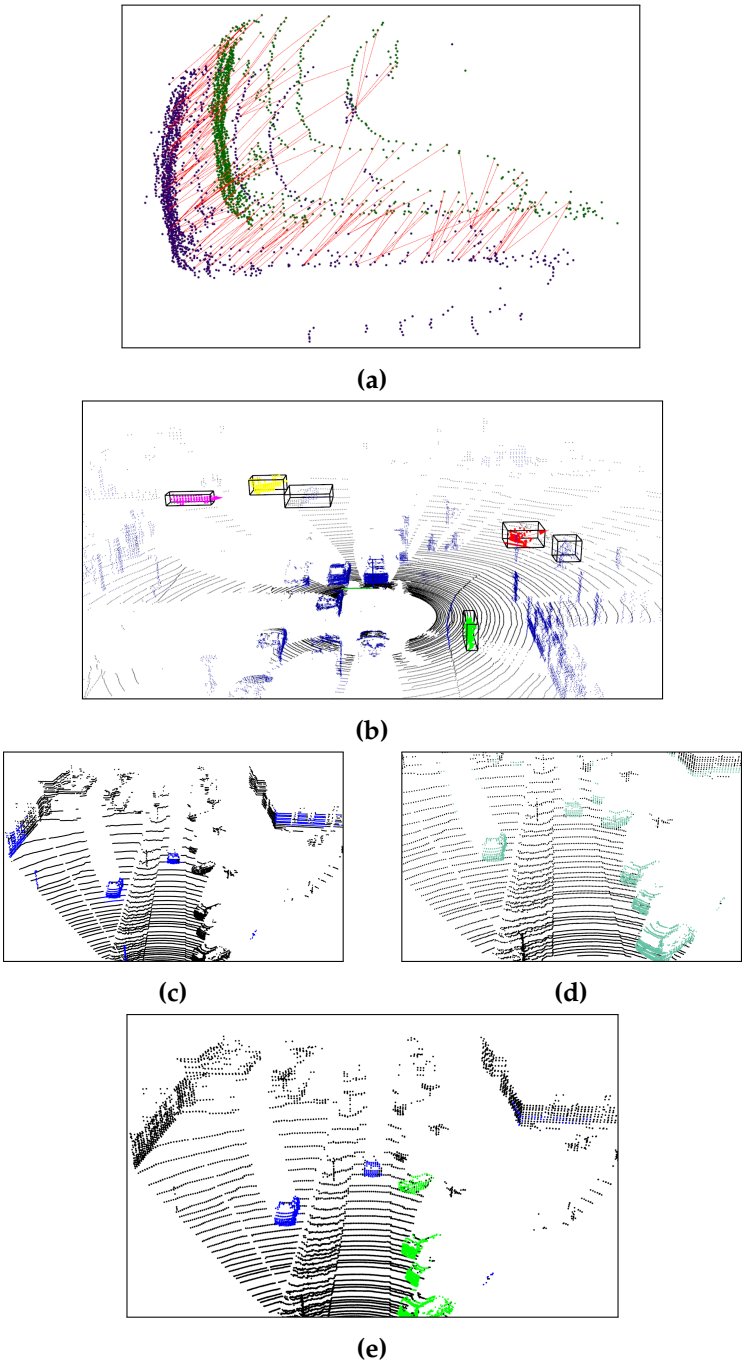


Figure 1.1: Illustration of different contributions made in this thesis. In figure (a), we illustrate the correspondences estimated using our descriptor learned together with metric. In Figure (b), we show different dynamic objects (colored points in bounding boxes) detected by our method for detection and tracking of dynamic objects. In Figure (c) & (d), we illustrate the inferred pointwise motion state (dynamic points are shown in blue color) using our method of estimating pointwise motion and the inferred semantic state using our proposed deep convolutional neural network architecture, respectively. For estimating pointwise motion, we use our proposed descriptor. The last figure shows the output of our proposed method for combining motion and semantic cues. Non-movable, movable and dynamic points are shown in color black, green, and blue, respectively. The points on the wall in figure (c) that are misclassified as dynamic are correctly classified as non-movable after incorporating the semantic information.

Chapter 2

Background

2.1 Acronyms

In the following table we list commonly used acronyms in this thesis.

Acronyms	Meaning	Description
LiDAR	**L**ight **D**etection **a**nd **R**anging	A Range Sensor
DCNN	**D**eep **C**onvolutional **N**eural **N**etwork	A deep neural network
ReLU	**R**ectified **L**inear **U**nit	An activation function used for learning non-linear decision boundaries
HMM	**H**idden **M**arkov **M**odel	A statistical Markov model
SLAM	**S**imultaneous **L**ocalization And **M**apping	A procedure for making maps
RANSAC	**Ran**dom **S**ample **C**onsensus	An iterative method to estimate parameters of a mathematical model
ICP	**I**terative **C**losest **P**oint	An algorithm for scan matching
CRF	**C**onditional **R**andom **F**ield	A method for statistical modeling

2.2 LiDAR

For a robot to navigate autonomously it heavily relies on various sensors to perceive the environment in its vicinity. Most commonly used sensors for this task of perception are cameras, LiDARs or RADARs among others. In this thesis, we use 3D LiDAR scanners as the primary sensor for solving various tasks pertaining to 3D perception.

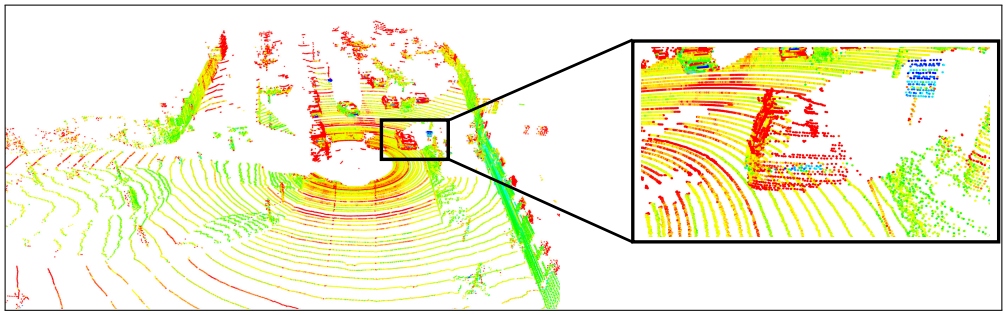


Figure 2.1: A LiDAR scan colored according to surface reflectance values collected using a Velodyne-64E scanner. 3D geometric information regarding different objects is captured neatly and precisely, thereby providing an opportunity to solve various crucial 3D perception tasks. In the zoomed in image on the right, high intensity values (color blue) correspond to the strong reflecting properties of the white number plate on a car and the signs on the side of the street. Having such discriminative information is also crucial for different perceptions tasks.

LiDAR is a *light* based sensor that operates by sending out laser light and then gathering the light which is reflected by different objects in the surrounding. The distance of an object w.r.t to the scanner is calculated by measuring the Time of Flight of the laser pulse. LiDARs not only measures the distance to the objects but also the reflected light energy from various objects. Through LiDAR scanners, precise 3D geometrical information about the environment can be retrieved readily, making a favorable sensor choice for solving important tasks like 3D object detection [36], among others.

In this thesis, we have primarily used Velodyne-64E as a LiDAR scanner. This scanner is based on Solid State Hybrid technology, involving a solid-state detector combined with a mechanical spinning system. This particular scanner has 64 lasers and has range upto 120m with 360° horizontal and 26.9° FOV. The output of the scanner is around 2.2 million points per second. In Figure 2.1, we show a single LiDAR scan collected using a Velodyne-64E as a LiDAR scanner, where the scan is colorized using the surface reflectance value.

2.3 Feedforward Neural Networks

Feedforward neural networks are the quintessential learning models. The objective of the network is to approximate a function that defines a mapping $\mathbf{y} = \mathbf{f}(\mathbf{x}; \theta)$. The network is a directed acyclic graph and the information flows only in one direction, from the input $\mathbf{x}$, through the intermediate computations defined by function $\mathbf{f}$, to the output $\mathbf{y}$. The network is composed of set of layers, where each layer is a collection of nodes. The first layer of the network is called the input layer, the last layer is called the output layer and remaining intermediate layers are called the

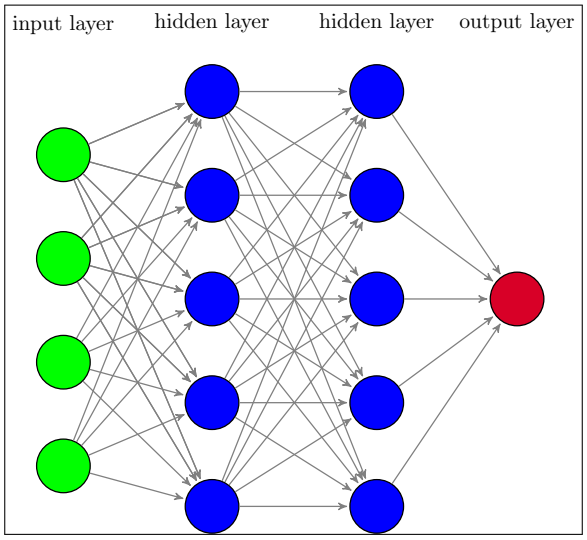


Figure 2.2: An illustration of a feedforward neural network. The network has four input nodes (green), one output node (red) and two hidden layers (blue) with 5 nodes each.

hidden layers. Figure 2.2 shows a feedforward neural network with an input layer with four nodes, a couple of hidden layers, and an output layer with a single node. The nodes in the hidden layers are called neurons and they function as the basic processing unit of a neural network. A neuron works in two steps, first it calculates a weighted sum of its input and then applies an activation function to the sum. The weights associated with inputs to the neuron is the weight vector $\mathbf{w}$. The activation value for a single neuron i in layer l is calculated as follows:

$$a_i^l = \sigma(b_i + \mathbf{w}_i^t \mathbf{x}) \tag{2.1}$$

where b_i is the bias and σ is the activation function. The network learns a linear or non-linear decision boundary depending on whether the activation function is linear or non-linear. Since the flow of information is sequential, the input to a layer l, is the output of layer $l-1$. More formally, the activation for a layer l, $\mathbf{a}^l \in \mathbb{R}^L$, where L is number of nodes in the layer, is calculated as follows:

$$\mathbf{a}^l = \sigma(\mathbf{W}^l a^{l-1} + \mathbf{b}^l) \tag{2.2}$$

where $\mathbf{W}^l$ is the weight matrix and $\mathbf{b}^l$ is the bias vector for layer l. These collection of weights and bias are the parameters θ learned by the network.

2.4 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a type of feedforward neural networks discussed briefly in the above section. In CNNs neurons are arranged arranged along

multiple dimensions and the connectivity is local, in contrast to the feedforward neural network shown in Figure 2.2, where neurons are arranged in a single dimension and a neuron is connected to all the neurons in a previous layer. In practice 2D CNNs are used most often. For 2D CNN, neurons are arranged along three dimensions: height, width and depth and the different layers within the network then transforms an input volume to an output activation volume. In this section, we primarily focus on explaining the working of a 2D CNN, therefore the term CNN is used for describing a 2D CNN, without loss of generality.

A typical CNN is comprised of convolutional, activation, pooling and dense layers. A convolution layer is the building block of a CNN. It consists of multiple learnable filters, where each filter has a small spatial resolution (height and width) but extends through the full depth of the input volume. In feedforward networks, the neurons in the layers have dense connections, i.e a neuron in layer l is connected to every neuron in the previous layer $l-1$. In contrast to that, a neuron in a convolutional layer is connected to only a small region of the input volume. For instance, if the input to the CNN is an RGB image, then a filter in the first convolution layer can have dimension of $3 \times 3 \times 3$, where the first two dimensions define the kernel size of the filter and the last dimension corresponds to depth of the input volume. The size of the kernel dictates the extent of local connectivity between a filter and the input volume. To assure a filter can learn features across the complete spatial resolution of the input volume, we slide the filters across the width and height of the input volume and calculate a 2D activation map by calculating a dot product between the values in the filter and the input volume (convolution operation). The spatial resolution of the output volume after the convolution operation is defined by the following formula.

$$w_{out} = \frac{w_{in} - k + 2 * p}{s} + 1, \tag{2.3}$$

where, w_{in} and w_{out} are input and output width, k is the kernel size, s is the stride, and p is the zero padding. Stride defines the amount of pixels by which a filter moves across the input. Height of the output volume is calculated using the same formula. In the case, where $s = 1$ and $p = \frac{k-1}{2}$, the input and output volume have same spatial resolution. The depth of the output volume is decided by the number of filters learned in a layer.

Figure 2.3 illustrates the working of the convolution layer as described above. The top figure, shows a filter convolving with the input data to generate a single feature map and the bottom figure shows how multiple feature representations can be learned within a single convolution layer. Assuming the input has dimension $h \times w \times c_{in}$, where h, w and c_{in} are height, width and the number of input channels and the size of the filter is $k \times k \times c_{in}$, the number of parameters associated with a convolution layer would be $k \times k \times c_{in} \times c_{out} + c_{out}$, where the last addition term is because of the bias. Assuming the filter moves with stride set to 1, the number of multiplication operations would be $k \times k \times c_{in} \times h \times w$. For the case

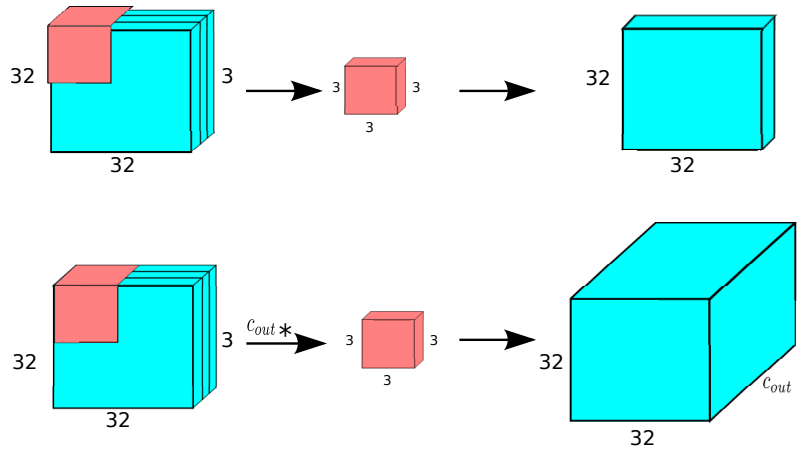


Figure 2.3: Working of the convolution layer. The figure on the top shows a filter of size $3 \times 3 \times 3$ convolving with the input data. The spatial size of the filter are the associated hyperparameters and the depth is same as the depth of the input data. As shown, each filter interacts with a small part of the input volume and to generate a single feature map, the filter slides along the spatial dimension of the input data. In order to learn multiple features, multiple filters are used and the depth of the output is equal to the number of filters used (bottom figure), which is also another hyperparameter related to this operation.

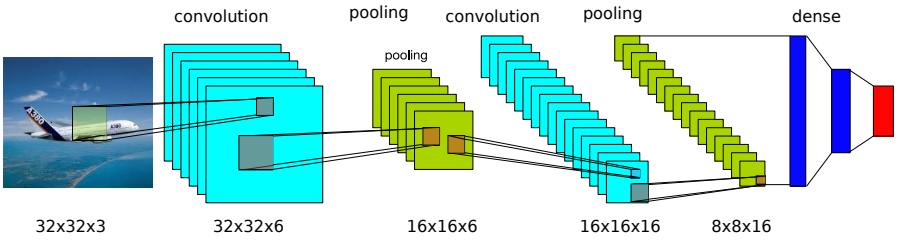


Figure 2.4: An illustration of a convolutional neural network for the task of classification. The input to the network is an RGB image, followed by a convolution layer which transforms the input volume of dimension $32 \times 32 \times 3$ into an activation volume of dimension $32 \times 32 \times 6$. The depth of the activation volume is the number of the feature maps learned by the convolution layer. Each convolution layer is followed by a pooling layer, which reduces the spatial dimension of the input volume without changing the depth. The last couple of layers are dense layers, which function in the exact same way as the hidden layers of the feedforward neural network.

illustrated in Figure 2.3, learnable parameters are $27 \times c_{out} + c_{out}$ and operations are $27 \times 32 \times 32 \times c_{out}$. Figure 2.4 illustrates a CNN for the task of classification. The architecture shows a standard arrangement of different commonly used layers in CNNs.

The values for the activation map is calculated similarly to the feedforward neural network (Eq. (2.1)), i.e. by calculating the dot product between the filter and input volume. To make sure the network can learn non-linear decision

boundaries, convolution layers are often followed by a layer implementing a non-linear activation function but unlike the convolution layer this layer does not contain learnable parameters. The most commonly used activation functions for are rectified linear unit (ReLU) and exponential linear unit (ELU), among few others. ReLU is a piecewise linear function that thresholds the activation value at zero as shown in Eq. (2.4).

$$\sigma(x) = \max(0, x) \tag{2.4}$$

Since ReLUs only have non-zero activation values only for positive input values, the neurons can potentially *die* during training. This happens when large gradients flows through the neuron, which causes a weight update that makes a neuron to be not activated again. In contrast to that, ELUs have non-zero activation for both negative and positive values as shown in Eq. (2.5) and because of this it does not suffer from *dying* neuron problem but can potentially saturate for large negative values.

$$\sigma(x) = \left\{ \begin{array}{ll} x & \text{if } x > 0 \\ \\ \exp(x) - 1 & \text{if } x \leq 0 \end{array} \right\} \tag{2.5}$$

To reduce the spatial dimensions of the feature maps and to limit the number of operations in a layer, it is a common practice to use pooling layers between convolution layers. The pooling layers operate independently on each feature map and reduces its spatial size. Similar to convolution layers, pooling layers also operate on a small region of the input volume and have kernel size and stride as parameters. A 2×2 kernel with stride 2 will reduce the spatial size of the feature map by half. Commonly used pooling layers are max pooling and average pooling. The first one will choose the max of the four values (for a 2×2 kernel) from a small region in a feature map and the latter will calculate the average of the same values. Unlike convolution layers, pooling layers do not have any learnable parameters like weights or biases. After a series of convolution and pooling layers, the last set of layers in a CNN are the dense layers which are also referred as *fully-connected* layers. Their activation value is computed by a matrix multiplication followed by a bias offset as shown in Eq. (2.2). Similar to convolution layers, an activation function is used for adding non-linearity to the output of fully-connected layers.

2.4.1 Depth Separable Convolution

In the previous section and in Figure 2.3, we explained the working of convolution layers, where a filter of small spatial dimension interacts locally with a set of input feature maps. In this case, even though the spatial size of the filter is small, it extends through the entire depth of input. In order to reduce the number of operations and the learnable parameters an alternative to standard convolution described before is

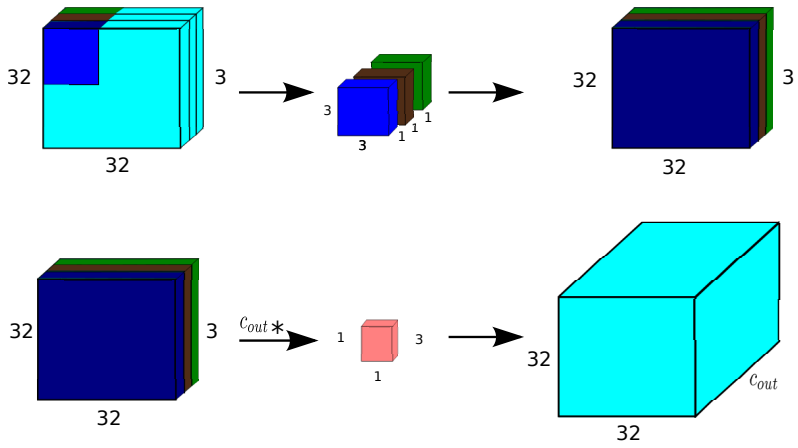


Figure 2.5: Using depth separable convolution is a two step process. In the top image we show the first step, which in involves *depthwise* convolution, where a separate filter is learned for each channel of the input data. In this illustration we learn features from each input channel independently using three different convolution filter and the depth of the input and output is same. To learn different feature representations, in the second step, we perform *pointwise* convolution on the learned feature maps from the previous steps using c_{out} filters of spatial dimension $1 \times 1 \times 3$.

depth separable convolution [24] [52]. Instead of interacting with the entire depth of the input through a single filter, we perform *depthwise* convolution where a filter is learned separately for each channel of the input. The top image in Figure 2.5, illustrates this operation. Since the number of input and output channels are same for *depthwise* convolution, in order to learn multiple feature representation, *depthwise* convolution is followed by *pointwise* convolution. This is a standard convolution described in the previous section, where the depth of the filter is same as the number of input channels but now the spatial dimension of the filter is 1×1. The *pointwise* convolution is illustrated in the bottom image of Figure 2.5. For both standard and depth separable (depthwise + pointwise) convolution, we transform the input volume of dimension $h \times w \times c_{in}$ to an output volume of $h \times w \times c_{out}$ but the number of operations and the associated learnable parameters are different. For *depthwise* and *pointwise* convolution the number of parameters are $k \times k \times c_{in}$ and $c_{in} \times c_{out}$ respectively and therefore the total parameters are $k \times k \times c_{in} + c_{in} \times c_{out}$, which are signficantly lower than the parameters associated with a standard convolution ($k \times k \times c_{in} \times c_{out}$). Comparing the cases illustrated in Figure 2.3 and 2.5, the number of parameters has been reduced from $27 \times c_{out}$ to $27 + 3 \times c_{out}$. Similarly the number of operations has been reduced from $32 \times 32 \times 27 \times c_{out}$ to $32 \times 32 \times (27 + 3 \times c_{out})$.

2.4.2 Dense Blocks

The CNN discussed above comprises of a set of layers, where input to one layer are

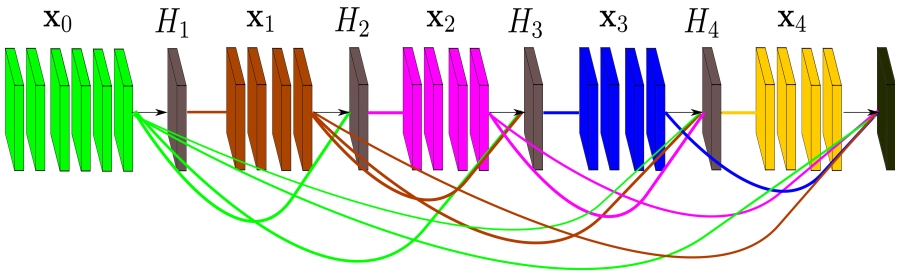


Figure 2.6: An illustration of working of a dense block. The input to the block is $\mathbf{x}_0$, it has four layers ($H_1 - H_4$) implemented within the composite function H_l. The growth rate i.e. feature maps learned by each layer is set to four. In contrast to a standard CNN, where input to a layer is the feature maps learned from the previous layer, in dense blocks the input is feature maps learned from all the previous layers. For instance, input to H_4 is all the feature maps from $\mathbf{x}_0$ to $\mathbf{x}_3$. The last layer is the transition layer which can involve operations like pooling.

the learned feature maps from the previous layer as shown in equation below.

$$\mathbf{x}_l = H_l(\mathbf{x}_{l-1}) \tag{2.6}$$

Here, $H_l(\cdot)$ is a composite function of different operations like convolution, non-linear activation among others. A modern day CNN is comprised of large number of layers, thereby making networks *deep*. These deep networks are capable of learning complex non-linear functions but can be notorious to train because of a problem of *vanishing gradients*. These networks are trained using gradient based methods and the gradients of loss w.r.t to the weights are back-propagated from the output layer to all the hidden layers. The magnitude of the gradients tends to get smaller as the information flows backward in the network, thereby causing earlier layers to learn slowly. In feedforward networks, the flow of information is from earlier layers to subsequent layers and if the features learned by earlier layers are inaccurate, the performance of the network is sub-optimal.

To overcome the problem of vanishing gradients, Huang et al. [54] proposed *dense* blocks. In contrast to using only the output from preceding layer as input, in dense blocks the input to a layer are the feature maps learned by all the previous layers as shown in Eq. (2.7).

$$\mathbf{x}_l = H_l([\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{l-2}, \mathbf{x}_{l-1}]) \tag{2.7}$$

This connectivity pattern allows improved flow of information and enables efficient training since each layer has direct access to gradients from the loss function, thereby alleviating the problem of vanishing gradients. Dense blocks also curtails learning of redundant features since each layer has access to all the feature maps learned from previous layers, which allows proper *feature reuse*.

The number of feature maps learned by each layer is called the *growth rate*. Each dense block often contain multiple *repetitions* of the composite function $H_l(\cdot)$ and different dense blocks within a CNN are separated by *transition* layers, which includes operations like pooling. Figure 2.6 illustrates working of a dense block, with growth rate and repetitions both set to four.

Different CNN architectures proposed in this thesis are based on dense blocks. The exact details of the composite function and parameters pertaining to dense blocks like growth rate or number of repetitions are discussed in subsequent chapters.

2.4.3 Weight Initialization

To begin the training the process, the first step is to initialize the value of different learnable parameters and in this thesis, for different CNNs, we use Xavier initialization [38] as a weight initialization method.

2.4.3.1 Xavier Initialization

Weight initialization is crucial for proper training of a neural network. If the initial value for weights are small then the signal propagating through different neurons in consecutive layers diminishes and in the case of large initial values the value for weights increases quickly. To avoid the signal from either diminishing or exploding, in Xavier initialization method, value of weights are sampled using a normal distribution with zero mean and variance $\frac{2}{n_{in}+n_{out}}$, where n_{in} and n_{out} are the input and output signals for a particular neuron. This particular choice of variance assures the variance of the input signal and the output signal are same.

2.4.4 Regularization

The models learned by different learning algorithms are often plagued by the problem of *overfitting*, where the learned parameters, models the training data too well and fails to generalize for examples which were not part of the training set. In case a learned model overfits, the gap between the performance of the model on the training set and the validation or the test set is significantly large. In this thesis we use the following techniques to prevent different learned models from overfitting.

2.4.4.1 L2 Regularization

This regularization involves adding a term in the loss function to prevent learned parameters from having large magnitude. The cost function for minimizing the negative log likelihood in Eq. (2.13) can be modified in the following way for

enforcing l2 regularization.

$$L(\theta) = -\sum_{j=1}^{N} \log(P(\mathbf{y}^j \mid \mathbf{x}^j; \theta)) + \frac{\eta}{2}\|\theta^2\| \tag{2.8}$$

The parameter η regulates penalty on the magnitude of the weights. A large value of η will force the weights to have small values and the learned parameters will be unable to model examples in the training set, i.e., the learned model will be *underfitting*. Conversely, choosing value of η to be close to zero can cause a learned model to potentially overfit. An optimal value of η can vary across different learning tasks and is one of many different parameters that needs to be tuned for proper learning of weights.

2.4.4.2 Dropout

One of the most commonly used techniques for preventing overfitting in neural network based methods is *dropout*. It involves randomly ignoring the contribution certain neurons during training. By ignoring the contribution we mean that these randomly selected neurons are not considered during forward and backward pass. More formally, at every training iteration each neuron is kept with probability p or dropped with probability $1-p$.

Dropout enforces regularization by controlling the co-dependency between learning parameters, thereby allowing individual neurons to fully exploit their learning capability. In every iteration only a sub-network of the original neural network is used, which enables learning of multiple feature representation, thereby enhancing the generalization capability of the neural network. Similar to l2 regularization, an optimal value of p varies across different learning task and therefore it needs to be tuned along with other parameters.

2.4.5 Loss Functions

In this thesis, we use different loss functions for training CNNs for various tasks. We use softmax cross-entropy for the purpose of classifying points in a LiDAR scan, and for combined learning of a feature descriptor and metric for matching the descriptor. We use hinge embedding loss for learning a feature descriptor which is discriminative in Euclidean space.

2.4.5.1 Softmax Cross-Entropy Loss

Cross-entropy indicates the difference between the *true* distribution of labels $\mathbf{y} \in \mathbb{R}^C$ and distribution of the predicted labels $\hat{\mathbf{y}} \in \mathbb{R}^C$ as shown in Eq. (2.9). Here C denotes the number of classes.

$$\boldsymbol{\mathcal{L}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{C} y_i . \log \hat{y}_i \tag{2.9}$$

By minimizing the cross-entropy, we are trying to minimize the number of bits required to encode the true distribution of labels $\mathbf{y}$ using the distribution of predicted labels $\hat{\mathbf{y}}$. For the distribution of labels $\mathbf{y}$, all the probability mass is at the correct class, i.e., $\mathbf{y} = [0, 0, \ldots, 1, \ldots, 0]$. This particular representation used for the distribution for ground-truth labels is called one-hot encoding. To estimate the distribution of the predicted labels we use the Softmax function $\mathbf{f}(\mathbf{z})$. This function takes a real valued vector $\mathbf{z}$ and normalizes it to a vector with values between zero and one that sum to one. $\mathbf{z}$ is the output of the last layer of CNN and has the same dimension as the number of classes. The softmax score for label i is calculated as following:

$$\mathbf{f_i}(\mathbf{z}) = \frac{\exp(z_i)}{\sum_k^C \exp(z_k)} \tag{2.10}$$

The loss in Eq. (2.9) is for a single training example. The loss for all the training examples is summation of loss of individual training examples.

$$\boldsymbol{\mathcal{L}}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{N}\sum_{i=1}^{C} y_i^j \log \hat{y}_i^j \tag{2.11}$$

Considering a probabilistic interpretation, the minimization of softmax cross-entropy loss is same as maximizing the likelihood of the prediction. The likelihood function L is given as:

$$L(\theta) = P(\mathbf{y} \mid \mathbf{x}; \theta) \tag{2.12}$$

Maximizing the likelihood is equivalent to minimizing the log negative likelihood and therefore Eq. (2.12) can be written as:

$$L(\theta) = -\sum_{j=1}^{N} \log(P(\mathbf{y}^j \mid \mathbf{x}^j; \theta)) \tag{2.13}$$

The probability distribution $P(\mathbf{y}^j \mid \mathbf{x}^j; \theta)$ can be estimated using the softmax function $\mathbf{f}(z)$.

$$P(\mathbf{y}^j \mid \mathbf{x}^j; \theta) = \frac{\exp(\mathbf{z})}{\sum_k^c \exp(z_k)} \tag{2.14}$$

Therefore Eq. (2.13) can be written as following:

$$L(\theta) = -\sum_{j=1}^{N}\sum_{j=1}^{C} y_i^j \log(\hat{y}_i^j) \tag{2.15}$$

The likelihood of an example j belonging to a class i is given by $\hat{\mathbf{y}}_i^j$.

2.4.5.2 Hinge Embedding Loss

In this thesis, we propose two different local feature descriptors. The first descriptor is learned simultaneously with a metric for matching the descriptor and the descriptors for corresponding keypoints that are close to each other in the learned metric space. For this task, we use the softmax cross-entropy loss discussed above. The second descriptor is learned using Euclidean metric i.e the descriptors for corresponding keypoings are similar in the Euclidean space. For this task we use the hinge embedding loss shown below.

$$\mathcal{L}(\mathbf{d_1}, \mathbf{d_2}) = \left\{ \begin{array}{ll} \|\mathbf{d_1} - \mathbf{d_2}\|_2 & \text{if } p_1 = p_2 \\ \\ \max(0, c - \|\mathbf{d_1} - \mathbf{d_2}\|_2) & \text{if } p_1 \neq p_2 \end{array} \right\} \tag{2.16}$$

In the case when patches p_1 and p_2 belong to corresponding points, a large distance between the learned feature descriptors $\mathbf{d_1}$ and $\mathbf{d_2}$ is penalized. For the case of non-matching keypoints, we want distance between the descriptors to be greater than c. The maximum loss c is incurred when distance between descriptors is zero and if distance is greater than c, then the loss is zero.

2.5 Evaluation Metrics

To evaluate various methods proposed in this thesis we use different metrics. For evaluating performance of object detection and classification we use Precision, Recall and F1-score. Precision is calculated as following:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.17}$$

where, TP and FP represents the number of true positives and false positives, respectively. Recall is calculated as following:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.18}$$

where, TN represents the number of true negatives. Precision and recall values are calculated for different confidence values associated with the prediction of the classifier. Using low confidence value as a threshold for classification, results in large recall but low precision. Conversely a high confidence value as threshold results in high precision and low recall. Choosing a threshold value for confidence score is often task dependent. In order to evaluate the performance of a classifier using a single score rather than precision or recall, a common practice is to report F_1 score, which is harmonic mean of precision and recall and it is is calculated as following:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.19}$$

These metrics are calculated for all the classes individually.

For evaluating semantic segmentation we use intersection over union (IoU) as a metric and it is calculated as following:

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{2.20}$$

For the evaluating the performance of learned feature descriptors, we plot the ROC (receiver operating characteristic) curve. This curve summarizes the performance of a classifier by combining the true positive rate (TPR) and the false positive rate (FPR). The TPR is calculated in the same way as recall (Eq. (2.18)) and the FPR is calculated as following:

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{2.21}$$

Similar to precision and recall, TPR and FPR are also calculated for different values of the confidence score of a classifier's prediction.

Chapter 3

Learning a Feature Descriptor for 3D LiDAR scans

Robust data association is necessary for virtually every SLAM system and finding corresponding points is typically a preprocessing step for scan alignment algorithms. Traditionally, handcrafted feature descriptors were used for these problems but recently learned descriptors have shown to perform more robustly. In this chapter, we focus on learning local feature descriptors for 3D LiDAR scans. The descriptor is learned using a Convolutional Neural Network (CNN). We discuss two methods for learning a feature descriptor. In the first method the descriptor is learned together with a metric for matching the descriptor and in the second method the descriptor is learned to be discriminative in a predefined metric. Our proposed architecture consists of two separate sub-networks, a Siamese network for learning a feature descriptor and a metric learning network for matching the descriptors. We also present methods for estimating local surface patches and obtaining ground-truth correspondences, required for learning the descriptor. In extensive experiments, we compare our learned feature descriptor with a set of handcrafted and learned local descriptors and report highly competitive results for multiple experiments in terms of matching accuracy and computation time.

3.1 Introduction

For several robotics tasks, it is required to have robust data association in order to match similar parts of the environment under different conditions. Estimating data association is always an important step in SLAM systems [97] and methods for lifelong visual localization [77] rely on finding corresponding points between scenes captured in different seasons. Furthermore, knowing corresponding points is also a requirement for scan alignment. In this thesis, one of our objectives was to

infer motion cues from the environment, which involves detecting dynamic objects in the environment using motion and estimating rigid motion field. A prerequisite for these tasks is to estimate data association between keypoints in consecutive scans and we address this problem in this chapter and propose different methods for learning local feature descriptors for 3D LiDAR scans.

The majority of existing feature descriptors for 3D data are handcrafted [9] and rely either on quantifying surface normals or the curvature around keypoints. In contrast to these methods, we do not try to explicitly extract geometric information but instead use raw scan data and learn descriptors using local shape information and surface reflectance values, around a keypoint. In the context of learning feature descriptors, various CNN-based methods have been proposed for learning feature descriptors for image patches [12, 42, 130] and dense 3D surface patches [131]. All of these methods include a two or multi-stream Siamese network for learning a feature descriptor, which is either discriminative in a predefined [12, 130, 131] or in a learned metric [42].

In this chapter, we propose a method to learn a feature descriptor simultaneously with a metric for matching the descriptors, and a method for learning a descriptor that is matched using a predefined metric. Our proposed architecture consists of two sub-networks. A two stream Siamese network for learning a feature descriptor and a network for learning a metric for matching the descriptors. Our feature learning network is based on the recent dense blocks architecture [55]; for metric learning, we use a stack of fully connected layers. In the first method, the metric learning network is appended at the end of feature learning network and both networks are trained together. Figure 3.1 demonstrates the matching of keypoints using our learned feature descriptor and metric for sparse 3D LiDAR scans. For the second method, only the feature learning network is used for training, and learned descriptors are discriminative in Euclidean metric, rather than in a learned metric.

The input to the network is a pair of surface patches around keypoints, where the patches capture the local surface information around keypoints. We generated our own training data, since the data used by existing learning based methods either consists of grayscale image patches or dense 3D surface patches and neither of these can be used for learning a descriptor for sparse LiDAR scans. We explore two different representations for the input surface patches. In the first case, we represent the 3D neighborhood around a keypoint as a 2D image patch and in the second case, we represent the same neighborhood using 3D voxels. In both cases, we encode the depth information and surface reflectance values. The training data is generated using the LiDAR scans in the KITTI tracking benchmark [36], and ground-truth correspondences are obtained by tracking keypoints using our method of estimating pointwise motion as discussed in Chapter 4.

The foremost contribution of this chapter are different methods for learning local feature descriptors for sparse 3D LiDAR scans. The learned descriptors can either

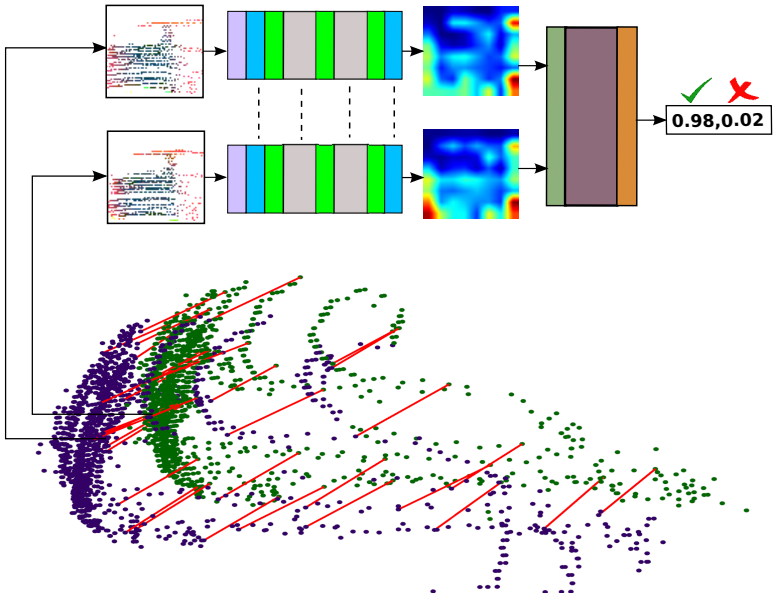


Figure 3.1: An illustration of keypoint matching using our learned feature descriptor. Surface patches around the keypoints are passed through the feature learning network to estimate feature descriptors. These descriptors are then passed through the metric learning network to estimate a matching score. Red lines show the correspondences between the keypoints in the two 3D LiDAR scans. Different colors in the architecture represent different layers, which are explained in later sections.

be matched using a learned metric or a predefined metric. The learned descriptor allows robust matching of keypoints, which is necessary for estimating motion. We also target relevant problems in the feature learning pipeline, i.e., extracting local surface patches and obtaining the ground-truth correspondences. In the context of local surface patches, we explore different representations, as well.

To validate the performance of our learned feature descriptors, we evaluate the matching accuracy (Section 3.3.1) and compare the performance with multiple hand-crafted feature descriptors and descriptors learned with different CNN architectures. We present results for another experiment, where we align multiple objects based on feature correspondences (Section 3.3.2). Furthermore, to highlight the difference between using a predefined and a learned metric, we present comparative results for our learned feature descriptor, using respective cases. Similarly, we compare the feature descriptor, learned using 2D patches and 3D voxels, as input data. In addition, we show that our descriptor can generalize to data from different type of LiDAR scanners. We do this by repeating the alignment experiment but with data from a different scanner. Finally, we present an ablation study (Section 3.3.4) to provide insight into the role of each modality used for learning the descriptor. To analyze the discriminative nature of a descriptor, we measure similarity between

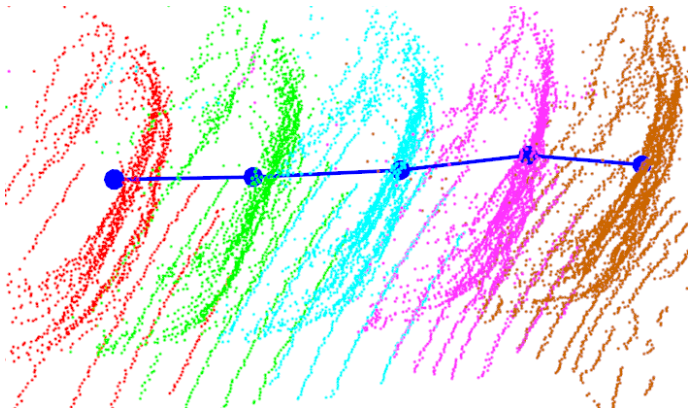


Figure 3.2: Illustration of keypoint tracking used to generate image patches for training.

the descriptors belonging to a set of matching and non-matching keypoint pairs (Section 3.3.5). We then compare this analysis across different handcrafted and learned feature descriptors.

3.2 Learning a Local Feature Descriptor

Our feature learning pipeline has three main steps. First, we extract keypoints and track them to obtain ground-truth correspondences. Then we extract surface patches around successfully tracked keypoints and in the last step we train the network to learn a feature descriptor.

3.2.1 Generating Training Data

A key requirement for supervised learning is labeled training data. Since labeling many correspondences by hand is a strenuous task, existing methods [12, 42, 130, 131] use 3D scene reconstruction for associating pixels corresponding to same 3D point for obtaining ground-truth correspondences. Using datasets from these methods is not possible, since our objective is to learn a feature descriptor for sparse 3D LiDAR data and the datasets made available from these methods either consists of grayscale image patches or dense 3D surface patches. Therefore, we create our own training data.

3.2.1.1 Ground-Truth Correspondences

To obtain ground-truth correspondences, we first select the keypoints using uniform sampling and then track those keypoints for the next five frames. For tracking, we use our method for estimating pointwise motion. Associating keypoints over multiple frames instead of one allows us to remove false correspondences. For

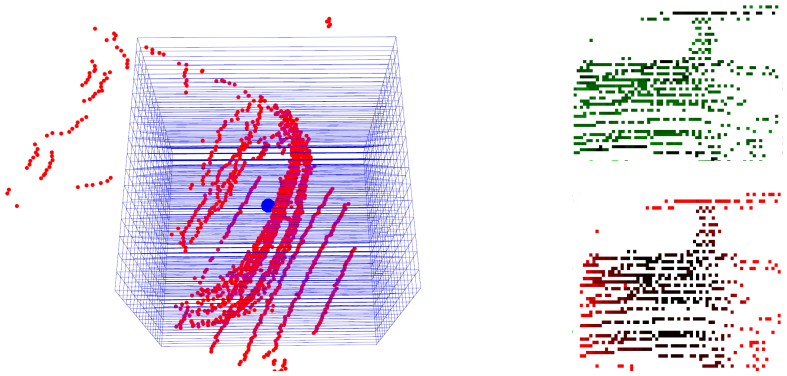


Figure 3.3: The left image shows a voxelized cube around a keypoint and the right image shows different channels of an extracted surface patch. Surface reflectance intensity is showed in green and depth is shown in red.

keypoints that are successfully tracked over five frames, we extract surface patches for all frames. Figure 3.2 shows tracking of a keypoint for five frames using our method of estimating of the pointwise motion.

3.2.1.2 Training Patches

Learning a feature descriptor requires local surface patches around keypoints. In the case of 2D image data, generating patches is a straightforward task, since the data is organized in a grid structure, but for unorganized sparse 3D pointclouds this task is non-trivial. In our approach, for a given keypoint, we use a cube with predefined length to capture the local neighborhood by choosing points inside the cube. We propose the following two different representations for these points:

1. We use a 2D image patch for representing points around a keypoint. The cube around the keypoint is divided into $64 \times 64 \times 1$ voxels. For every voxel we calculate the average distance w.r.t. the keypoint and the average surface reflectance intensity values for the points inside the voxel and store the 3D voxel as a two channel image patch ($64 \times 64 \times 2$). The first modality (depth) aims at capturing the geometry and the second (intensity) captures surface reflectance properties. Figure 3.3 illustrates this process, where the left image shows the voxel structure around a keypoint and the right image shows the modalities that we use.

2. We use 3D voxels for representing local surface information. The first step is to divide the cube into $24 \times 24 \times 24$ voxels and calculate the inverse sign distance function (ISDF) and the intensity value for each voxel. To calculate the ISDF value, we first calculate the distance between the voxel center and the nearest surface point. To estimate whether the voxel is inside or outside the

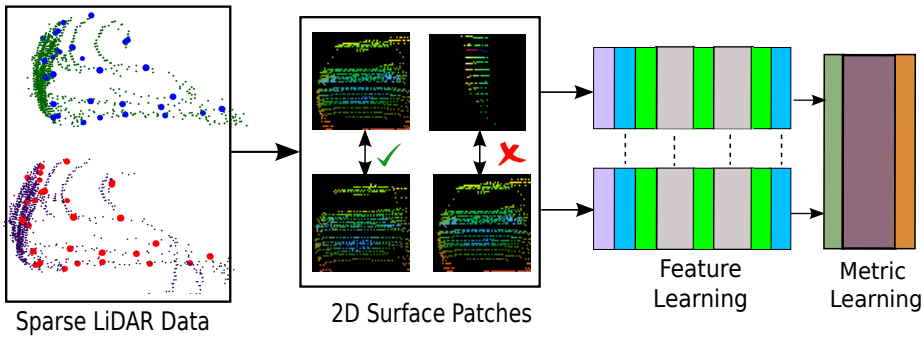


Figure 3.4: An overview of our feature learning method. The local information around the keypoints from sparse LiDAR scans is converted into surface patches. Input to the feature learning network is a pair of matching and a pair of non-matching surface patches and input to the metric learning network is the learned feature descriptors.

surface, we compute the angle between the normal of the nearest surface point and its vector to the voxel center. If the angle is less than 90°, the voxel center is outside or inside otherwise. The points outside the surface have a positive sign and the points inside the surface have a negative sign. The intensity value corresponding to each voxel is the intensity value of the nearest surface point. Using 3D voxels allow us to retain more information in contrast to the first case where a cube is being represented as a 2D image patch. Furthermore, the 3D voxel representation is denser in comparison to 2D image patches because the depth and intensity information is also available for empty voxels.

3.2.2 Network Architecture

Figure 3.4 illustrates the process of learning a feature descriptor. The input to the network is a pair of matching and non-matching surface patches, where the patches could either be 2D image patches or 3D voxels, as discussed above. Our proposed learning framework consists of a Siamese network for learning the features and a metric learning network. Siamese networks are a popular architecture choice for learning problems that involve finding similarity or relationships between comparable quantities. Therefore, they are perfectly suited for our task of learning a feature descriptor for matching keypoints. We use a two stream Siamese network, where each stream is an identical sub-network consisting of two convolution layers followed by two dense blocks and a bottleneck layer. All learnable parameters within each layer are shared across both streams.

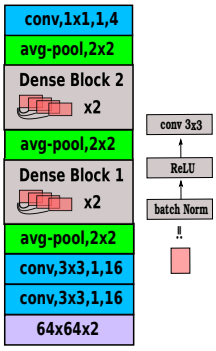
Figure 3.5a shows the details of the architecture used for learning the descriptor from 2D image patches. For each layer, the kernel size, stride and the output feature maps are shown in the figure. In the proposed architecture, the composite function $H_l(\cdot)$ for a dense block consists of batch-normalization, a Rectified Linear Unit

(ReLU), and a convolution layer. Within each block, the composite function is repeated twice. The bottleneck layer is a convolution layer and the flattened output of it is the learned feature descriptor. We use three average pooling layers: one after the second convolution layer and one after each dense block. In the case of learning a descriptor from 3D voxels, the architecture (Figure 3.5a) is similar to the architecture used for 2D patches with some necessary changes. Due to the increased dimensionality of the input data, the kernel size for convolution and pooling operations are changed to $3 \times 3 \times 3$ and $2 \times 2 \times 2$ respectively. The number of feature maps for the bottleneck layer is increased from 4 to 8. Since the spatial size of the 3D voxel is smaller in comparison to the 2D patches ($24 \times 24 \times 24$ and 64×64), the spatial size of the input tensor to bottleneck layer is also smaller ($3 \times 3 \times 3$ and 8×8). The increase in feature maps makes the sizes of the feature descriptors learned from 2D image patches (256) and 3D voxels (216) comparable.

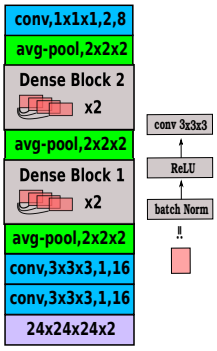
3.2.2.1 Simultaneous Feature Descriptor and Metric Learning

For simultaneous feature and metric learning, we append the feature learning network with a metric learning module. This module has five fully connected layers (*fc*) where every *fc* layer, except the last one (*fc4*), is followed by a ReLU. The input to the metric learning module is the concatenation of the output of respective Siamese network streams. The number of feature maps for every layer is shown in Figure 3.5c. The metric learning architecture is identical for both 2D patches and the 3D voxels except for the number of feature maps in the input layer. In case of the 2D patches the size of input layer is 256×2 (Figure 3.5c) and in the other case it is 216×2.

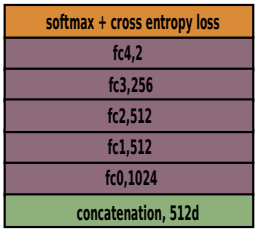
We pose the task of simultaneous feature and metric learning as a binary classification problem, where a pair of input surface patches has to be classified as matching or non-matching. Our training set is $\mathcal{T} = \{(\mathbf{X}_n^1, \mathbf{X}_n^2, \mathbf{Y}_n),\ n = 1, \ldots, N\}$, where $\mathbf{X_n^1}$ and $\mathbf{X_n^2}$ are two sets of surface patches and $\mathbf{Y_n} = \{\mathbf{y_k} \in \{1, 0\},\ k = 1, \ldots, N\}$ are the corresponding ground truth labels. When the input is a pair of 2D image patches, the function learned by the CNN is defined as $\mathbf{f}(\mathbf{x_k^1}, \mathbf{x_k^2}, \theta) \in \mathbb{R}^{64\times64\times2} \rightarrow \mathbb{R}^2$, where θ are the parameters of our model and $\mathbf{x_k^1} \in \mathbf{X_n^1}$ and $\mathbf{x_k^2} \in \mathbf{X_n^2}$ is a pair of 2D image patches. This function first maps a pair of image patches onto a pair of feature descriptors and then classifies the estimated feature descriptors as either matching or non-matching. The network learns the weights θ by minimizing the cross-entropy (softmax) loss in Eq. (3.1) over all patch pairs, as shown in Eq. (3.2). For the 3D voxels, the problem is exactly the same except that the function mapping changes to $\mathbf{f}(\mathbf{x_k^1}, \mathbf{x_k^2}, \theta) \in \mathbb{R}^{24\times24\times24\times2} \rightarrow \mathbb{R}^2$.



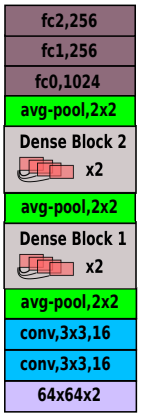
(a) Architecture for feature learning network using 2D image patches



(b) Architecture for feature learning network using 3D voxels



(c) Architecture for metric learning network



(d) Architecture for feature learning network for Hinge Embedding Loss

Figure 3.5: Different DCNN architecture for learning the feature descriptors. We use the architectures shown in in figures (a) & (c) for simultaneously learning a feature descriptor and a metric for matching the descriptors from 2D image patches. Similarly, we use the architecture shown in figures (b) & (c) for learning a descriptor and a metric together from 3D voxels. In figure (d), we show The DCNN architecture used for learning a descriptor using a pre-defined metric.

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c \in \{0,1\}} y_c \log \hat{y}_c \tag{3.1}$$

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{k=1}^{N} \mathcal{L}\left(\mathbf{y_k}, \mathbf{f}\left(\mathbf{x_k^1}, \mathbf{x_k^2}, \theta\right)\right) \tag{3.2}$$

3.2.2.2 Learning a Feature Descriptor using a Predefined Metric

An alternative to learning the metric is to use a loss layer directly after the feature learning. The most commonly used loss layers for the task of learning similarity or dissimilarity between comparable quantities are contrastive loss [131] and hinge embedding loss [102]. Both of these layers try to minimize the Euclidean distance between matching descriptor pairs and simultaneously increase the margin between non-matching pairs. In our case, we use the hinge embedding (H.E.) loss.

As before, the training set is defined as $\mathcal{T}$ but the function learned by the CNN is now $\mathbf{g}(\mathbf{x_k}, \theta) \in \mathbb{R}^{64\times64\times2} \rightarrow \mathbb{R}^d$, where $\mathbf{x_k}$ is a surface patch around the keypoint and d is the size of the learned descriptor. In the case of metric learning described in Section 3.2.2.1, the network simultaneously learned a mapping from an input surface patch to a feature descriptor and to classify whether a pair of descriptors are matching or non-matching. Here, the network only learns a function to map an input surface patch to a feature descriptor that is discriminative in Euclidean space.

For learning a feature descriptor using H.E. loss, we replace the bottleneck layer in the feature learning network with a set of fully connected layers. The output of the last layer is the learned feature descriptor. The modified architecture for the network used for learning from 2D image patches is shown in the Figure 3.5d. The layers prior to the fully connected layers are exactly the same as shown in Figure 3.5a.

3.2.3 Training

Our complete network architecture has been implemented in TensorFlow [8]. Using our patch generation method, we generated 58,710 surface patches for training. The input to the network is a batch of surface patch pairs. The surface patches could be either 2D image patches or 3D voxels. Each batch consists of an equal number of matching and non-matching surface patches, as shown in Figure 3.4. For generating surface patches for training, we used the first ten sequences of the KITTI tracking benchmark. With these surface patches, we estimated 117,400 positive and 704,400 negative pairs. To generate negative pairs, for every surface patch we randomly selected surface patches corresponding to keypoints sampled from a different sequence. Since the input to the network always consists of a negative and positive combination, our effective training set consists of 704,400 samples. We use the remaining sequences from the benchmark for generating surface patches that are part of the test set.

We use the following training procedure for learning a feature descriptor, with and without metric. We train the network with a batch size of 32 and use the Adam optimizer [62] with a learning rate of $1e^{-4}$ and weight decay $1e^{-5}$. The growth rate for dense blocks is 4. We train the network for 5 epochs and the complete training process required around 2 hours using an NVIDIA GeForce GTX 980 graphics card.

3.3 Results

To evaluate our method, we perform multiple experiments. We first evaluate the matching accuracy for various descriptors. We then report the average alignment errors for objects scanned using Velodyne HDL-64E and HDL-32E LiDAR scanners. We also report computation times for calculating and matching the descriptors. Among handcrafted descriptors, we compare with SHOT [112], FPFH [95], and 3DSC [35]. To justify the usage of dense blocks for our task, we present results for feature descriptors learned with the following two network architectures: [42, 46].

1. The first architecture [42] was proposed to learn a feature descriptor and a metric for grayscale image patches. Its feature learning architecture consists of blocks of convolution layers and ReLUs separated by max-pooling layers and it uses a stack of fully connected layers for metric learning. This feature learning architecture is similar to the initially proposed CNN architectures, for instance VGG [103]. Other methods [130, 131] proposed for learning feature descriptors have also used similar architectures. The reason we chose the architecture from MatchNet is that it has a similar input patch size to ours (64×64) and also uses metric learning.

2. The second architecture that we compare with consists of residual blocks [46]. Lately, residual blocks-based architectures have also been shown to perform well for a variety of tasks [46, 69]. We use a ResNet-8 [69] for feature learning and keep the metric learning architecture unchanged. Figure 3.6 shows the ResNet-8 architecture that we used.

Furthermore, we also compare with our feature learning network trained with the hinge embedding (H.E.) loss and the feature descriptor learned using 3D voxels. The first three comparisons present the advantages of learned feature descriptors over the handcrafted descriptors. The next two comparisons highlight the benefits of using dense blocks. The last two compare the performance of a learned with a predefined metric and 2D image patches with 3D voxels.

3.3.1 Matching Accuracy

The goal of this experiment is to test the performance of different descriptors on the pairs of surface patches from our testing set. We use 100,000 samples, half

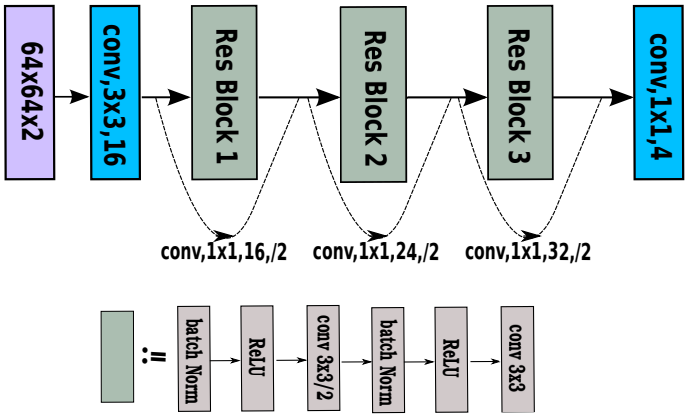


Figure 3.6: Architecture for ResNet-8

of which are matching and the other half are non-matching. For every case we plot a receiver operating characteristic (ROC) curve and report false-positive rate at 95% recall (FPR95). The ROC curves are shown in Figure 3.7 and the FPR95 is reported in Table 3.1. We plotted the curve for various matching thresholds, which in the case of metric learning is the softmax score. We used the Euclidean distance between descriptors for the handcrafted descriptors and the descriptor learned using H.E. loss. The error for our feature descriptor learned from 3D voxel is the lowest and it outperforms the handcrafted descriptors and other learned descriptors by a significant margin. This supports our initial claim that the 3D voxel representation allows the capturing of more information of the local surface, thereby enabling learning of a more discriminative local feature descriptor. This superior performance comes at the cost of large computation time for feature estimation. Our descriptor learned together with a metric that used 2D image patches was the next best performing feature descriptor, which outperformed other descriptors learned from same patch representation. The superior performance of learned over handcrafted descriptors shows the advantages of learning a descriptor. These results also justify our proposed architecture for learning a descriptor.

Comparing the descriptors with simultaneously learned and predefined metrics, the error increases when our feature learning network is trained with H.E. loss (the yellow curve in Figure 3.7), demonstrating the importance of metric learning. The increase in performance due to metric learning comes at the cost of an increase in matching time due to the computationally expensive forward pass through the metric learning network. We discuss the matching time for different descriptors in Section 3.3.3.

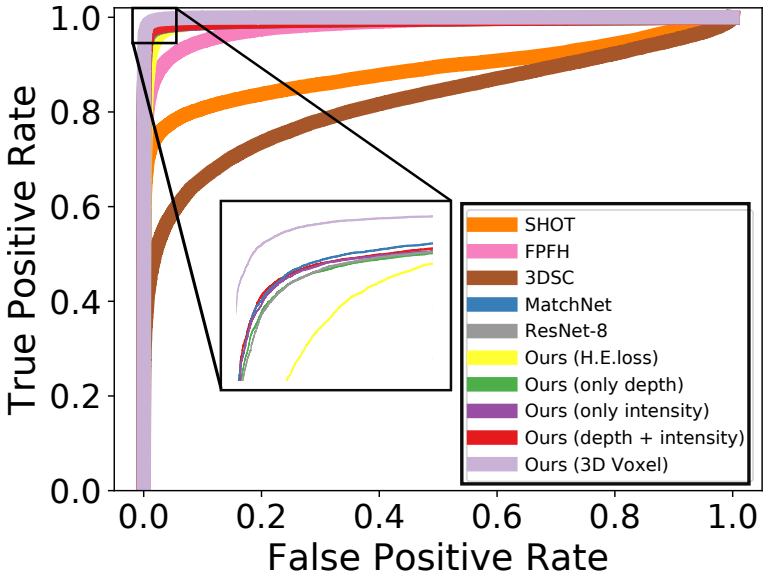


Figure 3.7: ROC curves for different feature descriptors. Feature descriptor learned using 3D voxels outperform other descriptors. Using H.E. loss instead of metric learning leads to a decrease in performance, whereas all handcrafted descriptors underperform in comparison to the learned descriptors.

Table 3.1: FPR95 Error.

Method	Feature Size	Error(%)
SHOT [112]	352	82.56
FPFH [95]	33	10.26
3DSC [35]	1980	89.16
MatchNet [42]	4096	0.45
ResNet-8	256	0.60
Ours (H.E. Loss)	256	1.94
Ours (depth + intensity)	256	0.42
Ours (only depth)	256	0.46
Ours (only intensity)	256	0.53
Ours (3D Voxels)	216	**0.15**

3.3.2 Alignment

Many methods for surface or scan registration require coarse initial alignment, especially when data is collected at a low rate (typically 10Hz for LiDAR) and the assumption that nearest neighbor points are corresponding does not hold. In this experiment, we align multiple objects scanned using two different LiDAR scanners.

Table 3.2: Alignment error for the individual objects using our descriptor learned with metric from 2D image patches

Object ID	Points	Keypoints	t_e(m)	r_e(rad)
Scans from Velodyne HDL-64E				
0	1369	483	0.28	0.04
1	493	285	0.54	0.04
2	787	332	0.47	0.04
3	250	186	0.52	0.02
4	1320	383	1.55	0.24
5	970	394	1.06	0.13
6	228	129	1.48	0.10
7	199	154	1.10	0.06
8	580	395	0.73	0.02
9	564	427	0.08	0.02
10	316	233	1.22	0.04
11	1517	908	0.23	0.01
Scans from Velodyne HDL-32E				
0	313	230	0.88	0.07
1	3271	1099	0.46	0.13
2	3741	1239	0.49	0.13
3	319	239	0.22	0.14

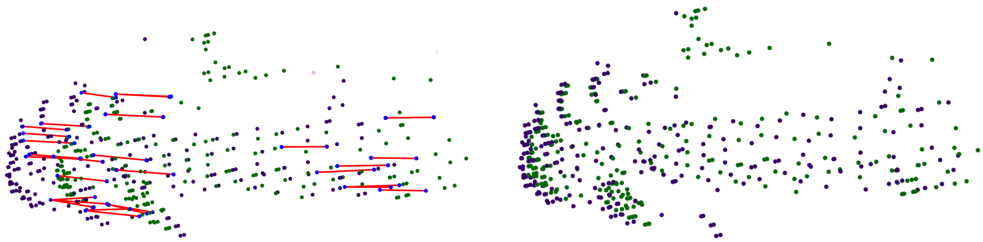


Figure 3.8: An illustration of the alignment experiment. The image on the left shows the sparse unaligned objects and the correspondences estimated by matching our feature descriptor. The image on the right shows the aligned pointclouds.

In Table 3.2, we report the number of objects that we used in the experiment, number of points belonging to each object, number of keypoints and translational and rotational alignment error for our descriptor that was learned using 2D image patches.

3.3.2.1 LiDAR Scans from Velodyne HDL-64E

In this experiment, we align 12 static objects extracted from consecutive LiDAR scans from the KITTI tracking benchmark. Since all objects are static, the ground-truth motion is the inverse sensor motion, which is provided by the benchmark. In Figure 3.8 we show example results: The image on the left shows the unaligned objects and the corresponding keypoints that were matched using our feature descriptor, learned with metric using 2D patches. The right image illustrates the aligned pointclouds. As before, we use uniform sampling for selecting keypoints. In Table 3.3, we report the average translational error t_e and rotational error r_e for the motion estimated using the raw point correspondences and correspondences filtered with RANSAC. The motion estimated using raw correspondences reflects more clearly on the feature matching accuracy in comparison to filtered correspondences. The alignment error reported for Ours (depth + intensity) method, using raw correspondences is the average of the errors reported in Table 3.2. Among the descriptors learned from 2D image patches, our descriptors with learned metric and hinge embedding loss have similar performance and they outperform the other descriptors by a significant margin. Similar to the matching accuracy experiment, the best performing descriptor (lowest error) is our descriptor learned from 3D voxels.

The rightmost column in Table 3.3 reports the feature matching time. The FPFH descriptor takes the least amount of time to match but has the largest alignment error. All of the handcrafted descriptors can be matched quickly using KD-trees in comparison to feature descriptors learned using a metric and, therefore, they have the lowest matching time. In this experiment, our feature descriptor learned using Euclidean distance performs most favorably considering both alignment error and matching time. Among the feature descriptors learned using the metric, the time for MatchNet is the largest. This increase in time is mainly attributed to the large descriptor size (4096 vs. 256), which results in twice as many parameters in the metric learning network in comparison to other learned descriptors. The matching times for ResNet-8, and our descriptor, both learned from 2D image patches and 3D voxels is similar, since the architecture for metric learning and the feature descriptor size are comparable.

3.3.2.2 LiDAR Scans from Velodyne HDL-32E

We repeat the alignment experiment with the scans from a 32 beam LiDAR scanner without retraining on these scans. Like before, we align static objects and use the sensor pose from the SLAM solution [64] as the ground-truth motion. The purpose of this experiment is to show that our feature descriptor is not overfitting to data collected from a single sensor but can generalize to data collected from different sensors. Even though both sensors provide the same modalities, the

Table 3.3: Average alignment errors for HDL-64E scans

Method	raw		RANSAC		t(s)
	t_e(m)	r_e(rad)	t_e(m)	r_e(rad)	
SHOT [112]	1.38±1.06	0.13±0.11	0.76±0.56	0.10±0.07	0.05
FPFH [95]	3.37±5.40	0.33±0.54	1.30±0.89	0.14±0.11	**0.004**
3DSC [35]	2.75±2.58	0.34±0.38	0.81±0.37	0.08±0.04	0.57
MatchNet [42]	0.88±0.47	0.08±0.06	0.93±0.77	0.07±0.05	14.78
ResNet-8	0.95±0.48	0.07±0.06	0.83±0.71	0.08±0.05	1.03
Ours (H.E. loss)	0.76±0.34	0.07±0.06	0.52±0.27	0.05±0.01	0.18
Ours (depth + intensity)	0.77±0.47	0.07±0.06	0.57±0.28	0.06±0.02	0.92
Ours (3D Voxels)	**0.43±0.23**	**0.04±0.02**	**0.37±0.24**	**0.03±0.02**	1.01

data from a Velodyne HDL-32E is sparser in comparison to data from a Velodyne HDL-64E. Table 3.4 shows the alignment error for different cases, and our feature descriptor with learned metric using 3D voxels outperforms the other descriptors. This experiment demonstrates that different learned descriptors are capable of generalizing on data from other sensors.

Table 3.4: Average alignment errors for HDL-32E scans

Method	raw		RANSAC	
	t_e(m)	r_e(rad)	t_e(m)	r_e(rad)
SHOT [112]	0.57±0.29	0.11±0.04	0.46±0.54	0.07±0.06
FPFH [95]	1.32±1.08	0.22±0.10	2.07±2.20	0.14±0.25
3DSC [35]	2.48±3.46	0.28±0.27	0.73±0.77	0.06±0.03
MatchNet [42]	0.97±1.01	0.13±0.08	0.88±1.24	0.14±0.11
ResNet-8	0.81±0.33	0.14±0.06	0.60±0.56	0.07±0.04
Ours (H.E. loss)	0.65±0.14	0.07±0.06	0.58±0.55	0.11±0.04
Ours (depth + intensity)	0.51±0.23	0.12±0.02	0.41±0.39	0.05±0.05
Ours (3D Voxels)	**0.42±0.18**	**0.07±0.03**	**0.26±0.11**	**0.05±0.01**

3.3.3 Computation Time

In Table 3.5 and 3.6, we report the computation time for estimating feature descriptors for different neighborhood and sampling radii. The neighborhood radius dictates the extent of local information captured around a keypoint and the sampling radius controls the number of keypoints in a scan for which descriptors are calculated. In the first case, we estimate descriptors for the same number of points (sampling radius of 0.4m) but for different neighborhood radii. This evaluation highlights per feature calculation time, which only depends on the input neighborhood radius.

In the second case, we use a fixed neighborhood radius of 3.2m while varying the sampling radius. This evaluation focuses on changes in computation time with the increase in number of keypoints. For each case, we report the processor details, i.e., whether it is a single core CPU (C), multi-core CPU (MC), or GPU (G). For learning methods, we separately report the time required for estimating the surface patches and for estimating the feature descriptors.

Table 3.5: Per feature computation time (in ms) for various neighborhood radii.

Method	Processor	Neighborhood radius (m)				
		0.4	0.8	1.6	3.2	6.4
SHOT [112]	MC	0.092	**0.096**	**0.119**	0.284	0.787
FPFH [95]	MC	**0.090**	0.282	0.961	3.01	10.23
3DSC [35]	C	0.148	0.440	3.21	31.19	344.10
MatchNet [42]	MC+	0.036+	0.037+	0.047+	0.075+	0.219+
	G	0.139	0.139	0.139	0.139	0.139
ResNet-8	MC+	0.036+	0.037+	0.047+	0.075+	0.219+
	G	0.151	0.151	0.151	0.151	0.151
Ours (H.E. loss)	MC+	0.036+	0.037+	0.047+	**0.075+**	**0.219+**
	G	0.132	0.132	0.132	**0.132**	**0.132**
Ours	MC+	0.036+	0.037+	0.047+	0.075+	0.219+
	G	0.133	0.133	0.133	0.133	0.133
Ours (3D Voxels)	MC+	5.14+	6.65+	9.04+	18.25+	42.36+
	G	0.99	0.99	0.99	0.99	0.99

For small neighborhood radii, handcrafted descriptors have a low computation time because different per-point operations, like estimating normals and estimating descriptors among other different operations, are performed more efficiently for smaller radii (fewer points). When increasing these radii (more points), these operations are not as efficient as before even when KD-Trees are used. In the case of the learned feature descriptors, the change in neighborhood radii only affects the patch computation time and not the feature estimation time. For 2D image patch computation, the voxelization of the neighborhood is independent of the number of points in the neighborhood. The only operation dependent on the number of points is calculating the average in depth and intensity values.

Among the descriptors learned using 2D image patches, our feature descriptor requires the least computation time in comparison to other architectures because our feature learning network has the fewest number of parameters. The computational time for estimating the 3D voxels and the corresponding descriptor is largest among learned descriptors. The 3D voxel representation requires estimating the nearest surface point for every voxel center and this computation contributes to the significant increase in computation time. The increase in feature computation time

Table 3.6: Computation time (in seconds) for various sampling radii.

Method	Sampling radius (m)						
	3.2	1.6	0.8	0.4	0.2	0.1	0.05
SHOT [112]	0.97	0.97	1.36	2.64	6.45	12.47	24.83
FPFH [95]	27.97	28.30	27.86	31.87	40.74	55.79	85.13
3DSC [35]	8.97	35.07	99.49	355.58	1111.65	3073.30	6179.10
MatchNet [42]	0.03+	0.11+	**0.28+**	0.81+	3.72+	4.40+	9.17+
	0.22	0.33	**0.67**	1.68	3.29	6.25	8.26
ResNet-8	**0.03+**	**0.11+**	0.28+	0.81+	3.72+	4.40+	9.17+
	0.20	**0.16**	0.70	1.68	3.61	6.75	9.84
Ours (H.E. loss)	0.03+	0.11+	0.28+	**0.81+**	**3.72+**	**4.40+**	**9.17+**
	0.33	0.42	0.70	**1.47**	**3.01**	**5.43**	**7.80**
Ours	0.03+	0.11+	0.28+	0.81+	3.72+	4.40+	9.17+
	0.34	0.43	0.71	1.49	3.05	5.52	7.90
Ours	5.098+	14.99+	41.70+	110.764+	271.38+	498.50+	810.23+
(3D Voxels)	0.42	1.57	2.66	6.85	16.38	30.52	46.20

is mainly due to an increase in dimension of the input data, causing an increase in the learnable parameters of the network.

With the decrease in sampling radius, the number of keypoints increases. The reported time is the combined computation time for all the keypoints. In this case, the better performance of learned feature descriptors is mainly attributed to the proper utilization of parallel processing capabilities of GPUs in comparison to the multi-core implementation of handcrafted descriptors on CPUs. In comparison to other learned descriptors from 2D patches, the performance of our feature learning network scales better with the increase in number of keypoints. Since our network is smaller than others, it allows us to process larger batches of data in parallel. Similar to before, the computation time among learned descriptors is largest for the descriptor learned using 3D voxels for the reasons discussed above.

3.3.4 Ablation Study

To better understand the contribution of each modality, we trained two separate networks with single channel input (depth and intensity). Table 3.1 shows the FPR95 error for both cases. The performance of the feature descriptor learned using depth is better than the feature descriptor learned using surface intensity values. Comparing the two modalities, depth is more informative since it captures the local geometrical information more reliably in comparison to the surface reflectance values. The error for both of these cases is higher than the feature descriptor learned using both modalities and, therefore, using them together helps in learning a more discriminative feature descriptor.

3.3.5 Discriminative Power

In the previous sections, we presented results for the matching and alignment accuracy for various descriptors. These experiments implicitly evaluated the discriminative nature of different descriptors. In this section, we analyze this characteristic more explicitly. We measure the similarity between descriptors for both matching and non-matching keypoints. Ideally, we would want the descriptor to be similar for matching keypoints and to be dissimilar for non-matching keypoints, and a large margin between similarity measure of descriptors for matching and non-matching keypoints. Our method for learning a descriptor, both simultaneously with metric learning and using the Euclidean metric, aims to achieve a large margin in similarity measure between these cases.

We use the same objects scanned using Velodyne HDL-64E reported in Table 3.2, for this experiment. For each of these objects, we estimate the ground-truth correspondences by first aligning the objects using the ground-truth motion and then selecting the nearest neighbor between keypoints in the aligned scan and the target scan. We report this analysis for the SHOT descriptor, which performs best among handcrafted descriptors; our feature descriptor learned using H.E. loss, and the feature descriptors simultaneously learned with a metric. This includes our feature descriptor (2D patches and 3D voxels) and the descriptor learned using the ResNet-8 and MatchNet architecture.

The similarity measure is different for the descriptors discriminative in the Euclidean space and the descriptors discriminative in the learned metric. For SHOT and our descriptor learned using H.E. loss, the similarity measure is the Euclidean distance between the descriptors, where a low distance value indicates a strong similarity and a large distance value indicates the opposite. For the other descriptors, the similarity measure is the confidence score of two descriptors being a match. The confidence score is the output of the metric learning network, where a value close to one indicates similarity between the descriptors, and a value close to zero indicates dissimilarity.

In order to visualize this similarity between descriptors, we use tSNE [118] to project these feature descriptors from a high dimensional space to a lower dimensional 3D space. tSNE is a method for estimating a low dimensional embedding for high dimensional data, where the objective is to preserve the relative distance between the original data in high dimensions and its low dimensional counterpart.

In Table 3.7, we report the estimated similarity between matching and non-matching keypoints averaged over all the objects for different descriptors. Additionally, we report the ratio of the descriptor similarity between non-matching and matching keypoints. This metric highlights the margin between them and, therefore, the descriptor's discriminative capability. For descriptors learned discriminative in Euclidean space, large values for this metric indicate the desired behavior. For the other descriptors the same behavior is characterized by small values.

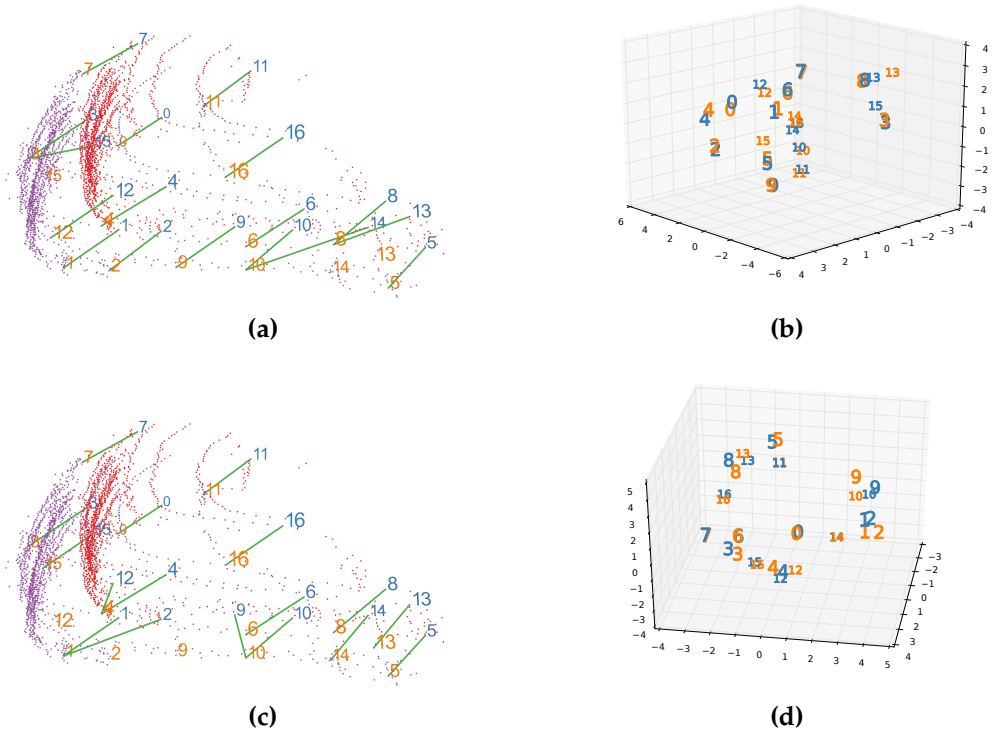


Figure 3.9: Illustration of feature descriptor matched using Euclidean metric. The left column shows the associated correspondences and the right column shows the low dimensional projection of the feature descriptors. The top row shows results for SHOT descriptor and the bottom row shows results for our feature descriptor learned using H.E. loss. For both cases, the majority of the corresponding keypoints are projected close to each other. The cases where non-matching keypoints are matched are also clearly visible. For SHOT, descriptor keypoints 13, 14 and 15 are incorrectly associated and for the other descriptor, keypoints 2,9 and 12 are incorrectly associated. The incorrect matching is evident in the tSNE representation as well.

The descriptor distance between matching keypoints for our descriptor learned using H.E. loss is lower than the distance computed for the SHOT descriptor but the distance for non-matching keypoints is larger for SHOT. The ratio of distances for non-matching and matching keypoints is larger for our descriptor, which strongly indicates that it is more discriminative in Euclidean space in comparison to the SHOT descriptor. This behavior was also observed in the other experiments, where our feature descriptor outperformed the SHOT descriptor by a significant margin.

In Figure 3.9c and 3.9d, for one of the objects in the experiment, we show the estimated correspondences between keypoints using our descriptor and the low dimensional projection of the feature descriptor for the same keypoints in the source and target scan respectively. Each keypoint is represented by a number and the same

number is used for representing the feature descriptor in low dimensional space. Keypoints with the same number but in different colors represent corresponding keypoints (ground-truth). The Majority of the matching keypoints are projected close to each other. The cases where non-matching keypoints are associated is also clearly visualized, for instance the keypoints: 2,9 and 12. The relative distance between these keypoints is small, as can be seen in the low dimensional space, which also indicates the same. Figure 3.9a and Figure 3.9b, show the same illustration for the SHOT descriptor.

For descriptors learned together with a metric, the similarity measure captures both the performance of the learned descriptor and the metric learned for matching the descriptors. Among different descriptors, the one learned using the MatchNet architecture and our descriptor learned from 3D voxels have the largest matching similarity score. This indicates that, on average, the descriptor learned using this architecture is more confident in matching similar keypoints, in comparison to the other descriptors. In the case of non-matching keypoints, the scores for all the descriptors are comparable but the descriptor learned using ResNet-8 architecture has the lowest score by a slight margin. This suggests that on average descriptors learned with a metric have a similar level of confidence in predicting when the keypoints are dissimilar. The ratio of the non-matching and matching score is also lowest for our descriptor (3D voxels). This indicates that our learned feature descriptor is better at differentiating between matching and non-matching keypoints in the learned metric, in comparison to the other learned descriptors and their respective metrics. In the case of both, our descriptor learned from 2D image patches and MatchNet, some non-matching keypoints are associated with a high confidence score, which results in a worse ratio of non-matching and matching score. In Figure 3.10, we illustrate the associated keypoints and the low dimensional projection of the descriptors learned using ResNet-8, MatchNet and our architecture (3D voxels and 2D patches) respectively. All correctly matched keypoints are grouped together in their low dimensional projection. The instances where keypoints are incorrectly matched are also clearly visible after the projection.

Table 3.7: Discriminative Power

Method	Matching	Non-matching	Non-matching/Matching
SHOT [112]	0.05±0.05	**1.17±0.08**	3.39±0.79
Ours (H.E. Loss)	**0.02±0.01**	1.02±0.13	**7.39±2.65**
ResNet-8	0.84±0.15	**0.04±0.02**	**0.06±0.02**
MatchNet	**0.90±0.08**	0.05±0.02	0.13±0.17
Ours (depth + intensity)	0.85±0.13	0.05±0.02	0.17±0.31
Ours (3D Voxel)	**0.90±0.08**	0.05±0.02	**0.06±0.02**

3.4 Related Work

In this section, we briefly discuss the handcrafted feature descriptors that we use for comparison with our method and an existing feature descriptor for sparse LiDAR scans. We also discuss different CNN based descriptors proposed for grayscale and dense 3D surface patches.

Several handcrafted local feature descriptors for 3D pointclouds have been proposed [9] and are currently part of the PCL library [94]. The first descriptor we compare with is the Fast Point Feature Histogram (FPFH) [95], which requires normals as input and generalizes the mean curvature around a point using a histogram. Tombari et al. [112] proposed the Signatures of Histograms of Orientations (SHOT) descriptor. Their contribution is a method for robustly estimating local reference frames and a descriptor that quantifies local surface normal information. The third image descriptor we compare with is 3D Shape Context (3DSC) [35]. Similar to other descriptors, it also requires surface normals as input and uses a spherical grid around the keypoint for counting the number of points in bins along azimuth, elevation and radial coordinates. We compare our results with these descriptors and show the advantages of using a learned feature descriptor over the handcrafted counterparts.

A feature descriptor designed for sparse 3D LiDAR scans was proposed by Serafin et al. [97]. They quantify the vertical structure in the scene with 3D lines and planes (circles) and show the efficacy of their approach by integrating these descriptors in a SLAM system. Even though their method improves the performance of the SLAM system, it might perform sub-optimally for environments that lack these vertical structures. In contrast to this, our feature descriptor is not quantifying any specific geometric structures in the environment and therefore can work in arbitrary environments.

With the advent of CNNs, several methods have been proposed for learning feature descriptors, but mainly for grayscale image patches. The architectures used by these methods consist of a network for learning the descriptor, followed by either a metric learning network for matching the descriptors or a loss layer, which minimizes the distance between the descriptors using a predefined metric. The architecture proposed in MatchNet [42] consists of a Siamese network for learning the descriptor and a metric learning module. DeepCompare [130] discusses several different architectures: Siamese, pseudo-Siamese, two-channel and central-surround two-stream networks. The difference between Siamese and pseudo-Siamese is that in the latter weights are shared only for selected layers instead of every layer. In two-channel architectures, the image patches are stacked as two channels instead of having a Siamese network. Central-surround two-stream networks consist of four input streams, two for complete image patches and two for the central crop of the input patches. They use a loss layer, which minimizes Euclidean distance instead of a metric learning module and show that the central-surround two-

stream architecture gives the best performance. MatchNet and DeepCompare are outperformed by recently proposed PN-Net [12]. This network architecture uses three input streams, where two streams have matching image patches and the third stream is a non-matching image patch. Unlike MatchNet, which uses softmax loss, they use SoftPN loss, a modified version of hinge loss [130].

A learning-based approach for 3D data was recently proposed by Zeng et al. [131]. Their approach, called 3DMatch, targets learning descriptors for dense 3D surface patches using a Siamese network trained with contrastive l_2 loss. One of the main contributions is the proposed 3D patch representation, which uses a 3D voxel grid of Truncated Distance Function values for representing the 3D shape. The key difference between their work and ours is that they target dense 3D surface patches extracted after aligning multiple scans, whereas we focus on learning descriptors for data from a single sparse 3D LiDAR scan.

3.5 Conclusions

In this chapter, we addressed the problem of robust correspondence matching between keypoints. We proposed two methods for learning a local feature descriptor for 3D LiDAR scans. The first method learns a feature descriptor and a metric for matching the descriptors simultaneously and the latter learns a feature descriptor that is discriminative in Euclidean space. We use an architecture based on dense blocks for feature learning and show how our architecture learns more discriminative feature descriptors in comparison to descriptors learned using other common architectures. We also propose two different methods for quantifying the local surface information around a keypoint. In the first method, the 3D neighborhood around a keypoint is represented as a 2D image patch, and in the other method with 3D voxels. We report results on matching accuracy and alignment error. For both cases our descriptor learned using the 3D voxel representation outperforms both handcrafted descriptors and other learned descriptors by a significant margin. We also report results for data collected from a different sensor and demonstrate how our descriptor can generalize to different sources of data. Among the two proposed methods, the feature descriptor learned together with a metric is more accurate but the other feature descriptor is more suitable when a faster matching time is required. Additionally, we present an ablation study to understand the importance of depth and intensity modalities and show that using them together enables learning of a more discriminative feature descriptor. Furthermore, we also provide an analysis regarding the discriminative characteristics of various descriptors and show that our learned feature descriptors are more discriminative in comparison to various other feature descriptors.

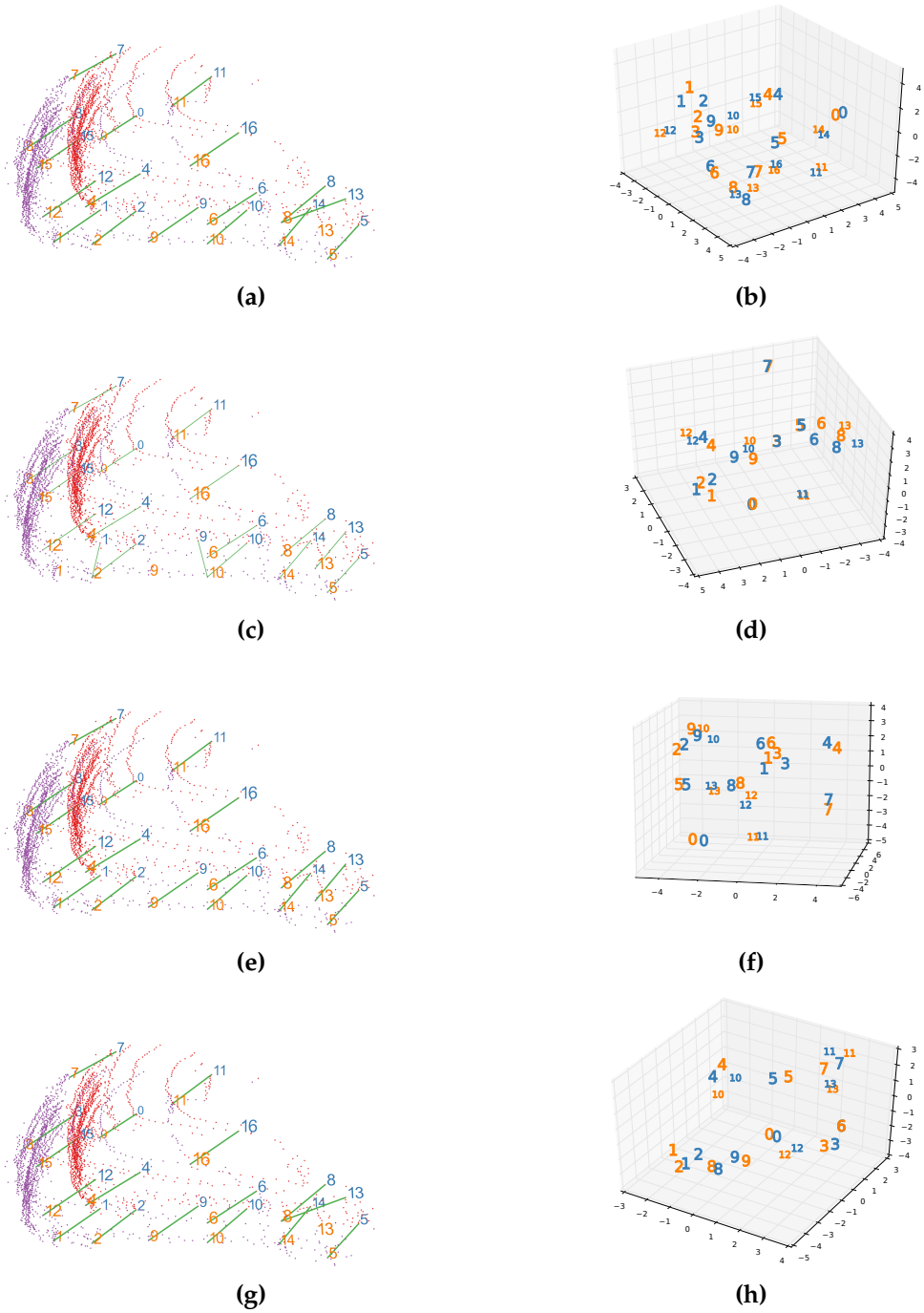


Figure 3.10: Illustration of feature descriptors matched using learned metric. The left column shows the matched keypoints and the right column shows low dimenstional projection of different descriptors. The results for feature descriptors learned using ResNet-8, MatchNet and our proposed architecture (3D voxels and 2D surface patches) are shown in the first, second, third and last row respectively. Keypoints that are correctly matched are grouped together after projection for all the descriptors. Similarly, incorrectly matched keypoints are also visible in the low dimensional projection.

Chapter 4

Motion Estimation in 3D LiDAR Scans

Robots are expected to operate autonomously in increasingly complex scenarios, such as crowded streets or heavy traffic situations. The perception of the dynamic aspects of the environment is crucial for safe and smart navigation and therefore is a highly relevant precondition and a key enabler for autonomous robot systems. In this chapter, we focus on scene understanding by leveraging only motion cues. We tackle this problem at both object and point level. We propose a novel model-free approach for detecting and tracking dynamic objects in 3D LiDAR scans obtained by a moving sensor. Our method only relies on motion cues and does not require any prior information about the objects. We sequentially detect multiple motions in the scene and segment objects using a Bayesian approach. For robustly tracking objects, we utilize their estimated motion models. We present extensive quantitative results on publicly available datasets and show that our approach outperforms a popular method. For understanding the dynamics of the scene at granular point level, we propose a novel method for estimating dense rigid motion field. We formulate the problem as an energy minimization problem and introduce the concept of local geometric constancy and incorporate regularization for dense smooth motion fields. We evaluate our approach on one simulated and two real datasets. Through evaluation, we show how our method is capable of estimating multiple rigid motion models, a necessity if the environment is highly dynamic. Building upon this method, we propose a Hidden Markov Model based method for inferring the motion state for points in a scan and present a comparison with our method of detection and tracking of dynamic objects.

4.1 Introduction

One of the major goals in the area of mobile robotics is to develop robot systems that can robustly navigate to accomplish different tasks, such as surveillance and transportation [44]. As the environment in which a robot is expected to operate typically cannot be assumed to be static [64], it is necessary for the robot to properly deal with the dynamic aspects of the environment. Perceiving the dynamics of the environment is necessary for proper scene understanding because it provides highly relevant information about the dynamic characteristics of various objects that the robot interacts with and also allows better understanding about how the environment might evolve in the future. For example, a self-driving car trying to cross an intersection in heavy traffic needs to be able to detect the individual dynamic objects in its vicinity, such as cars, bikes, trucks and pedestrians. Furthermore, it is necessary to estimate their individual motion characteristics to be able to navigate in a safe and efficient way.

In this chapter, we focus on 3D scene understanding solely based on motion cues. The objective is to infer dynamic and static parts of the scene by estimating the motion models pertaining to different dynamic entities in the environment. Inferring the motion cues i.e., estimating the motion models offers many advantages. It can be used for estimating the sensor motion as well as motion of multiple dynamic objects [74]. The estimated motion then seamlessly paves the way for desired semantic classification of a 3D scan into dynamic or static classes as well. Furthermore, removing the dynamic objects from a map can help to more accurately estimate the pose of a robot, and predicting the location of dynamic objects facilitates safe and efficient motion and path planning.

We approach our problem of 3D scene understanding using motion cues from two different perspectives. First, we address this problem at object level and propose a method for detection and tracking of dynamic objects in outdoor urban environments. This approach is a model free approach i.e. no prior object level semantic knowledge is used. Our method solely relies on the assumption that a scan can be decomposed into a set of rigid objects. Secondly, we address this problem at the finer point level. We propose a method for estimating pointwise motion and plug in the estimated motion in a Hidden Markov Model (HMM) based approach to classify points in a 3D LiDAR scan as either dynamic or static. In the first method, we assumed that objects are rigid and in the case of estimating pointwise motion, we partially relax this assumption and assume objects are locally rigid i.e. neighboring points are expected to have similar motion. This relaxation in assumption enables estimation of motion of non-rigid bodies, like humans.

Our proposed method for detection and tracking of dynamic objects is an iterative approach comprised of two steps. In the first step, we estimate the most dominant motion model using RANSAC [32]. The input in this case is a set of sparse correspondences between keypoints in two consecutive scans. In the next step, we

use a novel Bayesian approach to segment the complete object moving according to the estimated dominant motion. In the first iteration, the estimated motion is the sensor motion, assuming the majority of the points in a scan are static and in the subsequent iterations, we detect different dynamic objects by estimating different motion models.

To estimate pointwise motion, we formulate an energy minimization problem, the output of which is a dense rigid motion field. In contrast to the first approach, which required multiple iterations of RANSAC for estimating different motion models, in this approach we estimate arbitrary motions using a single optimization run. This method assumes local rigidity and we exploit this assumption by introducing the concept of geometric constancy i.e. local geometry remains unchanged after motion transformation. As we discussed in the previous chapter, a prerequisite for estimating motion is data association between keypoints. To this end, we introduce a method of estimating robust data association between sparse keypoints in consecutive scans.

To infer the motion state for points in a LiDAR scan, we propose a Hidden Markov Model (HMM) based approach for estimating the probability of a point being static or dynamic given the motion field and the sensor motion. Having such an approach helps in incorporating temporal consistency in estimated motion states, thereby reducing the number of false positives arising due to sporadic noisy measurements.

The primary contribution of this chapter is different methods for 3D scene understanding using motion cues. We address this problem at object level and also at a more granular point level. We propose a novel method for detection and tracking of dynamic objects. We test our method on the dataset provided by Moosmann and Stiller [74] and also compare ours with their approach. We report: evaluation for the detection of dynamic objects, the error in the estimated sensor motion and motion for one of the dynamic objects.

We also propose a novel method for estimating rigid motion field. This method is capable of estimating arbitrary motions in the scene and works with both non-rigid scenes and objects. We test our approach on one simulated and two real datasets. The simulated dataset [129] consists of urban outdoor environment with multiple dynamic objects. The first real dataset we evaluate our approach on is the KITTI odometry benchmark [36]. Similar to the simulated dataset, this dataset only consists of outdoor urban environments. In both of these datasets the environment itself is non-rigid whereas each individual object is often rigid. Focusing on non-rigid objects rather than scenes, we evaluate our approach on a second real dataset consisting of pedestrians. For comparison, we use the Iterative Closest Point (ICP) algorithm [70] as a baseline method. For all of the experiments we report the alignment accuracy and for the cases where the ground-truth motion is available, we also report the error in the estimated motion. We also present an ablation study for a better analysis of the different features of our approach. In Chapter 3,

we proposed a feature descriptor for estimating robust data association between keypoints in different scans, and in this chapter we incorporate this descriptor for estimating pointwise motion. To evaluate the estimated motion for each point we again use the dataset from Moosmann and Stiller [74] and compare the HMM based approach with the method of detecting and tracking of dynamic objects.

4.2 Detection and Tracking of Dynamic Objects

Our objective is to detect different dynamic objects in the scene without incorporating prior information about the objects. Traditionally, model-free approaches [61], [82], [124], [129] rely on detecting dynamic objects by analyzing the perceived change of the environment caused by motion. This change detection approach either requires prior map [82] or an online mapping technique. In contrast to that, our approach does not require a map, because instead of detecting changes our method focuses on segmenting distinct objects using motion cues. We begin to reason about objects at point level by matching corresponding keypoints in consecutive scans. This information is used to detect different motions in the scene by sequentially using RANSAC. Besides the motion models, RANSAC also provides an inlier set which in our case is a subset of input corresponding keypoints. For segmenting the local static structure and multiple dynamic objects solely based on the detected motion, we propose a Bayesian segmentation approach which uses the inlier set as an input seed for the segmentation. Since we estimate motion models, we can reason about the dynamics of a detected object to efficiently track it. Tracking objects is a challenging problem since the sensor motion and the motion of objects lead to frequent occlusions, making data association a hard problem. Our framework leverages the estimated motion models for associating different detected objects in consecutive scans.

4.2.1 Framework Overview

The goal of our approach is to segment and track dynamic objects in LiDAR scans obtained by a mobile robot. Our framework shown in Figure 4.1 consists of modules for detecting motion, tracking the sensor and tracking dynamic objects. We define, a LiDAR scan as a set of points:

$$P = \{\mathbf{p}_k \mid \mathbf{p}_k \in \mathbb{R}^3, k = 1, \ldots, K\} \tag{4.1}$$

At every time step t, the scans P_{t-1}, P_t, and the motion models $\boldsymbol{\tau}_{t-1} \in SE(3)$ are provided to the framework. In our notation $\boldsymbol{\tau}_t$ describes the motion from P_{t-1} to P_t. The points in P_{t-1} are classified as either static, dynamic, or unknown, i.e.,:

$$P_{t-1} = P_{t-1}^S \sqcup P_{t-1}^D \sqcup P_{t-1}^U \tag{4.2}$$

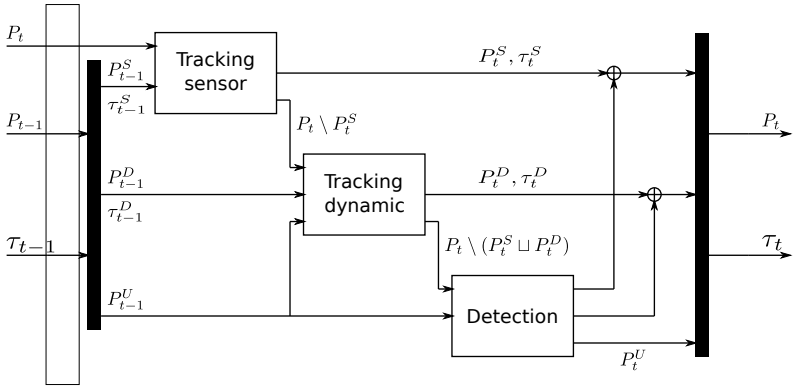


Figure 4.1: Our framework consists of modules for motion detection, tracking the sensor, and tracking dynamic objects. Point sets are indicated with P, motions models with $\boldsymbol{\tau}$.

First, the sensor tracking module classifies a subset of points $P^S_t \subset P_t$ as static and estimates the sensor motion $\boldsymbol{\tau}^S_t$ relative to these points. Second, the object tracking module classifies a subset of points $P^D_t \subset (P_t \setminus P^S_t)$ as dynamic, where $P^D_t = P^{D_1}_t \sqcup \ldots \sqcup P^{D_N}_t$ consists of multiple disjoint point sets assigned to different dynamic objects. The module also estimates motion models $\boldsymbol{\tau}^D_t = \{\boldsymbol{\tau}^{D_1}_t, \ldots, \boldsymbol{\tau}^{D_N}_t\}$ for these objects. Third, all points $P_t \setminus (P^S_t \sqcup P^D_t)$ not classified by the tracking modules are provided to the detection module, which either adds them to the static point set, creates a new dynamic object, or assigns them to the unknown set P^U_t. For the first scan all points are unknown, i.e. $P_1 = P^U_1$.

4.2.2 Motion-based Detection

We segment dynamic objects using only motion cues. The motion of points between two consecutive LiDAR scans is mainly caused by the motion of the sensor and the motion of dynamic objects. We assume that the motion of these objects is rigid. The following subsection describes how we estimate motion models for the sensor and the dynamic objects. Subsequently, we explain the proposed Bayesian approach which calculates the probability of a point to follow a given motion model. Finally, we introduce our data association between consecutive scans and outline the method for detecting multiple motions. To simplify the notation in this section we assume without loss of generality that no points are classified already, i.e., $P_{t-1} = P^U_{t-1}$.

4.2.2.1 Motion Models

We use RANSAC to estimate motion models $\boldsymbol{\tau}_t \in SE(3)$ for the sensor and the dynamic objects. To find initial point correspondences between the two scans, we uniformly sample keypoints $F_{t-1} \subset P_{t-1}$, match their SHOT descriptors [112]

against all points in P_t, and pick the matches with minimum descriptor distance. We use the Euclidean distance between feature descriptors as the metric for matching. To limit the false correspondences between keypoints, we discard correspondences with a distance greater than a threshold, which is determined by an assumed motion limit and the sensor frame rate. RANSAC estimates the motion model $\boldsymbol{\tau_t}$ consented by the majority of the remaining correspondences. We define the inlier point set $I_{t-1} \subset F_{t-1}$ as all points in P_{t-1} that are part of an inlier correspondence.

So far, $\boldsymbol{\tau}_t$ is only related to the motion of points in I_{t-1}, which is a sparse subset of all the points in P_{t-1}. Therefore, it is necessary to infer all non-inlier points which also follow $\boldsymbol{\tau}_t$. In the next subsection, we propose an approach to tackle this problem.

4.2.2.2 Bayesian Segmentation Approach

We propose a Bayesian segmentation approach to determine the probability of each point $\mathbf{p}_k \in P_{t-1}$ to follow a given motion model. By these means, we expand the sparse subset I_{t-1} to all points in scan P_{t-1}. We represent the consent of $\mathbf{p}_k$ with $\boldsymbol{\tau}_t$ as a Bernoulli distributed random variable H_k, where $H_k = 1$ means that $\mathbf{p}_k$ follows the motion model $\boldsymbol{\tau}_t$. The objective of the proposed Bayesian approach is to calculate the probability $P(H_k \mid P_t, \hat{\mathbf{p}}_k)$, where $\hat{\mathbf{p}}_k \in \hat{P}_{t-1}$ is the point $\mathbf{p}_k$ transformed by $\boldsymbol{\tau}_t$. We apply Bayes' rule and assume independence of H_k and $\hat{\mathbf{p}}_k$ to calculate this probability:

$$P(H_k \mid P_t, \hat{\mathbf{p}}_k) = \frac{P(P_t \mid H_k, \hat{\mathbf{p}}_k) P(H_k, \hat{\mathbf{p}}_k)}{P(P_t, \hat{\mathbf{p}}_k)} \tag{4.3}$$

$$= \frac{P(P_t \mid H_k, \hat{\mathbf{p}}_k) P(H_k) P(\hat{\mathbf{p}}_\mathbf{k})}{P(P_t \mid \hat{\mathbf{p}}_k) P(\hat{\mathbf{p}}_k)} \tag{4.4}$$

$$\propto P(P_t \mid H_k, \hat{\mathbf{p}}_k) P(H_k) \tag{4.5}$$

In the following subsections, we describe how we model the likelihood $P(P_t \mid H_k, \hat{\mathbf{p}}_k)$ and the prior $P(H_k)$ in Eq. (4.5). The proposed Bayesian approach relies on the fact that $\hat{\mathbf{p}}_k$ is well aligned with points in P_t, if $H_k = 1$. Therefore the depth, i.e. the distance to the sensor, of $\hat{\mathbf{p}}_k$ and the neighboring points in P_t should be similar. This is also true for local geometry. Furthermore, we impose by regularization that $\mathbf{p}_k$ is likely to follow the same motion as points in its vicinity.

4.2.2.3 Depth

The depth of the point $\mathbf{p}$ is denoted z. Since the likelihood of H_k actually only depends on the neighborhood $N_k \subset P_t$ of $\hat{\mathbf{p}}_k$ and we assume independence between the points $\mathbf{p}_l \in N_k$ we can model the likelihood in Eq. (4.5) based on the depth

alignment [49] as:

$$P(P_t \mid H_k, \hat{\mathbf{p}}_k) = \prod_{\mathbf{p}_l \in N_k} P(z_l \mid H_k, \hat{z}_k) \tag{4.6}$$

Similar to [82], we define N_k as the conical frustum around $\hat{\mathbf{p}}_k$. It better captures the disparity caused by misalignment in comparison to a spherical neighborhood.

To model the likelihood $P(z_l \mid H_k, \hat{z}_k)$, we use the beam-based sensor model from [110]. For $H_k = 1$, it is a linear combination of four distributions:

$$P(z_k \mid H_k = 1, \hat{z}_k) = \begin{pmatrix} w_{hit} \\ w_{short} \\ w_{max} \\ w_{rand} \end{pmatrix}^T \begin{pmatrix} P_{hit} \\ P_{short} \\ P_{max} \\ P_{rand} \end{pmatrix} \tag{4.7}$$

where $w_{hit} + w_{short} + w_{max} + w_{rand} = 1$ and P_{hit} models the probability of hitting the expected surface, P_{short} of hitting an unexpected obstacle in front, P_{max} of a maximum range measurement, and P_{rand} of an unexplainable measurement.

$$P_{hit} = \begin{cases} \eta \mathcal{N}(z_k; \hat{z}_k, \sigma^2_{hit}) & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

$$P_{short} = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short} z_k} & \text{if } 0 \leq z_k \leq \hat{z}_k \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

$$P_{max} = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

$$P_{rand} = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

For $H_k = 0$ the likelihood (4.7) changes to a mixture of three distributions:

$$P(z_k \mid H_k = 0, \hat{z}_k) = \begin{pmatrix} w_{short} \\ w_{max} \\ w_{rand} \end{pmatrix}^T \begin{pmatrix} p_{short} \\ p_{max} \\ p_{rand} \end{pmatrix} \tag{4.12}$$

where the weights are adapted and P_{short} in Eq. (4.9) is cut off at the maximum range z_{max} instead at the expected measurement $\hat{z}_k$.

4.2.2.4 Local Geometry

We measure consistency of local geometry by calculating the cosine similarity c_{kl} between the SHOT descriptors f_k and f_l:

$$c_{kl} = \frac{f_k \cdot f_l}{\|f_k\| \|f_l\|} \tag{4.13}$$

To incorporate this consistency measure of local geometry into the likelihood, we change Eq. (4.6) by weighting the individual factors with c_{kl}:

$$P(P_t \mid H_k, \hat{\mathbf{p}}_k) = \prod_{\mathbf{p}_l \in N_k} c_{kl} P(z_l \mid H_k, \hat{z}_k) \tag{4.14}$$

4.2.2.5 Regularization

Since $\mathbf{p}_k$ likely follows the same motion as points in its vicinity, we impose a prior $P(H_k)$ in Eq. (4.5). We model this prior by utilizing the information provided by RANSAC, namely if points in the neighborhood of $\mathbf{p}_k$ are in the inlier point set $I_{t-1} \subset F_{t-1}$ or not. The neighborhood is calculated by drawing a sphere around $\mathbf{p}_k$:

$$N_k^F = \{\mathbf{p}_l \in F_{t-1} \mid \|\mathbf{p}_k - \mathbf{p}_l\| < 3\sigma_w\} \tag{4.15}$$

Points $\mathbf{p}_k$ that have many inlier and few outlier points in their vicinity should have a high prior to consent with $\boldsymbol{\tau}_t$ and vice versa. We realize these characteristics by computing a weighted average:

$$P_0(H_k) = \begin{cases} 1 & \text{if } \mathbf{p}_k \in I_{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

$$P(H_k) = \frac{\sum_{\mathbf{p}_l \in N_k^F} w_{kl} P_0(H_k)}{\sum_{\mathbf{p}_l \in N_k^F} w_{kl}} \tag{4.17}$$

Since, we want closer points to have a higher influence on the prior, we use a Gaussian for the weighting:

$$w_{kl} = \frac{1}{\sqrt{2\pi\sigma_w^2}} \exp - \frac{\|\mathbf{p}_k - \mathbf{p}_l\|^2}{2\sigma_w^2} \tag{4.18}$$

The proposed Bayesian approach is used to calculate the probability of points in P_{t-1} to follow the motion model $\boldsymbol{\tau}_t$. The next subsection describes how we associate $\boldsymbol{\tau}_t$ with points in the current scan P_t.

4.2.2.6 Data Association

To classify points in the current scan P_t, we need to associate them to the corresponding motion models $\boldsymbol{\tau}_t$. Due to sensor motion and noise, every scan consists of varying points which are sampled from surfaces of the real structure. Therefore, establishing point-to-point correspondences between P_{t-1} and P_t is not feasible.

To realize the data association, we first identify points $\mathbf{p}_k^* \in P_{t-1}^* \subset P_{t-1}$ with a probability $P(H_k = 1 \mid P_t, \hat{\mathbf{p}}_k) > \zeta$, i.e. points that have a high probability to follow the motion model $\boldsymbol{\tau}_t$. Second, we determine the union of neighborhoods N by drawing spheres around all points $\hat{\mathbf{p}}_k^* \in \hat{P}_{t-1}^*$. Third, we declare points $\mathbf{p}_l \in P_t$

to follow the motion $\boldsymbol{\tau}_t$ if they lie inside N. In this way, we classify points in P_t according to their motion and assign them to the static point set P_t^S or to a point set of a dynamic object $P_t^{D_i}$. We apply Euclidean clustering to handle multiple objects following the same motion.

4.2.2.7 Multiple Motions

We use RANSAC to detect motion in the scene Section 4.2.2.1. To identify all points in scan P_t that follow a motion model $\boldsymbol{\tau}_t$, we use the Bayesian approach (Section 4.2.2.2) and apply the data association (Section 4.2.2.6). To detect multiple motions, we apply this pipeline sequentially. The detected $\boldsymbol{\tau}_t$ always stems from the motion which is consented by the majority of points. Therefore, we remove all these points from P_{t-1} and P_t to subsequently determine the next motion model.

Since we assume that most points originate from static structure, the first $\boldsymbol{\tau}_t$ we detect is the sensor motion $\boldsymbol{\tau}_t^S$. We create a new dynamic object if

$$\|\boldsymbol{\tau}_t^{D_i} - \boldsymbol{\tau}_t^S\|_F > \epsilon \tag{4.19}$$

where $\boldsymbol{\tau}_t^{D_i}$ is its estimated motion model, $\|\cdot\|_F$ the Frobenius norm, and ϵ a threshold that determines the minimum motion of an object to be considered as dynamic. If Eq. (4.19) is false, we add points to P_t^S.

4.2.3 Tracking the Sensor and Dynamic Objects

Tracking is concerned with propagating information over time, in our case from LiDAR scan P_{t-1} to P_t. This involves updating the motion models of the sensor and the dynamic objects from $\boldsymbol{\tau}_{t-1}$ to $\boldsymbol{\tau}_t$. Furthermore, the points in P_t have to be classified as static or assigned to a specific dynamic object. In the following subsection, we describe how we track the motion of the sensor and of dynamic objects.

4.2.3.1 Sensor Motion

We first update the motion model of the sensor from $\boldsymbol{\tau}_{t-1}^S$ to $\boldsymbol{\tau}_t^S$. To get an initial estimate how static points have moved, we apply the previous motion model:

$$\hat{P}_{t-1}^S = \boldsymbol{\tau}_{t-1}^S * P_{t-1}^S \tag{4.20}$$

Based on this estimate, we search for correspondences between all points $\hat{\mathbf{p}}_\mathbf{k} \in \hat{P}_{t-1}^S$ and points in P_t using nearest neighbor in Euclidean space. We apply RANSAC to estimate the motion $\hat{\boldsymbol{\tau}}_t^S$ and update the motion model of the sensor:

$$\boldsymbol{\tau}_t^S = \hat{\boldsymbol{\tau}}_t^S * \boldsymbol{\tau}_{t-1}^S \tag{4.21}$$

To classify points in P_t as static, we first use the proposed Bayesian approach (Section 4.2.2.2). Based on the probabilities $P(H_k \mid P_t, \hat{\mathbf{p}}_\mathbf{k})$, we identify points that were static in P_{t-1} but are dynamic or disappeared in P_t, i.e. have a low probability. Points with a high probability to consent with the sensor motion $\boldsymbol{\tau}_t^S$ are used in the data association Section 4.2.2.6 to assign points to $P_t^S \subset P_t$.

4.2.3.2 Dynamic Objects

The tracking of dynamic objects relies on a similar concepts as the sensor tracking. However, updating the motion model of an object from $\boldsymbol{\tau}_{t-1}^{D_i}$ to $\boldsymbol{\tau}_t^{D_i}$ is realized differently. First, we again apply the previous motion model to get an initial estimate where the object has moved:

$$\hat{P}_{t-1}^{D_i} = \boldsymbol{\tau}_{t-1}^{D_i} * P_{t-1}^{D_i} \tag{4.22}$$

We determine a neighborhood $N \subset P_t$ as it is done in Section 4.2.2.6. This provides a coarse prior information, where to search for correspondences, namely between points in $P_{t-1}^{D_i}$ and N. We find correspondences by matching SHOT descriptors, which are subsequently used by RANSAC to estimate the motion model $\boldsymbol{\tau}_t^{D_i}$. To assign object points to $P_t^{D_i} \subset P_t$, we choose the same approach as for the sensor tracking.

We create a tracklet for every segmented dynamic object, which is defined by its motion model $\boldsymbol{\tau}_t^{D_i}$ and point set $P_t^{D_i}$. Tracklets tracked for more than N_T scans are promoted as tracks. We make this distinction to avoid false positives. A track is lost when an object is no longer in the sensors field of view or when it is entirely occluded, i.e. $P_t^{D_i} = \emptyset$. To tackle temporary occlusions we predict a bounding box based on the objects last observation. We recover a track when points reappear inside the tracks bounding box.

4.2.4 Results

We evaluate our approach on two datasets made available by Moosmann and Stiller [74]. Both scan sequences A and B are collected in non-flat urban environments using a Velodyne HDL-64E LiDAR sensor at 10Hz and are 38 and 50 seconds long, respectively. The ground truth velocity of the sensor was obtained using DGPS/IMU and is also available for one other car. To conduct an extensive quantitative analysis, we manually labeled all dynamic objects apart from pedestrians and compared the results of our approach against the Moving Object Mapping (MOM) method presented by Moosmann and Stiller [74]. For all experiments presented in this section, we chose $\zeta = 0.95$, $\epsilon = 0.2$, and $N_T = 8$.

Figure 4.2 shows a snapshot of sequence B taken at $t = 26.2s$. Since we remove the ground plane before we provide a scan to the framework, ground points are not classified. It can be seen that we are able to segment and track many objects

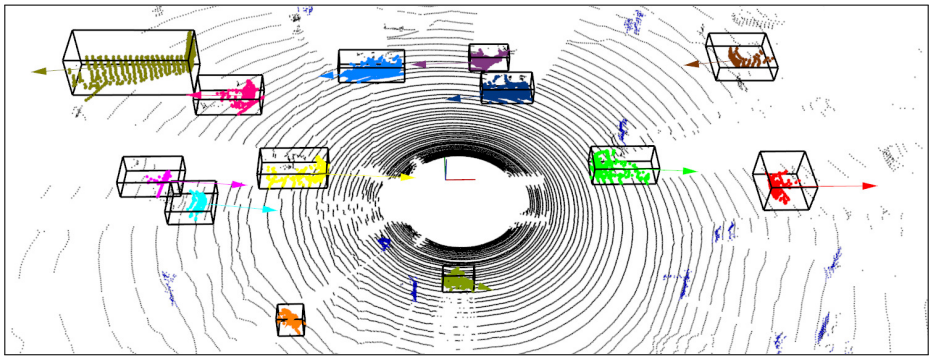


Figure 4.2: LiDAR scan of sequence B at $t = 26.2s$. The Ground truth is indicated by black bounding boxes. Structures classified as static are shown in blue, dynamic objects in other colors. Arrows display the translational part of the estimated motion models.

of various types, e.g. cars, trucks, and bikes. This is one major advantage of our approach compared to model-based methods. Admittedly, our approach does not work well for pedestrians. This is mainly for the reason that they move slowly and our detection method only relies on motion cues. Furthermore, the assumption of rigid body motion does not hold. Due to their comparatively small size, pedestrians may also consist of very few points, especially if they are far away from the sensor. Since pedestrians are not labeled in the ground truth, we ensure not to count them as false positives in the analysis.

Table 4.1: Classification of Dynamic Objects

Method	Precision	Recall	F_1	ObjRcl
Sequence A				
Ours	**62.41**	**91.33**	**70.76**	**81.77**
MOM	14.89	72.91	22.02	30.90
Sequence B				
Ours	**72.57**	78.18	**73.05**	74.08
MOM	29.85	**89.01**	41.81	**79.91**

For a quantitative evaluation of our approach, we use the ground-truth bounding boxes of the dynamic objects. For each scan we compute precision (Eq. (2.17)) and recall (Eq. (2.18)). We average precision and recall over all scans of the sequence and compute the F_1 score (Eq. (2.19)). We also define an object recall (ObjRcl) that measures the ratio between the number of scans in which an object is detected and scans in which it is actually there according to the ground-truth . We average this value over all objects. In contrast to recall, every object equally contributes to the object recall, independent on the number of scans in which the object is present.

Table 4.1 reports results for our approach and compares them. In sequence A, we clearly outperform MOM which suffers from a high number of false positives. Its

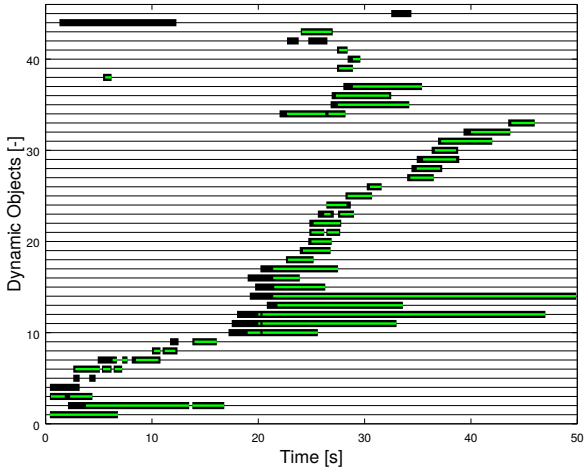


Figure 4.3: Tracks for sequence B. Ground truth is depicted in black, estimated tracks are colored green. Gaps in the ground truth are caused by occlusions.

different recall and object recall values are caused by the dominance of one object, that in contrast to others, is present over the whole sequence and can be tracked. In sequence B, our approach reaches significantly better precision with a slightly worse recall. In both sequences we achieve better F_1 scores.

Figure 4.3 illustrates the tracks estimated by our approach in comparison to the ground truth for sequence B. We are able to segment almost all objects and track them robustly. Since our approach requires a minimum motion to consider an object as dynamic, we can only detect objects when they reach a certain velocity. Our loss of recall is primarily due to the late detection of the objects 10-18 and the few undetected objects. These cases were recorded at an intersection and include slow moving objects either approaching or leaving the intersection. Figure 4.3 also depicts cases where objects were temporarily occluded. Since we always recover from occlusions if we detect the object, we claim that our approach is robust against occlusions.

To further compare our approach, we also conduct the tracking quality experiment presented by Moosmann and Stiller [74]. The objective of this experiment is to estimate the *absolute* speed of another vehicle. Therefore, we evaluate the estimation of the sensor motion and the relative motion between sensor and dynamic object. Figure 4.4 shows that we can robustly track both in sequence A. The peak in the sensor motion is caused by a corrupted LiDAR scan. In Table 4.2 we report the median, mean, standard deviation, and RMSE generated without the outer 10%-quantiles for the sensor motion in both sequences. The table also shows results for the dynamic objects tracking in comparison to the results reported in [74]. It can be seen that our approach outperforms MOM. To provide a more informative measure, we also report the RMSE for our approach.

Table 4.2: Estimation of Sensor and Dynamic Object Motion [generated without outer 10%-quantiles]

	Median	Mean	Std.-Dev.	RMSE
Sensor Speed Error				
Ours A	-0.19	-0.06	±0.57	0.57
Ours B	-0.09	-0.08	±0.26	0.30
Object Speed Error				
Ours	**-0.34**	**-0.32**	±**0.68**	0.75
MOM	-0.84	-0.69	±1.16	-

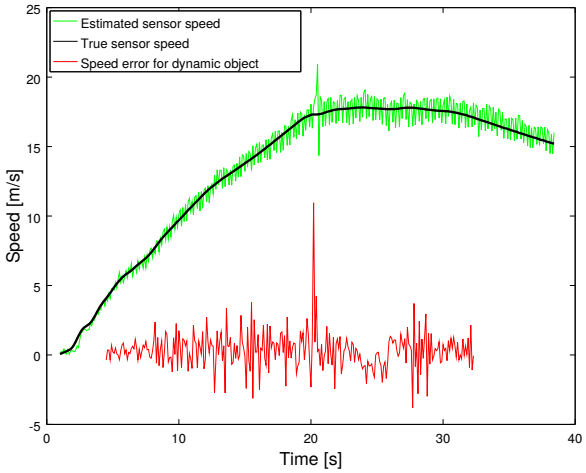


Figure 4.4: Speed estimation for sequence A. The ground truth for the sensor motion is depicted in black, our estimate in green. The error in the speed estimation for the dynamic object is colored red.

4.3 Estimating Pointwise Motion

So far we have focused on estimating motion models at object level and proposed an approach for detection and tracking of dynamic objects. In this section, we propose a model free approach for estimating pointwise motion for 3D LiDAR scans. In the approach discussed above, we focused on capturing object level dynamics. Now we focus on capturing the fine details of the environment dynamics by estimating pointwise motion. For example, such an approach can help the robot deal with human motion, which is typically non-rigid.

Pointwise motion has garnered a lot of attention, mainly in the computer vision community. Various methods have been proposed for estimating 2D pointwise motion in images using color [34]. With the recent advent of affordable depth sensors, different methods exist for scene flow estimation using color and depth images [50, 58, 86]. These methods cannot be directly applied to 3D pointcloud

data due to inherent differences in the problem structure. First, the constancy assumptions for intensity and gradients are not valid for LiDAR data. The intensity values from LiDARs are often unreliable as they depend on the angle of inclination. Furthermore, due to the sparse nature of LiDAR data, gradients are not informative. Second, the concept of the neighborhood is well understood for images (fixed size image patch), whereas a similar well defined structure does not exist for unstructured 3D pointcloud data. Third, most of these methods assume a piecewise linear motion (translation), which is justified if data is collected at a high frame rate (30Hz). In cases in which data is not collected at a sufficiently high frame rate (with a LiDAR scanner at 10Hz), this assumption cannot be justified.

To estimate pointwise motion, we propose a novel approach for estimating rigid scene flow that addresses all of the aforementioned challenges. We formulate the problem as an energy minimization problem. To this end, we introduce the concept of *geometric constancy*, i.e. that the local structure is not deformed due to the motion, thereby assuming the 3D structure is locally rigid. This assumption is exploited by using local feature descriptors for matching keypoints between two scans. For this task, we introduce a method to estimate robust data associations. In addition, we introduce neighborhood structure for 3D pointclouds. Our approach approximates the scene surface using triangular meshes and considers two points as neighbors, if they are vertices of the same triangle. Lastly, we estimate the complete 6D rigid motion, instead of linear (translational) motion only, thereby estimating dense rigid scene flow for 3D LiDAR data.

We test our approach on both simulated and real data. For simulated data, we use the dataset from Yoon et al. [129]. The main advantage of such datasets is that the ground-truth poses are available for both the sensor and the dynamic objects, with which ground-truth motion models can be calculated. For real data, we test our approach on multiple sequences of the KITTI odometry dataset [36] and a dataset capturing pedestrians that we collected. In the case of KITTI, we evaluate the accuracy of the estimated motion using the ground-truth sensor motion provided by the benchmark. This evaluation is only done for sequences without dynamic objects, since ground-truth motion for dynamic objects is not available. For the cases in which ground-truth motion is not available, we quantify the accuracy of the estimated motion by measuring the alignment between scans. We compare our approach against the ICP algorithm and also provide an exhaustive analysis of different features of our approach through an ablation study. We also incorporate the learned feature descriptor from Chapter 3 and present results for that. Through experiments on simulated and KITTI dataset, we demonstrate how our method effortlessly estimates arbitrary motion models in a non-rigid scene i.e. a scene containing multiple dynamic objects. The experiments on pedestrian dataset reveal that our method seamlessly adapts to the case of non-rigid objects as well.

4.3.1 Problem Formulation

A LiDAR scan P is given by a set of 3D points as defined in Eq. (4.1). Given two LiDAR scans P_{t-1} and P_t, the objective is to find a dense rigid motion field that best explains the motion between two scans. A rigid body transformation for a point $\mathbf{p} \in \mathbb{R}^3$ can be written as:

$$T(\mathbf{p}) = \mathbf{R}\mathbf{p} + \mathbf{t}, \tag{4.23}$$

where $\mathbf{t} \in \mathbb{R}^3$ is the translation and $\mathbf{R} \in SO(3)$ is the rotation. Transformation in Eq. (4.23) can be written as:

$$\boldsymbol{\tau} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \in SE(3). \tag{4.24}$$

As $\boldsymbol{\tau}$ only has 6 degrees of freedom, we also introduce a compact representation $\boldsymbol{\varsigma} = (\mathbf{t}^T, \mathbf{q}^T) \in \mathbb{R}^6$, where $\mathbf{t}$ is the translation and $\mathbf{q}$ is the vector part of a unit quaternion $\tilde{\mathbf{q}}$.

The motion of the scene is embedded in a rigid motion field $\mathcal{T}$,

$$\mathcal{T} = \{\boldsymbol{\tau}_k \mid \boldsymbol{\tau}_k \in SE(3), k = 1, \ldots, K\}, \tag{4.25}$$

and the objective is to find $\mathcal{T}^*$:

$$\mathcal{T}^* = \arg\min_{\mathcal{T}} E(\mathcal{T}), \tag{4.26}$$

which minimizes the energy $E(\mathcal{T})$:

$$E(\mathcal{T}) = -\ln \phi(\mathcal{T}). \tag{4.27}$$

Here, $\phi(\mathcal{T})$ is the function parameterizing the factor graph $G = (\Phi, \mathcal{T}, \mathcal{E})$. The graph has two node types: factor nodes $\phi \in \Phi$ and state variables nodes $\boldsymbol{\tau}_k \in \mathcal{T}$. Here, $\mathcal{E}$ is the set of edges connecting a pair of state variable nodes through a factor node. Each edge defines a constraint between the state variable nodes it is connecting with a factor node. Figure 4.6 shows two different factor graph representations for our problem, where $\{\phi_d, \phi_s\} \in \Phi$ are two types of factor nodes describing the energy potentials for the data term (purple squares) and smoothnes term (green squares) respectively. The difference between the two factor graphs is discussed in Section 4.3.1.1. The data term minimizes the Euclidean distance between corresponding keypoints and the smoothnes term, spreads the estimated motion to all the points in the scan, resulting in a dense smooth motion field.

4.3.1.1 Data Term

Our approach relies on the assumption of geometric constancy, i.e. that the local geometric structure is not deformed due to the motion. We parametrize the local

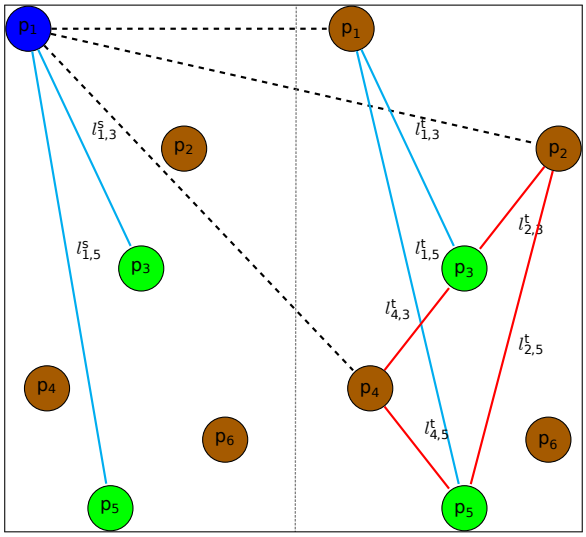


Figure 4.5: An illustration of keypoint matching using the proposed approach. The gray line in the middle separates keypoints in the source scan (left) from the target scan (right).$\mathbf{p}_1$ is matched to three keypoints, $\mathbf{p}'_1$, $\mathbf{p}'_2$ and $\mathbf{p}'_4$ in the target scan (black dotted lines) and the objective is to find the most similar keypoint among them, given a set of matched keypoints $M = \{\langle \mathbf{p}_3, \mathbf{p}'_3 \rangle, \langle \mathbf{p}_5, \mathbf{p}'_5 \rangle\}$ shown in green color. For each pair of matched keypoints, the set L containing Euclidean distances is built and the set that maximizes the posterior $p(m_i \mid L)$ is considered similar. Among the three keypoint pairs, the maximum posterior value will be for the pair $\mathbf{p}_1$ and $\mathbf{p}'_1$ since the Euclidean distances in their set $L = \{\langle l^s_{1,3}, l^t_{1,3} \rangle, \langle l^s_{1,6}, l^t_{1,6} \rangle\}$ have the most consistent geometric relation (indicated by blue lines) in comparison to the geometric relation for the other pairs (red lines).

geometry using a local feature descriptor. Unlike other methods [18, 50], which make similar constancy assumptions, we do not consider the geometric structure explicitly in the optimization, because gradients w.r.t. the geometric structure are not well behaved due to the sparsity of LiDAR data. Instead, we exploit our constancy assumption by matching points in the scan that have a similar local geometric description.

4.3.1.1.1 Estimating Corresponding Keypoints To estimate pointwise motion, our approach solely relies on the matched keypoints and therefore having robust data association is of critical importance. In our case, we are trying to solve a partial correspondence problem because the same set of keypoints are not observed in both the source and target scans, due to the sensor motion and presence of dynamic objects.

The main steps involved in finding data association between keypoints are: keypoint detection, estimating feature descriptors for the keypoints and matching the descriptors. For keypoint detection, we use uniform sampling [94]. After that, we estimate the feature descriptors for the keypoints in source and target scans. A

naive way for matching corresponding keypoints using the descriptors is to calculate the distance of a descriptor in the source scan to descriptors for all the keypoints in the target scan and choose the one with the minimum distance. The results are often inaccurate in this case for two main reasons. Firstly, a keypoint similar to the query keypoint might not exist in the target scan because of the changes in the environment and therefore the matched keypoint could be geometrically different. The observed change in the environment is mainly due to the motion of the sensor and dynamic objects. The second reason is that a matched keypoint with a high similarity score could still be a wrong correspondence, since there are often repeated structures in the scene, for instance, multiple similar looking poles on the side of a street.

In the method proposed for detection and tracking of dynamic objects, we used two heuristics to filter out false correspondences (Section 4.2.2.1). For a keypoint in the source scan we choose a keypoint in the target scan with a minimum descriptor distance, thereby limiting false data association when the keypoint is not observed in the target frame. Secondly, if the distance between matched keypoints exceeds a threshold, the correspondence was discarded.

Building upon these heuristics, we introduce an approach for estimating robust data association between keypoints in two LiDAR scans. The objective is to find a set of matching keypoints that results in minimum local surface distortion. Leveraging over our assumption of local rigidity, we consider a surface to be distorted if the rigidity constraints are violated. To achieve this objective, we first define a geometric relation between neighboring keypoints in the source scan and then find a set of matching keypoints in the target scan with the most similar geometric relation. Since our definition of surface distortion is based on the assumption of rigidity, we use the Euclidean distance between the neighboring keypoints as the geometric relation.

Similar to [109], we implement this approach by estimating ($P(M_i = 1 \mid L)$), the probability of a point $\mathbf{p}_i$ in the source scan and $\mathbf{p}'_j$ in the target scan being similar given a set of Euclidean distances $L = \{\langle l^s_{i,1}, l^t_{j,1}\rangle \cdots \langle l^s_{i,n}, l^t_{j,n}\rangle\}$: Here M_i is a Bernoulli distributed random variable and $M_i = 1$ means that a pair of keypoints are similar. To build the set L, we assume a set of matched keypoints $\mathrm{Y} = \{\langle \mathbf{p}_1, \mathbf{p}'_1\rangle \cdots \langle \mathbf{p}_n, \mathbf{p}'_n\rangle\}$ exists and use it for defining the geometric relation. $l^s_{i,k}$ is the Euclidean distance between a point $\mathbf{p}_i$ and a keypoint $\mathbf{p}_k$ in its local neighborhood and similarly $l^t_{j,k}$ is the Euclidean distance between a point $\mathbf{p}'_j$ and $\mathbf{p}'_k$, where $\langle \mathbf{p}_k, \mathbf{p}'_k\rangle$ is an element of the set Y.

In Eq. (4.28), we use Bayes rule to simplify the expression $P(M_i \mid L)$, where $P(M_i)$ is the prior probability of a matched correspondence being correct, which in our case is the feature matching score. The Eq. (4.29) shows the likelihood function, where n is the number of keypoints used for defining the geometric relation. Similar to [109], we also assume the measurements from different keypoints are independent and

distribution of the error is Gaussian.

$$P(M_i \mid L) \propto P(L \mid M_i)P(M_i) \tag{4.28}$$

$$P(L \mid M_i) = \frac{1}{\sigma^n (2\pi)^{n/2}} \prod_{k=1}^{n} \exp(-\frac{(l_{i,k}^s - l_{j,k}^t)^2}{2\sigma^2}) \tag{4.29}$$

Our approach consists of multiple steps:

1. Find a maximum of γ matching keypoints in a target scan for every keypoint in the source scan and sort them according to their matching score. In this step, we only include a pair of keypoints if their matching score is greater than a threshold.

2. Choose a small set of matched keypoints with the highest matching score as an initial seed and add them to the set Y. The keypoint pairs in this set are the final matched keypoints and will be used for defining the required geometric relation, as discussed above.

3. To find remaining matching keypoints, we estimate the posterior value as described in Eq. (4.28) for all the keypoints whose neighborhood contains keypoints from the set Y. In this step, for a given keypoint pair, we construct the set L using the keypoints in the set Y, where only the neighboring keypoints from Y are used.

4. The matching keypoint pair with the maximum posterior value is added to the set Y. In the case where none of the unmatched keypoint pairs is in proximity to the keypoint pairs in Y, we repeat step two and add the keypoint pair with the highest matching score, if the score is greater than a certain threshold.

Figure 4.5 illustrates the above mentioned steps. In Section 4.3.2, we show the qualitative results and also report the quantitative results indicating the decrease in the error of estimated motion after using this approach for keypoint matching.

4.3.1.1.2 Error Function The objective of the data term is to estimate a rigid motion model by minimizing the Euclidean distance between the corresponding keypoints. We propose two different factor graph representations to formulate this optimization problem (Figure 4.6). In the first case, we define the data term error in the following way:

$$\mathbf{e}_d(\boldsymbol{\tau}_i) = \boldsymbol{\tau}_i \tilde{\mathbf{p}}_i - \tilde{\mathbf{p}}'_i. \tag{4.30}$$

The data term is defined as:

$$\phi_d(\boldsymbol{\tau}_i) = \exp(-\|\mathbf{e}_d(\boldsymbol{\tau}_i)\|^2). \tag{4.31}$$

Here, $\tilde{\mathbf{p}}_i$ and $\tilde{\mathbf{p}}'_i$ are the corresponding points in consecutive scans represented in homogeneous coordinates, $\boldsymbol{\tau}_i \in SE(3)$ is the rigid motion transformation and $i \in I_d$,

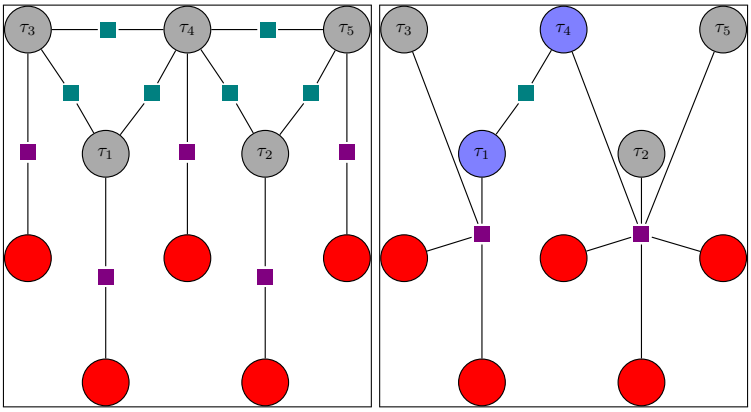


Figure 4.6: Different factor graph representations. The gray and red nodes correspond to matched keypoints in the source and target scans respectively. The purple squares represent the factor for the data term and green squares represent the factor for the pairwise term. For the factor graph on left, a data term is defined for every matched keypoint and a smoothnes term connecting a pair of neighboring keypoints in the source scan. The data term tries to estimate a rigid motion defined in $\mathbb{R}^6$ by only using its own measurement (estimated correspondence) defined in $\mathbb{R}^3$. The difference in dimensionality makes the optimization problem under-constrained. In contrast to this, in the factor graph on the right, the data term is defined for a subset of matched keypoints called deformation variables (shown in blue). In this case, a deformation variable incorporates multiple measurements including its own. In the right image, the data term corresponding to τ_1 will use its own measurement and the measurement for the keypoint corresponding to τ_3. Using multiple measurements makes the optimization problem well posed in comparison to the other representation. The smoothnes term is now defined between a pair of neighboring keypoints selected as deformation variables.

where I_d is the set of indices corresponding to keypoints in P_{t-1}. A rigid motion variable $\boldsymbol{\tau}_i$ has six DoF and for estimating the motion, we only use a single measurement, which in our case is the difference between the corresponding keypoints defined in $\mathbb{R}^3$. The lower dimensionality of the measurement makes our non-linear equation system under-constrained. The factor graph on the left in Figure 4.6 illustrates this representation, where the factor (purple square) corresponding to the data term only connects a single pair of matched keypoints.

To improve the well-posedness of our optimization problem, we propose a second factor graph representation. Instead of defining a data term for each keypoint, we define a data term for a subset of keypoints and call such data terms as deformation variables, $I_{dv} \subset I_d$. For each such keypoint, within the error function, we include its own measurement and measurements of its neighboring keypoints, where the set for possible neighboring keypoints is $I_d - I_{dv}$. The factor graph on the right in Figure 4.6 illustrates this representation. The factor for the data term (purple square) connects multiple corresponding keypoints rather than a single pair of corresponding keypoints. The method to choose the neighboring keypoints is

described in Section 4.3.1.2.

The error function for the data term for this factor graph representation is defined as the following:

$$\mathbf{e}_d(\boldsymbol{\tau}_i) = \sum_{k \in N_{dv}} \boldsymbol{\tau}_i \tilde{\mathbf{p}}_k - \tilde{\mathbf{p}}'_k, \tag{4.32}$$

where, N_{dv} is a set of neighboring keypoints for a given deformation variable. For any two deformation variables, the intersection of their neighboring keypoint sets is the empty set.

4.3.1.2 Smoothnes Term

The data term helps to estimate the motion, but is defined only for a subset of points in a scan. Therefore, we include a smoothnes term in our energy function, to obtain a dense, locally smooth motion field by spreading the motion from keypoints to nearby points. The factor for the smoothnes term is represented using a green square in Figure 4.6. They are defined between neighboring points in the scan, irrespective of whether they are keypoints or not.

A similar smoothnes term is often included in energy minimization problems [18, 50, 86]. In the case of estimating a motion field in an image domain, the local neighborhood is well understood. A common practice is to consider all pixels in a small image patch as neighbors. In the case of 3D pointcloud data, the concept of neighborhood is not as straightforward as for images. A commonly used approach to calculate neighboring points in the case of pointclouds is to consider all the points in a sphere around a point as neighbors. Since our method relies on the assumption of rigidity of the local structure, it is important that the definition of neighborhood considers the structure of the scene, which spherical neighborhood fails to do. Instead, we construct a triangular mesh structure [71] to approximate the surface and consider points as neighbors only if they are vertices of the same triangle. This technique is also used for estimating neighboring keypoints of a deformation variable discussed in the previous section.

4.3.1.2.1 Error function The error function for the smoothnes term is defined in the following way:

$$\mathbf{e}_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j) = \xi(\boldsymbol{\tau}_i^{-1} \boldsymbol{\tau}_j) \tag{4.33}$$

where, $\boldsymbol{\tau}_i$ and $\boldsymbol{\tau}_j$ are the rigid motion transformations for neighboring nodes and $\xi(\cdot)$ is the mapping function from $SE(3)$ to a compact representation in $\mathbb{R}^6$. This error term forces neighboring points to have similar motion, thereby enforcing the local rigidity constraint, spreading the motion from keypoints to other points in the scan and additionally making the problem well-posed. The energy potential for the regularization term is defined as:

$$\phi_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j) = \exp(-\|\mathbf{e}_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)\|^2) \tag{4.34}$$

In Figure 4.6, green squares connecting neighboring state variable nodes represent $\phi_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)$. In the first graph, all neighboring nodes (keypoints and non-keypoints) can be connected through the regularization term. In the second case, a smoothnes term is defined between all the nodes except between nodes corresponding to I_{dv} and nodes in the set $I_d - I_{dv}$. Since nodes in the second set are already connected to I_{dv} through the data term, connecting them again through smoothnes term will not add any new information.

4.3.1.3 Optimization

The two factor graph representations proposed in Section 4.3.1.1 represents the factorization of the following functions:

$$\phi(\mathcal{T}) = \prod_{i \in I_d} \phi_d(\boldsymbol{\tau}_i) \prod_{l \in N_p} \phi_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j), \tag{4.35}$$

$$\phi(\mathcal{T}) = \prod_{i \in I_{dv}} \phi_d(\boldsymbol{\tau}_i) \prod_{l \in N_p} \phi_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j), \tag{4.36}$$

where, $N_p = \{\langle 1,2\rangle, \langle 2,3\rangle, \ldots, \langle i,j\rangle\}$ is the set containing indices of neighboring vertices. The only difference between Eq. (4.35) and (4.36) is that the energy for the data term ϕ_d in Eq. (4.35) is defined for all the keypoints, in contrast to Eq. (4.36), where it is only defined for a subset. As discussed in Section 4.3.1.2, one of the contributions of the smoothnes term is that it enforces local rigidity constraint. To regulate this rigidity constraint, we modify the Eq. (4.35) by introducing a weighting factor β as shown in Eq. (4.37). A large value of β will enforce a strong rigidity constraint i.e. the difference between the estimated motion of neighboring points will decrease, while a small value of β will weaken this constraint and will have an opposite effect. In the same way, Eq. (4.36) can be modified to incorporate the weighting factor β as well.

$$\phi(\mathcal{T}) = \prod_{i \in I_d} \phi_d(\boldsymbol{\tau}_i) \prod_{l \in N_p} \beta \cdot \phi_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j). \tag{4.37}$$

The error for both data and regularization terms is quadratic and the problem is of sparse non-linear least square form. Using Eq. (4.31), (4.34) and (4.37), for the first factor graph representation, we can simplify Eq. (4.27) as:

$$E(\mathcal{T}) = \sum_{i \in I_d} \|\mathbf{e}_d(\boldsymbol{\tau}_i)\|^2 + \sum_{l \in N_p} \beta \cdot \|\mathbf{e}_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)\|^2. \tag{4.38}$$

Similarly for the second factor graph representation we can simplify Eq. (4.27) as:

$$E(\mathcal{T}) = \sum_{i \in I_{dv}} \|\mathbf{e}_d(\boldsymbol{\tau}_i)\|^2 + \sum_{l \in N_p} \beta \cdot \|\mathbf{e}_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)\|^2. \tag{4.39}$$

To solve this, we use graph optimization. A general framework for graph optimization involves constructing the graph i.e. defining the parameters associated with each node and the constraints between the nodes. Section 4.3.1.1 discusses the construction of the proposed graphs used for optimization. In general for graph optimization, a vector error function $\mathbf{e}_{ij}(\mathbf{x})$ is defined for a pair of nodes in the graph and the constraint between nodes i and j is the difference between the expected measurement $\hat{\mathbf{z}}(\mathbf{x}_i, \mathbf{x}_j)$ and the observed measurement $\mathbf{z}_{ij}$.

$$\mathbf{e}_{ij}(\mathbf{x}) = \hat{\mathbf{z}}(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{z}_{ij}, \tag{4.40}$$

where, $\mathbf{x}$ is the parameter block associated with each node. In our case, the parameter block is comprised of the motion $\boldsymbol{\tau}$. For the data term (Eq. (4.30) and (4.32)), $\mathbf{e}_{ij}$ is defined in $\mathbb{R}^3$, the observed measurement is the corresponding point $\mathbf{p}'$ and the expected measurement is the location of the keypoint $\mathbf{p}$ after being transformed by $\boldsymbol{\tau}$. For smoothnes term (Eq. (4.33)), $\mathbf{e}_{ij}$ is defined in $\mathbb{R}^6$, the observed measurement is an *identity* matrix and the expected measurement is difference between $\boldsymbol{\tau}$ for a pair of neighboring nodes.

The error is calculated in a non-Euclidean manifold space [40] and therefore Eq. (4.40) can be reformulated as:

$$\mathbf{e}_{ij}(\mathbf{x}) = \hat{\mathbf{z}}(\mathbf{x}_i, \mathbf{x}_j) \ominus \mathbf{z}_{ij}, \tag{4.41}$$

where, the difference in the non-Euclidean space is computed using the $\ominus$ operator.

The error minimization function can be formulated as,

$$\mathbf{x}^* = \arg\min \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^T \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x}), \tag{4.42}$$

where, Σ_{ij} and Σ_{ij}^{-1} are covariance and information matrix respectively, and $\mathbf{x}^*$ is the optimal parameter configuration, which satisfies the constraints between the nodes. When $\Sigma_{ij} = I$, Eq. (4.42) has same formulation as Eq. (4.38) and Eq. (4.39) because in all the cases the objective is to minimize the sum of squared error.

The numerical solution of Eq. (4.42) can be calculated using second-order optimization algorithms like Gauss-Newton or Levenberg-Marquardt by approximating the error function by its first order Taylor expansion around the current initial guess $\check{\mathbf{x}}$,

$$\mathbf{e}_{ij}(\check{\mathbf{x}} + \Delta\mathbf{x}) \simeq \mathbf{e}_{ij}(\check{\mathbf{x}}) + \mathbf{J}_{ij}\Delta\mathbf{x}, \tag{4.43}$$

where the jacobian $\mathbf{J}$ is defined as:

$$\mathbf{J}_{ij} = \left. \frac{\partial \mathbf{e}_{ij}(\check{\mathbf{x}} + \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} \right|_{\Delta\mathbf{x}=0}. \tag{4.44}$$

The Jacobians corresponding to all the constraints can be stacked together in a matrix form and the minimization problem can be efficiently solved by solving this

system of linear equations:

$$\mathbf{H}\Delta\mathbf{x}^* = -\mathbf{b}, \tag{4.45}$$

where, $\mathbf{H} = \Sigma_{ij}\mathbf{J}_{ij}^T\Sigma_{ij}^{-1}\mathbf{J}_{ij}$ is the Hessian matrix of the system and $\mathbf{b} = \Sigma_{ij}\mathbf{J}_{ij}^T\Sigma_{ij}^{-1}e_{ij}$. To obtain the solution, increments $\Delta\mathbf{x}^*$ are added to the initial guess,

$$\mathbf{x}^* = \check{\mathbf{x}} + \Delta\mathbf{x}^*. \tag{4.46}$$

For the derivation of Eq. (4.45) and an elaborate explanation for different operations regarding optimization on a manifold, please consult [40] and [63].

We use the Levenberg-Marquardt algorithm to find an estimate that minimizes the energy. Since the regularization term only connects neighboring points, the problem can be decomposed into multiple independent sub-problems, which can be solved efficiently in parallel.

The correspondences we estimate are plagued by outliers, which have a strong negative impact on the estimate. To tackle outliers, we use a saturated robust kernel ρ,

$$\rho(x^2) = \left\{ \begin{array}{ll} \frac{x^2}{2}, & \text{if } x^2 \leq c^2 \\ \\ \frac{c^2}{2}, & \text{if } x^2 > c^2 \end{array} \right\}, \tag{4.47}$$

where, x^2 is the squared error and c is the kernel size. We use a robust kernel only for the error in the data term and rewrite Eq. (4.38) as:

$$E(\mathcal{T}) = \sum_{i \in I_d} \rho(\|\mathbf{e}_d(\boldsymbol{\tau}_i)\|^2) + \sum_{l \in N_p} \beta \cdot \|\mathbf{e}_s(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)\|^2 \tag{4.48}$$

The robust kernel is also used for the second factor graph representation and incorporating it for Eq. (4.39) is as discussed above. Since the initial error is large, we perform two steps of optimization. First, we optimize without a robust kernel to reduce the error and then perform another optimization run with the robust kernel to minimize the effect of outliers.

As we use sequential data, the motion transformations can also be expected to be temporally smooth. We do not include a term for temporal smoothing explicitly in our energy minimization function, but we initialize the estimate for a vertex in the graph with the estimates from previous scans. Points with known motion are transformed into the frame of reference of the next scan. We perform data association between the transformed points and the points in the next scan on the basis of Euclidean distance and propagate the estimated motion. Since we do not have data association for each point, we still perform two steps of optimization to make sure that state variable nodes without good initial estimates are not treated as outliers due to a large initial error.

4.3.2 Results

We evaluate our approach on three different datasets. The first one is a simulated dataset from Yoon et al. [129], where the simulation environment is an urban outdoor scenario with multiple dynamic objects. The dataset consists of two different sequences and the ground-truth poses for the sensor and dynamic objects are available. Since the environment contains multiple dynamic objects, the scene is non-rigid and thus provides us an opportunity to highlight the advantages of our approach. The next dataset we use is the KITTI odometry benchmark and it is also based on urban outdoor scenarios. We evaluate our approach on five sequences from the benchmark. Estimating the ground-truth motion for each point or for every moving object is labor intensive and non-trivial, but if the scene only contains static structure, the motion of each point is the inverse sensor motion. Therefore, to evaluate the accuracy of the estimated motion, among the five sequences, we choose three sequences (3, 5 and 6) comprising only of static structure, while the remaining sequences contains multiple dynamic objects. These urban outdoor environment datasets (simulated and real) mainly comprise of non-rigid scenes but objects like vehicles are rigid. Focusing on non-rigid objects, we also evaluate our method on a dataset with pedestrians. The dataset contains multiple sequences with different body movements and the objective of this experiment is to highlight that our method is capable of capturing fine dynamic details and can estimate motion of non-rigid bodies as well.

For quantitative evaluation of the accuracy of the estimated motion, we report the average error in translation and rotation motion, and the alignment accuracy. The error for the estimated motion is only reported for the cases where the ground-truth motion is available. This is true for the simulated dataset and the three sequences from the KITTI dataset. For measuring alignment accuracy, we use the *crispness score* C_s [48] [100] as a metric. This score is defined as following:

$$C_s = \frac{1}{n} \sum_{k=1}^{n} G(\mathbf{p}'_k - \hat{\mathbf{p}}_k, 2\Sigma), \tag{4.49}$$

where, $G(\mu, \Sigma)$ is a Gaussian with mean μ and covariance Σ, $\hat{\mathbf{p}}_k$ is the nearest point in the point set P_t to the transformed point $\mathbf{p}'_k$ from the point set P_{t-1} and n is the number of points. The crispness score defines the compactness of two point sets by measuring the kernel correlation [113] between the two point sets and in this case, we use a Gaussian kernel. We have scaled the scores between 0 and 1, and a score close to 1 indicates that the two point sets are aligned. For qualitative evaluation, we show the estimated motion field after subtracting the sensor motion. The color wheel in Figure 4.7 illustrates our representation of the motion field. The magnitude and the direction of the estimated motion are represented by saturation and hue values respectively. With an increase in magnitude, the saturation value increases from the center to the circumference of the color wheel. The blue and red

lines indicate the positive z and x directions respectively.

For all the datasets, we compare *our* method with the Iterative Closest Point (ICP) algorithm, where *our* method uses the second factor graph representation (Section 4.3.1.1), and the method for estimating robust correspondences (Section 4.3.1.1.1), giving higher importance to the smoothnes term by choosing $\beta = 1000$ (Section 4.3.1.3). As a part, of an ablation study, we also present a comparison between the two proposed factor graph representations and report results to analyze the affect of our method for estimating robust correspondences between the keypoints and the weighting factor between the data and smoothnes term. For comparing the two representations, we replace the second factor graph representation with the first (fg1), while keeping the other choices the same. To validate our method of estimating corresponding keypoints, we report results for an experiment where we do not filter correspondences causing surface distortion (w/o c.f), but select the correspondences with the highest matching score. We also report results for an experiment where both data and smoothnes terms have equal weights ($\beta = 1$). For both these experiments, we use the second factor graph (fg2) representation. For all of these experiments, we use the SHOT local feature descriptor [112] for matching keypoints. We also report results for an experiment where we replace the SHOT descriptor with our learned feature descriptor from Chapter 3 for matching keypoints in two consecutive scans.

We employ the g2o toolkit [63] for optimization. For all of the experiments, we remove all of the ground points in a preprocessing step. In the case of experiments using the KITTI odometry dataset and the simulated dataset, we chose $c^2 = 0.05$ as kernel size (Section 4.3.1.3) and for the dataset with pedestrians we chose $c^2 = 0.01$. This choice depends on the maximum error that can be tolerated. The difference in value of c^2 between the two datasets is mainly attributed to the difference in the maximum possible motion between consecutive scans, which in turn affects the maximum error. Datasets comprising of urban outdoor environments consist of vehicles moving at speeds significantly higher than the speed at which different body parts are moved in the case of the dataset with pedestrians.

4.3.2.1 KITTI Odometry Dataset

In Table 4.3, 4.4 and 4.5, we report the error in translation and rotation motion, and the alignment accuracy, respectively. Figure 4.8 illustrates the estimated motion field for our method. The colorization of the pointcloud is as per to the color wheel in Figure 4.7. The color of the dynamic objects moving in opposite directions are opposite in the color wheel as well, which indicates the motion is estimated correctly for these objects and similarly static structures like walls on the sides have black color, which indicates the magnitude of their estimated motion after subtracting the sensor motion is close to zero.

The objective of this experiment is to evaluate the accuracy of the estimated

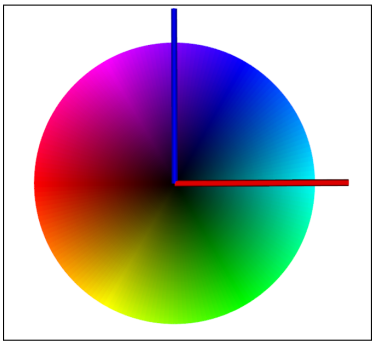


Figure 4.7: The color wheel indicating the magnitude (saturation) and the direction (hue) for the translational motion. Motion with low magnitude is closer to the center of the wheel and motion with large magnitude is closer to the circumference of the circle. The z and x direction of the motion is represented by the blue and red lines respectively.

motion and we use ICP algorithm as a baseline for comparison. Considering the error in translation motion (Table 4.3), among the three sequences, for the first sequence (seq. 3), our method outperforms ICP, for the second sequence (seq. 5), performance for ICP is slightly better and for the last sequence, ICP significantly outperforms our method. The increase in the error for the last sequence is mainly because points in this sequence are on average farther away in comparison to the other sequences. With the increase in distance, the data gets sparser, which in-turn affects the quality of the estimated correspondences resulting in a decrease in the accuracy of the estimated motion. These results show that our proposed method is capable of achieving accuracy similar to ICP, with the limitation where the error increases in the case when pointclouds are sparse. This scenario with only static structure is not a proper use case for our approach, since the main contribution of our approach is the capability of estimating multiple motion models. In these cases, the motion between scans can be explained solely by the sensor motion and for estimating that a method like ICP should suffice.

Comparing the different factor graph representations, the one where only a subset of keypoints are used, attains lower motion error (both translation and rotation) in comparison to the case where all keypoints are used. This shows that defining data term only for a subset of keypoints makes our non-linear system of equations less under-constrained, thereby improving well-posedness of the optimization problem in comparison to the first factor graph representation. Figure 4.10 compares the motion field estimated after using both representations.

Our proposed approach for estimating correspondences between keypoints also contributes towards reducing the error. The measurement for motion solely relies on the estimated correspondences and, therefore, having a method that discards possible false correspondences by selecting a set of correspondences that minimizes the surface distortion helps in estimating motion accurately. Figure 4.11 illustrates

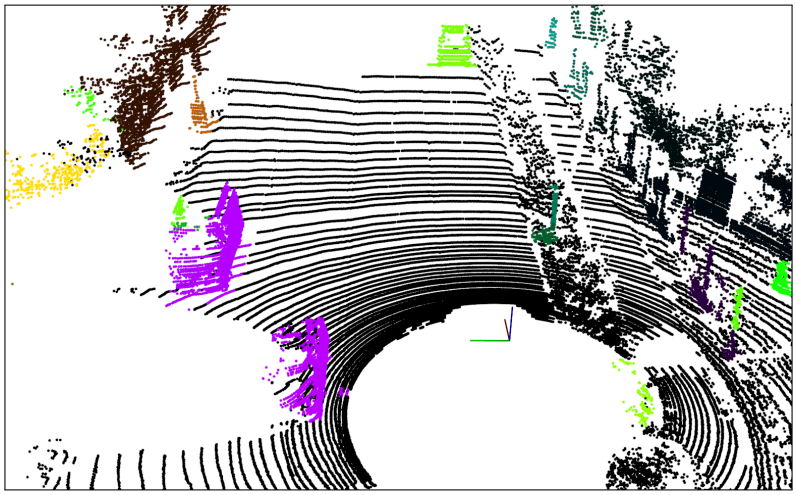


Figure 4.8: Visualization of the motion field estimated by our method. Points are colorized according to the color wheel (Figure 4.7). Points belonging to the static structures like walls, have a color similar to the center of the color wheel (black), which indicates that after subtracting the sensor motion, the estimated motion is close to zero. The two dynamic objects (purple color) are moving in the same direction and have similar hue value and the third dynamic object (green color) is moving in the opposite direction. These colors are opposite in the color wheel and also along the circumference, thereby illustrating that both the direction and magnitude of the motion is correctly estimated.

the filtering of false correspondences between the keypoints and the decrease in the error of the estimated motion through visualization of the motion field.

A larger weight for the error contributed by smoothnes term also helps in reducing the error in motion because this forces nearby points to have similar motion, which also improves the well-posedness of our optimization problem and makes the estimated motion more accurate. Figure 4.12 illustrates the comparison of the motion field when both the data and smoothnes terms are weighted equally and the case when the error contributed by the smoothnes term has higher importance.

In Figure 4.9a, 4.9b & 4.9c, we show the distribution of the motion error for our approach, for different sequences. For all of the sequences, the error for the majority of the scans is centered around the mean reported in Table 4.3 & 4.4 with some outliers. The largest range in error is for the sequence 6 (Figure 4.9c). The explanation for large error for this sequence mentioned before also explains this large range.

Figure 4.13 illustrates a pair of LiDAR scans aligned using our proposed method. Comparing the alignment accuracy (Table 4.5), our method outperforms ICP for all the cases. The interesting cases are sequences 4 and 12 because they contain multiple dynamic objects. In contrast to sequences 3, 5 and 6, these sequences are a proper use case of our approach. The large difference in accuracy for sequences

Table 4.3: Translation error for KITTI odometry (meters)

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Seq. 3	0.172±0.126	0.169±0.092	0.131±0.062	0.150±0.077	**0.121±0.067**
Seq. 5	**0.126±0.083**	0.192±0.078	0.150±0.064	0.173±0.073	0.137±0.062
Seq. 6	**0.103±0.065**	0.409±0.111	0.389±0.132	0.390±0.121	0.321±0.114

Table 4.4: Rotation error for KITTI odometry (radians)

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Seq. 3	**0.010±0.011**	0.023±0.013	0.023±0.014	0.028±0.016	0.021±0.015
Seq. 5	**0.012±0.017**	0.029±0.015	0.028±0.017	0.032±0.017	0.024±0.017
Seq. 6	**0.010±0.022**	0.04±0.02	0.052±0.023	0.052±0.031	0.041±0.031

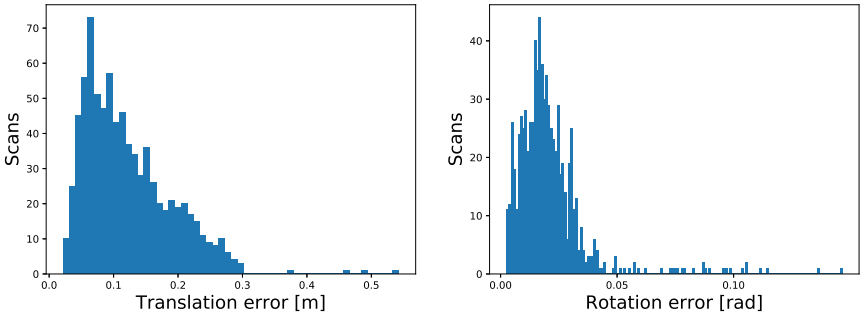
with dynamic objects, highlights the contribution of our proposed approach.

Comparing the two factor graph representations, in contrast to the error in motion, the improvement in the estimated motion due to the second factor graph representation and the weighting of the smoothnes term is not apparent in the alignment accuracy. Consecutive scans can be aligned accurately by using either the first factor graph representation or by equally weighting the data and the smoothnes term even though the estimated motion has a larger error in comparison to other methods. This is primarily because our system of equations is under-constrained and, in this case, many solutions are possible and, therefore, the objective of reducing the Euclidean distance between the corresponding keypoints (data term) can be achieved by different solutions. For all of the cases, alignment accuracy decreases when the correspondences between the keypoints are not filtered out using our proposed method. As mentioned before, the matched keypoints are the only source of measurements for estimating the motion and therefore having a small set of false measurements affects both the motion error and the alignment accuracy.

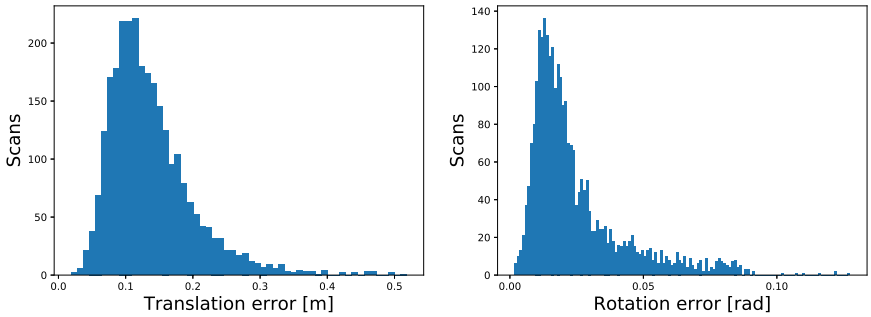
4.3.2.2 Simulated Dataset

For the next set of experiments, we use a simulated dataset, comprised of two sequences (Town 1 and 2). Similar to KITTI, this dataset only consists of urban outdoor scenes. Figure 4.14 illustrates a simulated LiDAR scan. The scanner used in the simulation has the same number of channels as the scanner used in the KITTI dataset. The most noticeable difference between the simulated and the real data is the rendering of the movable objects. The shape of these objects only consists of planar structures, which makes matching of keypoints a challenging problem due to presence of limited distinctive shape features.

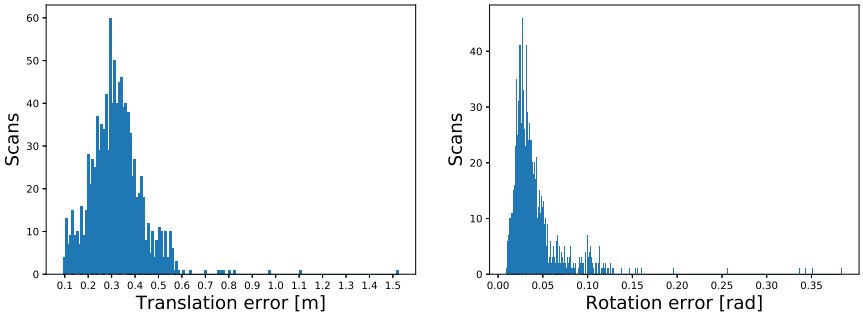
For evaluating this dataset, we repeat the evaluation done for the KITTI dataset. The main motivation of using this dataset is the availability of the ground-truth



(a) Distribution of error in the estimated motion for Seq. 3.



(b) Distribution of error in the estimated motion for Seq. 5.



(c) Distribution of error in the estimated motion for Seq. 6.

Figure 4.9

position for movable objects, which in this case is vehicles. We split the evaluation of error in motion into two cases. The first evaluation is for non-movable objects like buildings or vegetation. The motion observed for points belonging to this class is solely because of the sensor motion. We evaluate the accuracy of the estimated motion for this case by comparing the estimated motion with the odometry. The benchmark does not provide the ground-truth odometry but instead the ground-truth sensor position, and we estimate the sensor egomotion by calculating the

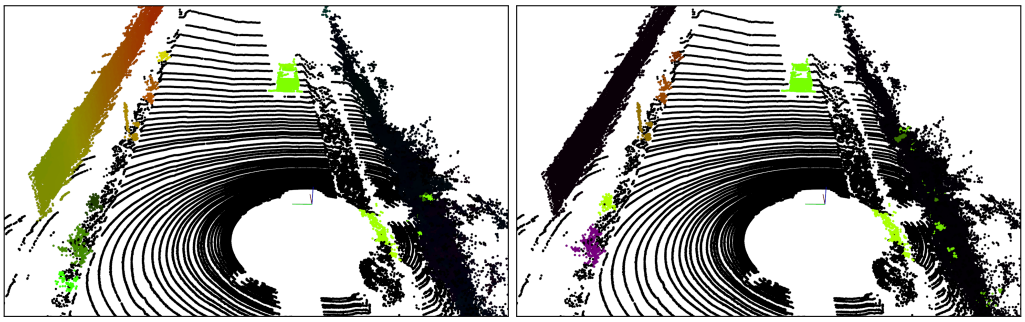


Figure 4.10: Comparison of the estimated motion field between two factor graph representations. On the left is the motion field for the first factor graph representation and on the right is the motion field for the second representation. The colorization of the pointcloud is as per the color wheel. For the first representation, the points on the wall on the left have large motion after subtracting the sensor motion from the estimated motion, in comparison to the second representation. This illustrates that by using the second graph representation, motion can be estimated more accurately.

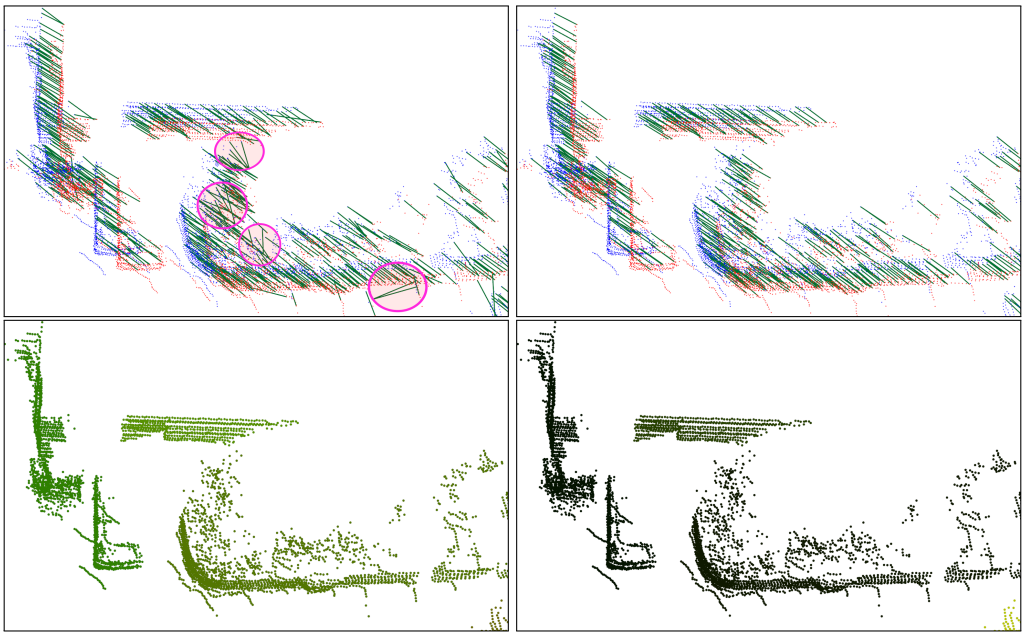


Figure 4.11: Figures in the top row show the estimated feature correspondences prior to using our descriptor matching approach (left) and after using our approach (right). The purple ellipses show the false correspondences that are removed after using our method. Figures in the bottom row shows the estimated motion field for these respective cases. The error in the estimated motion is higher when our feature matching approach is not used (bottom-left) but the error significantly decreases after using our feature matching approach (bottom-right).

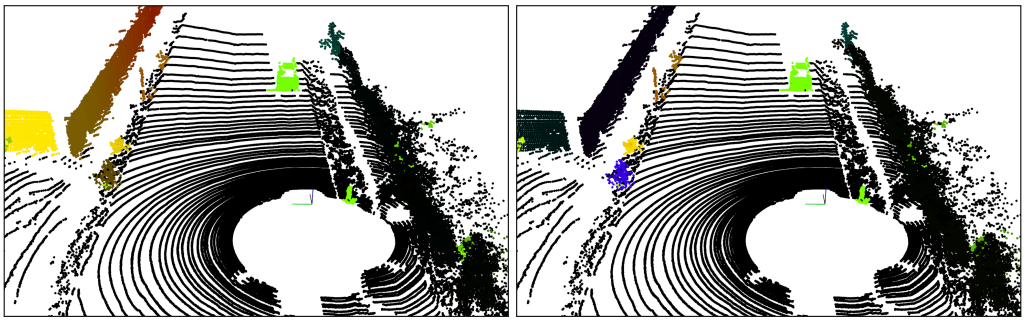


Figure 4.12: Figure on the left shows the estimated motion field when the same weights ($\beta = 1$) are used for the data and the smoothnes term and the figure on the right shows the motion field after a large weight ($\beta = 1000$) is used for the smoothnes term. In the case of same weights, the points on the wall on the left have large motion after subtracting the sensor motion from the estimated motion, in comparison to the case when the error contributed by the smoothnes term is weighted more.

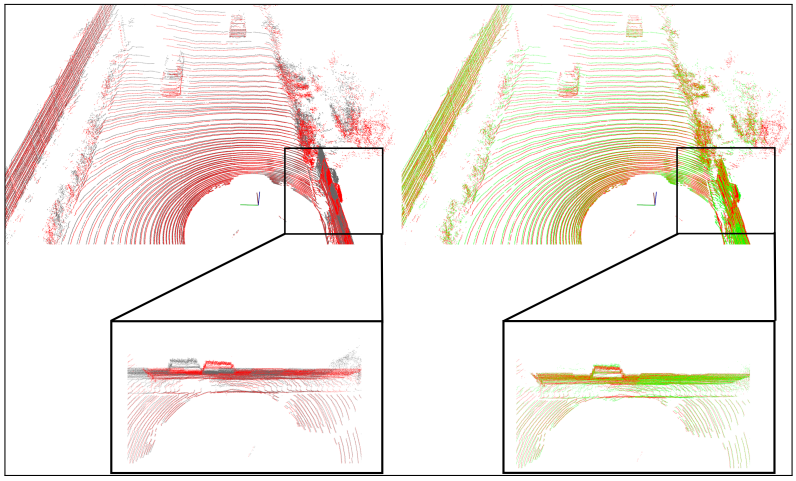


Figure 4.13: Figure on the top-left shows two consecutive misaligned pointclouds and the figure on the top-right shows the aligned pointclouds after using our method. The bottom-left figure shows a zoomed in view of the misaligned wall, which is correctly aligned by our method (bottom-right).

Table 4.5: Alignment accuracy for KITTI odometry

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Seq. 3	0.888±0.011	**0.934±0.020**	0.927±0.023	0.929±0.022	0.931±0.022
Seq. 4	0.754±0.044	**0.901±0.046**	0.880±0.059	0.894±0.050	0.896±0.049
Seq. 5	0.905±0.030	**0.938±0.016**	0.931±0.020	0.933±0.020	0.935±0.018
Seq. 6	0.868±0.025	**0.894±0.020**	0.873±0.032	0.881±0.030	0.887±0.028
Seq. 12	0.630±0.076	0.843±0.078	0.802±0.095	0.840±0.073	**0.847±0.069**

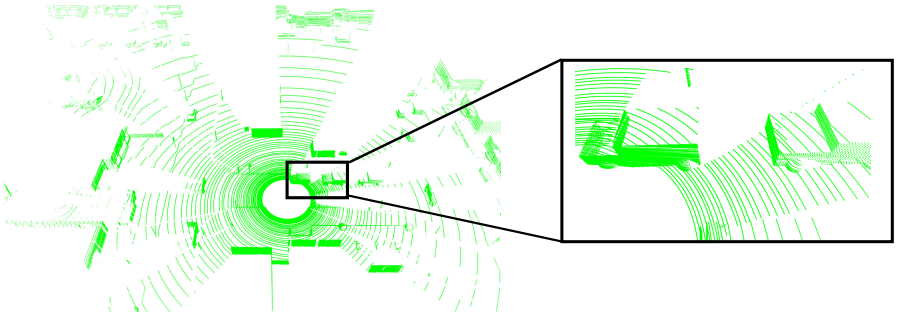


Figure 4.14: An illustration of a simulated LiDAR scan. Similar to the scanner used in the KITTI dataset, the simulated scanner also has 64 channels. In comparison to the real dataset, the noise level in the measurements for simulated data is very limited. The main difference between the real and this simulated dataset is the rendering of the movable objects, which in this case is vehicles (zoomed image). The shape of the objects is only comprised of planar structures and has limited distinctive shape properties.

difference in the positions. In Table 4.6 and 4.7, we report the motion error for the sensor motion. The next evaluation is for movable objects. For the majority of the scans, movable objects have non-zero motion i.e. they are dynamic but for some cases they are static as well. Similar to the first evaluation, we compare the estimated motion for these objects with the difference of the ground-truth position of movable objects in the consecutive scans. In Table 4.8 and 4.9, we report the motion error for the movable objects. In Table 4.10, we report the alignment accuracy for the complete scan i.e. for both movable and non-movable objects together.

Figure 4.15 illustrates the estimated motion field for one of the scans. Similar to results on the KITTI dataset (Figure 4.8), our method is able to estimate the sensor motion (black points) and motion for different dynamic objects (points in green and purple color). Comparing the error in sensor motion, the lowest error in motion for the Town 1 sequence is achieved by our method and for the other sequence, the lowest error is for ICP. The difference in the error between sequences for ICP is because the sensor is often static in the sequence Town 2.. In such cases, the non-occluded parts of the scene have perfect data association (nearest neighbor for ICP) due to limited noise in the simulated data. This is also observed for our

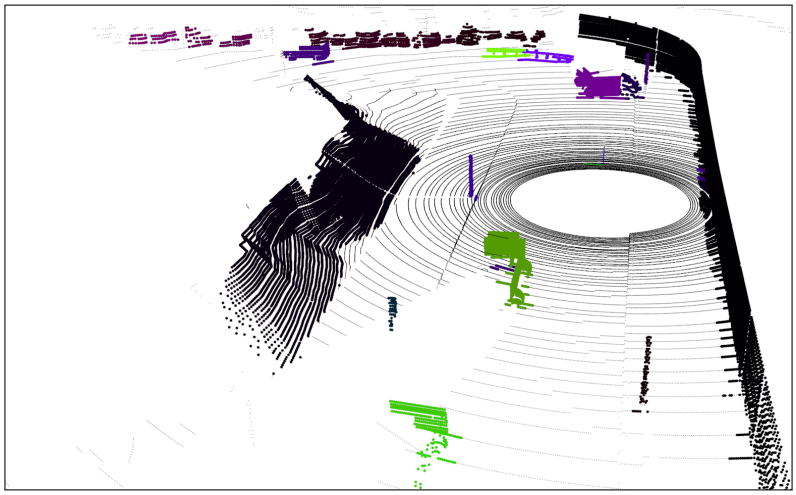


Figure 4.15: Visualization of the estimated motion field for a simulated LiDAR scan. Like other qualitative evaluations, points are colorized as per the color wheel. Static structures, like walls, have colors similar to the center of the color wheel, which indicates that after subtracting the sensor motion, the estimated motion is close to zero. The two dynamic objects behind the scanner (green color) are moving in the opposite direction to the cars moving in the direction of the sensor (purple color). These colors are opposite in the color wheel and also along the circumference, which illustrates that both the direction and magnitude of the motion is estimated correctly.

method, where for all the cases the error is lower for the Town 2 sequence. Similar to the evaluation on KITTI dataset, the error increases after using the first factor graph representation, not filtering the correspondences and equally weighting the smoothnes and the data term. The explanation for this is the same as mentioned in Section 4.3.2.1.

Even though the relative error between different cases for the real and simulated data is similar, the standard deviation in the case of simulated dataset is either larger or comparable to the mean, indicating that the distribution of error is not unimodal. Figure 4.16a and 4.16b shows the histogram for the translation and the rotation error of sensor motion for Town 1 and 2 sequence respectively. For both sequences, the first distribution of error is around zero. This is the case where the sensor is not moving and the data association for non-movable points that are not occluded by movable objects is perfect. As mentioned before, the perfect data association is possible because of limited noise in the simulated data. The next distribution is around the mean reported in Table 4.6. The error for the large majority of the scans are part of these two distributions, while the errors for the few remaining scans are sparsely distributed across the complete range of error.

In comparison to the error in sensor motion, error for movable objects is larger (Table 4.8 and 4.9). This is an expected result because our method relies on estimating

motion through correspondences between keypoints, which in turn requires a large overlap between points in two consecutive scans. Even in the case of a moving sensor or occlusions due to dynamic objects, the overlap of the static structure in consecutive scans is large enough to estimate the motion accurately. In contrast to that, the overlap for movable objects is often limited because such objects are smaller in size in comparison to the static structure and therefore the overlap decreases, especially for the cases when both the sensor and the object have non-zero motion.

For both sequences, the error for our method is lower than for ICP. This is expected, since the objective of ICP is to estimate a single motion model. This result again reiterates our initial claim that our method can seamlessly estimate multiple motion models. Similar to the results reported before, the use of the first factor graph representation, not filtering the correspondences and identically weighting the data and the smoothnes term deteriorates the accuracy of the estimated motion. The standard deviation for the error in case of movable objects is either larger or comparable to the reported mean, similar to the case of sensor motion. In Figure 4.17a and 4.17b, we show the distribution of error for movable objects for both sequences. The range of error is larger in comparison to the error for sensor motion (Figure 4.16a and 4.16b), which supports our initial claim that sensor motion can be estimated more accurately in comparison to the movable objects for aforementioned reasons. The error (both in translation and rotation) for the majority of the scans is around the reported mean, while large error in a few scans pushes the range of error, which results in a large standard deviation.

In the case of alignment accuracy (Table 4.10), our method outperforms ICP. The alignment for static and movable objects are estimated together and due to presence of dynamic movable objects, the accuracy for ICP is lower. The highest accuracy is achieved by using the first factor graph representation, which is marginally better than the second representation. We reported similar results for KITTI as well and explained the reason for this behavior (Section 4.3.2.1). The same explanation holds for this case as well.

Through the evaluation on the simulated dataset, we show that our method is able to estimate the sensor motion and motion of multiple objects accurately. Our method overcomes the limitation of a method like ICP, which only estimates a single rigid motion model, whereas our method estimates multiple rigid motion models and this characteristic makes our method useful in the case where the environment is non-rigid.

Table 4.6: Translation error for sensor motion (meters)

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Town 1	0.122±0.195	0.158±0.225	0.145±0.243	0.146±0.208	**0.111±0.179**
Town 2	**0.044±0.063**	0.134±0.162	0.109±0.130	0.125±0.154	0.105±0.129

Table 4.7: Rotation error for sensor motion (radians)

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Town 1	**0.002±0.003**	0.012±0.014	0.010±0.013	0.012±0.015	0.007±0.012
Town 2	**0.001±0.003**	0.012±0.015	0.013±0.017	0.015±0.022	0.011±0.019

Table 4.8: Translation error for movable objects (meters)

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Town 1	0.409±0.419	0.313±0.629	0.284±0.406	0.285±0.485	**0.259±0.475**
Town 2	0.289±0.280	0.254±0.316	0.196±0.189	0.220±0.276	**0.193±0.229**

Table 4.9: Rotation error for movable objects (radians)

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Town 1	**0.006±0.011**	0.028±0.061	0.043±0.093	0.040±0.109	0.029±0.084
Town 2	**0.005±0.009**	0.026±0.034	0.031±0.046	0.035±0.066	0.028±0.056

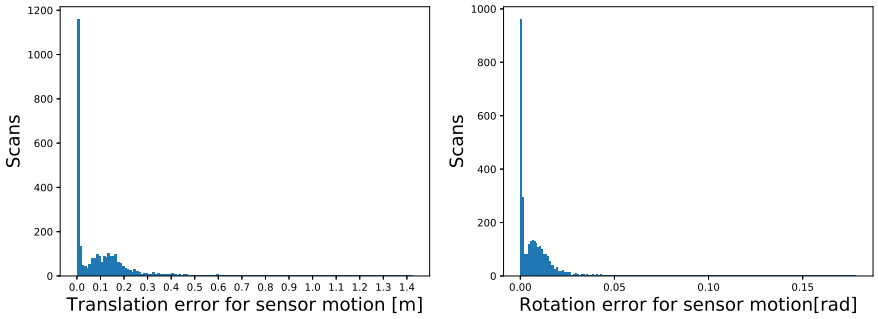
Table 4.10: Alignment accuracy for the simulated Dataset

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Town 1	0.925±0.075	**0.972±0.040**	0.961±0.057	0.970±0.035	0.969±0.040
Town 2	0.930±0.069	**0.968±0.042**	0.960±0.052	0.965±0.045	0.964±0.045

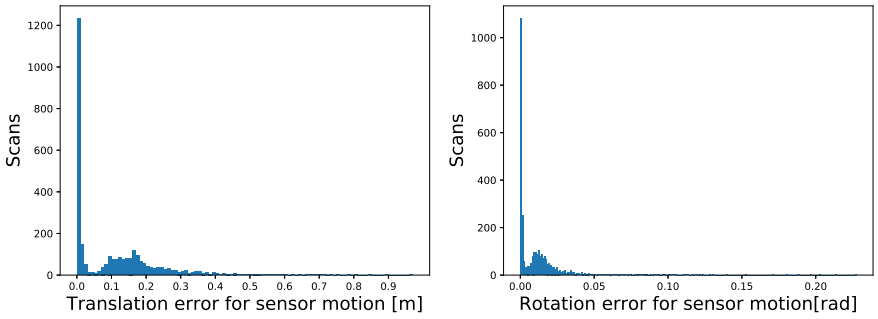
4.3.2.3 Non-rigid Objects

The foremost contribution of our method is that it can seamlessly estimate multiple motion models. We illustrated the advantage of this feature in the previous section, where we showed results for sequences with multiple dynamic objects. In that case, even though objects like vehicles or bicyclists were rigid, the whole scene was non-rigid due to dynamic objects. In this section, we focus on non-rigid objects rather than non-rigid scenes. We collected a dataset with four different kinds of human motion: bending forward, bending sidewards, moving arms upwards and moving arms backwards. Figure 4.18 shows the input scans and the alignment results for ICP and our method. Unlike ICP, our method is able to estimate different motions and align consecutive scans for all of the cases.

In Table 4.11, we report the alignment accuracy for all of the cases. Similar to before, we use the crispness score as a metric for the alignment accuracy and compare our method with ICP, and report results for the cases where false correspondences between the keypoints are not filtered and the weighting for the smoothnes and data term is identical. For all of the cases, the accuracy of our method is higher than ICP. The performance of two factor graph representation is similar and the reason for this is same as discussed in Section 4.3.2.1. Having same weight for data



(a) Distribution of error for sensor motion in Town 1 sequence.



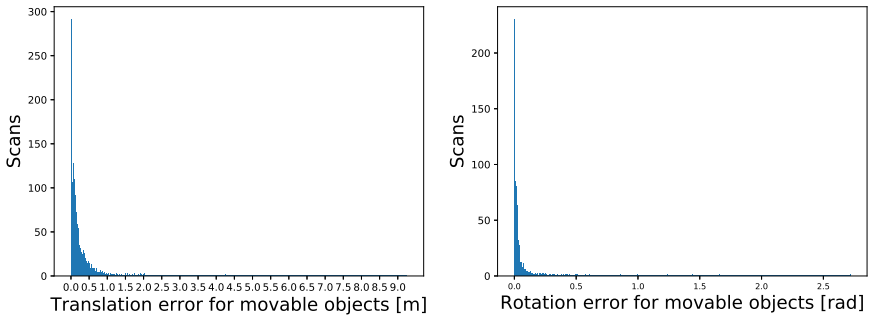
(b) Distribution of error for sensor motion in Town 2 sequence.

Figure 4.16

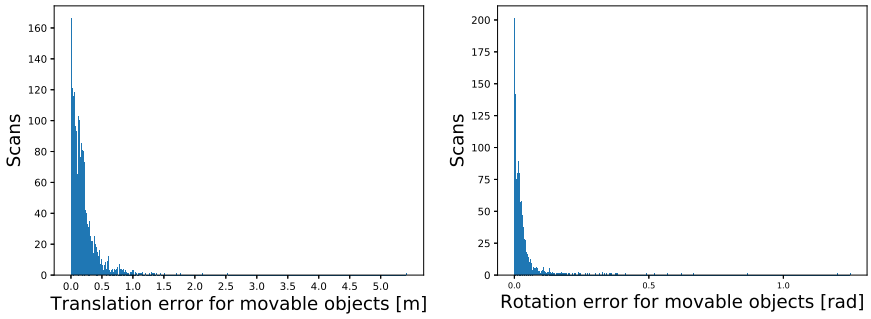
and smoothnes term results in a significant decrease in accuracy. In the case of non-rigid objects, the smoothnes term is of critical importance to estimate motion that is only locally rigid and therefore equally weighting data and smoothnes term deteriorates the performance. Filtering out correspondences also results in a slight decrease in accuracy for a few cases.

Table 4.11: Alignment accuracy for non-rigid objects

	ICP	fg1	w/o c.f	$\beta = 1$	Ours
Moving arm upwards	0.937±0.043	**0.983±0.006**	0.980±0.008	0.924±0.040	0.981±0.006
Moving arm backwards	0.933±0.045	0.984±0.008	0.986±0.004	0.920±0.048	**0.987±0.004**
Bending sidewards	0.946±0.022	**0.972±0.002**	0.969±0.005	0.881±0.067	0.970±0.005
Bending forward	0.857±0.067	0.930±0.029	0.937±0.025	0.802±0.011	**0.936±0.026**



(a) Distribution of error in motion for movable objects in Town 1 sequence.



(b) Distribution of error in motion for movable objects in Town 2 sequence.

Figure 4.17

4.3.2.4 Learned Feature Descriptors

Table 4.12: Translation error for handcrafted and learned descriptors (meters)

	SHOT	Descriptor w/o metric	Descriptor w/ metric
Seq. 3	0.121±0.067	0.114±0.057	**0.113±0.051**
Seq. 5	0.137±0.062	**0.127±0.053**	0.129±0.048
Seq. 6	0.321±0.114	0.322±0.113	**0.301±0.105**

For the results reported so far, we used the SHOT descriptor for finding corresponding keypoints between consecutive scans. In Chapter 3, we proposed a method for learning local feature descriptors for the same task of matching keypoints between scans. In this section, we compare the motion error and the alignment accuracy after replacing the SHOT descriptor with the learned descriptors. Among the learned descriptors, we show results for the descriptors learned with and without metric from 2D image patches.

In Table 4.12, 4.13 and 4.14 we report the error in translation and rotation motion, and the alignment accuracy respectively. For all of the cases, using learned

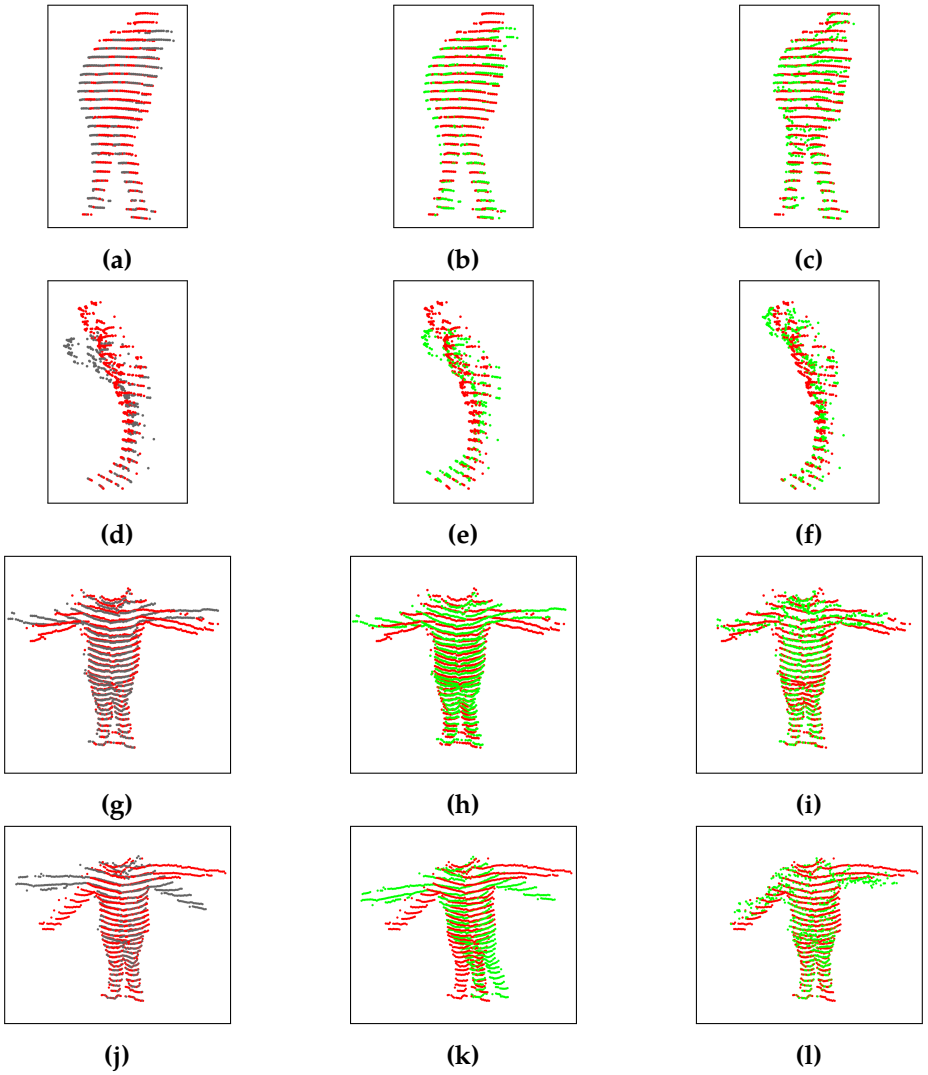


Figure 4.18: Alignment of non-rigid bodies for body bending upwards (a)-(c), body bending sidewards (d)-(f), arm moving backwards (g)-(i) and arm moving upwards (j)-(l).The first column shows the input scans. The points in scan P_t are shown in red, scan P_{t-1} is shown in gray, and the scans transformed by estimated motion model are shown in green. The results for ICP are illustrated in the second column and the results for our method are shown in the last column. In all cases, different parts of the body have different motion, and our method seamlessly estimates various locally rigid motion models to align consecutive scans, whereas ICP only estimates a single motion model and fails to align the scans.

Table 4.13: Rotation error for handcrafted and learned descriptors (radians)

	SHOT	Descriptor w/o metric	Descriptor w/ metric
Seq. 3	0.021±0.015	0.019±0.013	**0.017±0.011**
Seq. 5	0.024±0.017	0.023±0.015	**0.021±0.014**
Seq. 6	0.041±0.031	0.04±0.02	**0.035±0.021**

Table 4.14: Alignment accuracy for handcrafted and learned descriptors

	SHOT	Descriptor w/o metric	Descriptor w/ metric
Seq. 3	0.931±0.022	0.938±0.017	**0.939±0.016**
Seq. 4	0.896±0.049	0.902±0.046	**0.905±0.042**
Seq. 5	0.935±0.018	0.940±0.015	**0.941±0.014**
Seq. 6	0.887±0.028	0.901±0.020	**0.905±0.021**
Seq. 12	0.847±0.069	0.847±0.076	**0.854±0.070**

descriptors helps in reducing the error in the estimated motion. This reiterates the results reported in Table 3.3 from Chapter 3, where the error in alignment was lower for the learned descriptors. Comparing the alignment accuracy, for all of the cases the learned descriptor outperforms the SHOT descriptor. Among learned descriptors perform comparably, the descriptor learned with metric performs better for most cases.

4.4 Semantic Classification using Pointwise motion

In Section 4.3, we proposed a novel method for estimating pointwise motion and through extensive evaluation; we showed that our method is capable of estimating multiple rigid motion models. Having such motion cues can potentially aid semantic scene understanding, more specifically, it can help to infer the motion state of 3D LiDAR points i.e. whether a point in a scan is static or dynamic. In Section 4.2.2, we targeted the same problem and proposed a method for detection and tracking of dynamic objects solely through motion cues. This method assumed that a scan can be decomposed into a set of rigid objects, which can be iteratively detected using the estimated motion models. The largest rigid object detected using this method is considered static and the objects detected subsequently are considered dynamic.

Now we focus on estimating the pointwise motion state for each point rather than detecting dynamic objects. Building upon our method of estimating pointwise motion, we propose a Hidden Markov Model (HMM) based approach for estimating motion state for points in a 3D LiDAR scan. For a point $\mathbf{p}^t$ in a scan at time t, we define a random binary state variable $X^t = \{\mathtt{s}, \mathtt{d}\}$, where $\mathtt{s}$ and $\mathtt{d}$ are the two possible values for X representing the two motion states, **s**tatic and **d**ynamic

respectively. The objective is to estimate the belief of the current state.

$$Bel(X^t) = P(X^t \mid \boldsymbol{\tau}^{1:t}), \tag{4.50}$$

where, the current belief $Bel(X^t)$ depends on the the motion measurements $\boldsymbol{\tau}^{1:t}$. In Eq. (4.51), we show the simplification of Eq. (4.50) using the Bayes rule and the Markov assumption. Since X is a discrete random variable, Eq. (4.51) can rewritten as Eq. (4.52).

$$Bel(X^t) = \eta P(\boldsymbol{\tau}^t \mid X^t) \int P(X^t \mid X^{t-1}) Bel(X^{t-1}) dX^{t-1} \tag{4.51}$$

$$Bel(X^t) = \eta P(\boldsymbol{\tau}^t \mid X^t) \sum_{X^{t-1}=\{\mathsf{s},\mathsf{d}\}} P(X^t \mid X^{t-1}) Bel(X^{t-1}) \tag{4.52}$$

The likelihood of the measurement $\boldsymbol{\tau}^t$ given the state is defined in Eq. (4.53), where $\hat{\boldsymbol{\tau}}^t$ is the expected measurement, which in this case is the odometry. A large likelihood value indicates a point being static, since the estimated motion is similar to the odometry and a small likelihood value indicates the opposite. In order to propagate the belief between consecutive scans, we use the estimated motion for finding dense pointwise data association. Points in a scan at time t are aligned with points in scan $t+1$ using the estimated motion and then data association is performed by finding the nearest neighbor using the Euclidean distance.

$$P(\boldsymbol{\tau}^t \mid X^t) = \mathcal{N}(\boldsymbol{\tau}^t; \hat{\boldsymbol{\tau}}^t, \Sigma) \tag{4.53}$$

A key advantage of our method is that the inference of the motion state is not solely dependent on the current measurement but also on the previous belief. This makes our method robust towards spurious sporadic incorrect motion measurements and makes the inference temporally consistent.

4.4.1 Results

We test our approach on the dataset from Moosmann and Stiller [74], that we used before to evaluate our method on detection and tracking of dynamic objects, and now, in order to compare the performance of both methods. To validate our claim that our method enables temporally consistent inference, we report results for the case where inference is solely based on the current measurement, which in this case is the likelihood term described in Eq. (4.53). For evaluation, we report the intersection over union (IoU), and pointwise precision and recall.

The semantic classification method proposed in the previous section relies on the pointwise motion for estimating the motion state of points in a scan. In Section 4.3.2.4, we showed how learned descriptors from Chapter 3 enable estimation of accurate motion in comparison to a handcrafted descriptor. For estimating the motion state, we thus use the pointwise motion estimated using the descriptor learned

without metric, since matching of this descriptor requires low computational time in comparison the descriptor learned with a metric. We also report results for the pointwise motion estimated using the SHOT descriptor.

In Figure 4.19, we illustrate the estimated motion state for points in a scan from sequence B. In Table 4.15, for sequence A, we report results for our method of detection and tracking (D & T) of dynamic objects, the method where the inference is based on the likelihood of the measurement, and our method based on HMM. For the latter two, we report results for both SHOT and our learned descriptor. In Table 4.15, we report the same results for Sequence B.

Comparing the IoU for the static class for sequence A, the performance for all the methods are comparable. Results are better for HMM in comparison to the method where inference is solely based on the likelihood term. These results justify our initial claim that our method based on HMM makes the inference temporally consistent. Comparing descriptors, the performance is marginally better for SHOT descriptor. This is mainly due to the low recall in the case of learned descriptor. For sequence B, the HMM based method performs better than the other methods. Similar to sequence A, between descriptors, the performance is slightly better for the SHOT descriptor. Figure 4.20 illustrates a visual comparison for a sequence of three scans, between the method where inference is only based on the likelihood values and method based on HMM. By combining the prior belief with the correct measurement, our method is able to overcome isolated erroneous motion measurements.

For the dynamic class, the detection and tracking method outperforms our method based on HMM. Even though our method has either better or comparable recall, it is plagued by false positives, which is reflected in the low IoU and precision values. In the detection and tracking method, we used the concept of trackelets to combat the false positive detections, but a similar approach for pointwise tracking is not feasible. The majority of the false positives are for points that are far from the sensor and in these cases the estimated feature descriptors are not reliable enough for correct data association. This effect is further exaggerated by the fact that the number of dynamic points are significantly lower compared to the static points and, therefore, a small number of misclassified static points could severely impact the classification of dynamic points. In the case of Sequence A, only 10% of the static points are misclassified as dynamic, which results in a low precision value of 40.6% for dynamic points. A similar pattern is observed for sequence B as well.

Comparing our HMM based method with inference solely relying on likelihood, a consistent improvement is achieved for both sequences for the dynamic class. In the case of the likelihood method, the IoU is marginally better for the learned descriptor across sequences, with both higher precision and recall values and comparable for HMM based method. In the case of sequence B for the learned descriptor, recall decreases drastically for the HMM based method. This is mainly due to the fact

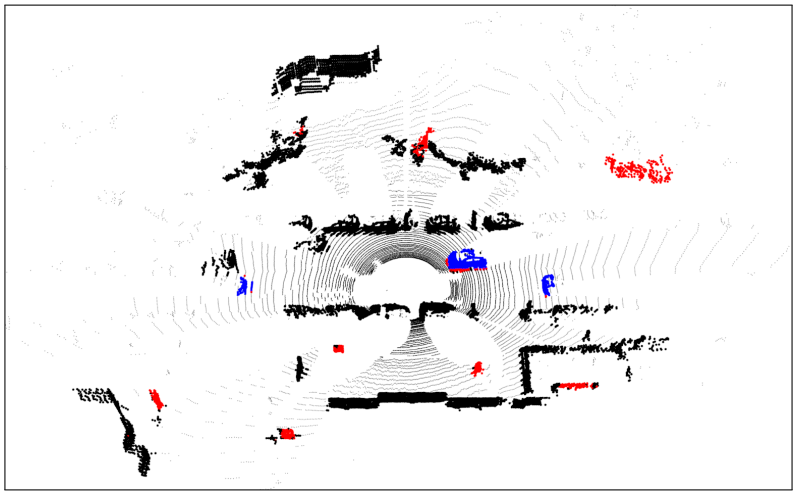


Figure 4.19: An illustration of the estimated motion state for points in a 3D LiDAR scan. Points classified as static are shown in black and dynamic points in blue color. Static points that are misclassified as dynamic are shown in red.

Table 4.15: Results for sequence A

Method	IoU		Precision		Recall	
	static	dynamic	static	dynamic	static	dynamic
D & T	88.4	**57.3**	99.3	**71.2**	88.9	75.2
Likelihood (SHOT desc.)	86.1	30.1	97.3	34.2	88.1	71.9
Likelihood (learned desc.)	84.4	32.4	98.0	35.0	85.9	**81.2**
HMM (SHOT desc.)	**88.6**	37.4	98.2	41.0	**90.0**	**81.2**
HMM (learned desc.)	87.4	36.8	97.9	40.6	89.1	80.0

that the sequence starts at an intersection with a large number of static vehicles and which switch their state to dynamic later in the sequence. The transition of states is faster in the case of likelihood method in comparison to the HMM based method.

Using the proposed HMM based method, we are able to classify dynamic points correctly, but also incorrectly classify small percentage of static points as dynamic. In Section 5.4, we revisit the problem of semantic classification of a LiDAR scan, where we combine these motion measurements with the learned semantic cues and address the issue of high false positive rate.

4.5 Related Work

The problem of detecting and tracking multiple moving objects has been studied actively for decades [13], [14]. Proposed methods to solve this problem can be broadly subdivided into model-free [61], [82], [124] and model-based [81], [106],

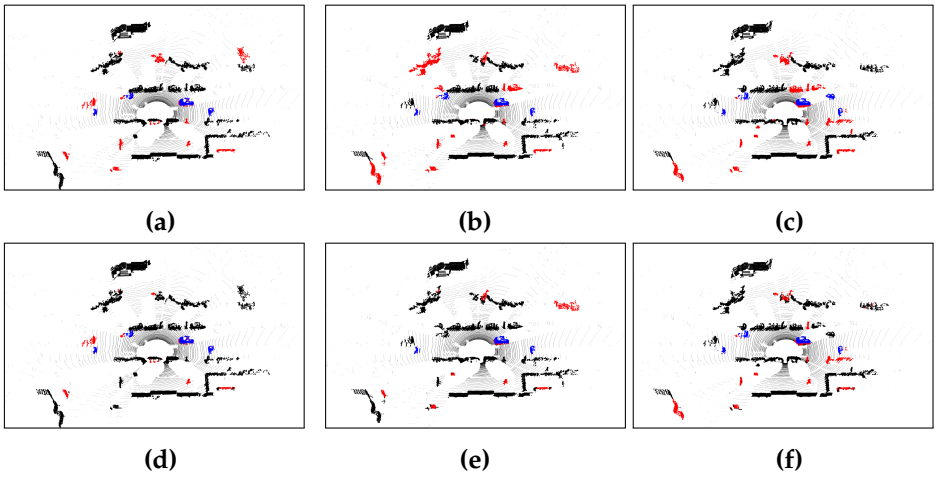


Figure 4.20: A visual comparison of the estimated motion state, solely based on likelihood (top) and our method based on HMM (bottom) for a sequence of three scans. The points in red are false positives for the dynamic class i.e. static points that are incorrectly classified as dynamic. Part of a wall (top left) is correctly classified as static for the first scan (a), misclassified as dynamic in the next scan (b) and then correctly classified again (c). For the HMM based method, the same part of the wall is correctly classified for of all the three scans. Using the belief from the previous scan, our HMM based method is able to overcome erroneous motion measurement, making temporally consistent predictions.

Table 4.16: Results for sequence B

Method	IoU		Precision		Recall	
	static	dynamic	static	dynamic	static	dynamic
D & T	82.5	**53.8**	**98.6**	**75.6**	83.4	65.1
Likelihood (SHOT desc.)	85.6	31.0	96.2	37.0	88.6	65.8
Likelihood (learned desc.)	84.1	32.8	96.6	37.4	86.6	**72.7**
HMM (SHOT desc.)	**89.2**	36.3	96.5	46.2	**92.5**	63.0
HMM (learned desc.)	88.4	36.5	95.8	46.3	91.9	63.4

[98] approaches.

In model-based approaches, objects are detected on the basis of known model information. These approaches are preferred when the object to be detected is known and therefore can be modeled a priori. In [81], an approach for detection and tracking of cars is presented. For people, Spinello et al. [106] proposed a learning based approach. They subdivide a human structure into multiple layers based on height and then learn a classifier for each layer. Similarly, Shackleton et al. [98] outline another method for detecting and tracking people. The main disadvantage of model-based approaches is that they do not generalize to objects of different categories, whereas a model-free approach allows detection of generic objects. The advantage of model-based approach is that this semantic knowledge can be seen as prior knowledge about the motion state of objects in a scan. Points belonging to a vehicle have a higher chance of being dynamic in comparison to the points on a building. In the next chapter, we revisit this idea of using the semantic knowledge for estimating the motion state of points in a LiDAR scan.

Model-free methods are mainly based on motion cues and enable detection and tracking of objects of arbitrary shape and size. Since these approaches require motion information, they are unable to detect objects that can potentially move but are static in the current observation. Model-free approaches are generally based on building a static map of the scene and using this map information for detecting dynamic objects. Pomerleau et al. [82] make a visibility assumption that the scene behind the object is observed, if an object moves. To leverage over this information, they compare an incoming scan with a global map and detect dynamic objects. Since they only use depth as cue for change detection, their method might fail if the motion between scans is small.

Kaestner et al.[61] propose a generative Bayesian approach for detecting dynamic objects. For tracking, they use an approach based on the Kalman filter. They show results for a static sensor but as mentioned by Pomerleau et al. [82] there is no straightforward extension of their approach to a moving sensor. Recently, Wang et al. [124] proposed a model-free approach for detection and tracking in 2D LiDAR data. Using a joint state representation, they estimate the state of the sensor, a local static map, and the state of the dynamic object. Every incoming scan is associated with a local static map and with dynamic objects. For tracking, they use a constant velocity motion model. While we have similar objectives, a comparison to our method is infeasible, since our approach works on 3D instead of 2D LiDAR data.

Azim and Aycard[10] represent the environment using an octree-based occupancy grid and determine inconsistencies between the map and incoming scans to detect dynamic objects. In contrast, we do not build a map, but similar to [124] we only store local static information. For tracking, they use Global Nearest Neighbor for associating tracks between consecutive frames. Tipaldi and Ramos [111] outline an approach for detecting and estimating motion using CRF for 2D LiDAR data. Van

De Ven et al. [117] extended their approach by integrating the CRF based method with scan matching using a graphical model.

Moosmann and Stiller [74] use a segmentation method based on local convexity for detecting object hypotheses. They combine ICP and a Kalman filter for tracking and a classification method for managing tracks. Unlike them, we do not use a shape prior for detection but rely only on motion information. We compare our approach with their method and show superior performance.

The problem of estimating motion flow has been studied intensively in the past but primarily in the computer vision community. The different developed methods can be distinguished according to the dimension of the motion field. *Optical flow* [34] describes 2D translation motion in image plane, *sceneflow* [50, 58, 120] describes 3D translation motion, and the *rigid scene flow* [79, 86] describes the rigid motion.

Fortun et al. [34] provides extensive literature review for optical flow. Building on optical flow, Vedula et al. [120] introduced the term scene flow. They included first order approximations of the depth map to estimate 3D flow. Herbst et al. [50] extended the approach presented by Brox et al. [18] and include a depth constraint to estimate scene flow. Jaimez et al. [58] introduced a real time, primal-dual algorithm based method to estimate scene flow for RGB-D data. However, these methods make assumptions that for reasons discussed in Section 4.3 are not valid for our case.

For estimating dense semi-rigid flow for RGB-D data, Quiroga et al. [86] solved an energy minimization problem, using TV regularization to estimate piecewise smooth motion. Vogel et al. [122] proposed a method to estimate the 3D motion and structure using RGB-D data. The main contribution of their work is using local rigid regularization instead of variational regularization. Newcombe et al. [79] proposed a method for dense SLAM using RGB-D scans, where they reconstruct deforming surfaces and simultaneously estimate dense volumetric 6D motion field. These methods show commendable results but they use color and depth images, while our approach relies on sparse depth data, therefore a direct comparison is infeasible.

The methods discussed so far show results mainly for indoor scenes. For outdoor environments, Menze and Geiger [73] proposed a method for object scene flow using images. They oversegment the scene into super pixels and using CRF jointly estimate rigid motion and an association between super pixels and objects. Even though their assumption about rigid structure of the outdoor environment is not invalid but our assumption of local rigidity enables our method to estimate motion of a non-rigid object (in case of humans), making our method more robust towards changes in the environment.

Our assumption of local rigidity allows for deformation of surfaces. Addressing this problem, Haehnel et al. [41] proposed an approach extending ICP for registering deformable surfaces for sparse LiDAR data. Similar to us they perform

pointwise estimation, allowing for smooth deformation of the surface. The main difference between our method and their method is that we estimate feature based correspondences, while they use nearest neighbors for data association. The assumption of nearest neighbors breaks if two surfaces are far from each other, for instance, when a dynamic object moves in the direction opposite to the direction of sensor motion. Our feature based method is immune to these cases and can estimate large motion. For registering deformable 3D surfaces, Dragomir Anguelov et al. [27] extended the approach by Haehnel et al. [41] by introducing the concept of correlated correspondences, where they estimate pointwise deformation and correspondence. Our approach adjusts to the cases of non-rigid objects but in this paper we are not concentrating on registering deformable dense 3D surfaces but instead estimating pointwise motion for sparse LiDAR data, therefore a comparison with the work of Dragomir Anguelov et al. and the approach of Cosmo et al. [25] is beyond the scope of this thesis.

More recently, different methods for estimating motion flow for images [57], [31] and pointclouds [15] using deep learning have been proposed. Unlike these methods, which employ end-to-end learning techniques, we incorporated a learned descriptor to leverage the superior performance for deep learning methods. Even though in this thesis we did not focus on end-to-end learning techniques for estimating the motion, such an approach can be seen as a favorable possible future extension of our work.

4.6 Conclusions

In this chapter, we focused on the problem of scene understanding solely using motion cues. We tackle this problem at object level and also at finer point level. Firstly, we propose a novel approach to detect and track dynamic objects. We detect motions between consecutive scans by sequentially using RANSAC and propose a Bayesian approach to segment and track multiple objects. Our method is model-free, i.e. it does not require any prior information about the objects. We evaluate our approach on two publicly available data sequences and compare it with an existing method. For both sequences, our approach achieves a better F_1 score. Furthermore, we track the speed of the sensor and of another object with a higher accuracy. For understanding the finer dynamic details about the environment, we propose a novel method for estimating the rigid scene flow for 3D LiDAR data. We introduce the concept of geometric constancy and use spatial smoothing to estimate dense rigid motion flow. Since our method solely relies on the data association between keypoints for estimating motion, we also introduce a method for estimating robust data association. We test our approach on one simulated and two real datasets and use ICP as a baseline method. Through exhaustive evaluation, we show how our method is capable of estimating multiple rigid motion models, which is necessary

in a scenario where either the scene or the object is non-rigid. Even though our method overcomes the limitation of a method like ICP, which estimates a single rigid motion model, the accuracy of the estimated motion for our method decreases in the cases where the data is sparse. We also present an ablation study where we analyze the contribution of different features of our approach. Building upon the pointwise motion, we present an HMM based approach for estimating the motion state for points in a scan. Such an approach makes the inference temporally consistent and also robust towards sporadic noisy measurements. We justify this claim through both qualitative and quantitative evaluation. We also compare the HMM based approach with our approach of detection and tracking of dynamic objects. The latter approach is more precise when compared to the HMM based approach and we address this issue in the next chapter.

Chapter 5

Scene Understanding using Motion and Semantic Cues

Robots are expected to operate autonomously for long periods of time in dynamic and changing environments. Understanding the underlying dynamic and semantic characteristics causing the changes, is a key enabler for achieving this goal. In this chapter, we focus on garnering the semantic knowledge about the environment and combining this knowledge with motion cues to understand the 3D environment. We propose a DCNN to segment points in a LiDAR scan into the classes *car*, *pedestrian* or *bicyclist*. This architecture is based on dense blocks and efficiently utilizes depth separable convolutions to limit the number of parameters while still maintaining state-of-the-art performance. To make the predictions from the DCNN temporally consistent, we propose a Bayes filter based method. This method uses the predictions from the neural network to recursively estimate the current semantic state of a point in a scan. This recursive estimation uses the knowledge gained from previous scans, thereby making the predictions temporally consistent and robust towards isolated erroneous predictions. Combining the motion cues from the previous chapters and the learned semantic cues, we propose a novel HMM based method for pointwise semantic classification of a 3D LiDAR scan into three classes: *non-movable*, *movable* or *dynamic*. We concentrate on understanding these specific semantics because they characterize important information required for an autonomous system. We compare our proposed DCNN with other architectures and report a remarkable improvement over state-of-the-art methods. For the Bayes filter and HMM based methods, we show results for the entire KITTI tracking benchmark. Using the Bayes filter method, we consistently improve the neural network predictions across multiple sequences in the benchmark. Through our evaluation of the HMM based method, we show how combining the semantic and motion cues can significantly improve the classification of dynamic points.

5.1 Introduction

One of the vital goals in mobile robotics is to develop a system that is aware of the dynamics of the environment. As the environment changes over time, the system should be able to understand this evolution and then act accordingly to deal with these changes. So far, we have primarily focused on improving our understanding of the environment from the perspective of motion. In Chapter 3, we proposed a method for learning local feature descriptors with the objective of estimating robust correspondences between keypoints. In Chapter 4, we proposed a method for detecting and tracking of dynamic objects and a method for estimating pointwise motion. Both of these methods heavily relied on data associations between keypoints for estimating the motion models. For 3D scene understanding using motion cues, in Section 4.4, we proposed a Hidden Markov Model based approach for inferring the motion state of each point using the estimated pointwise motion.

In this chapter, we shift our focus on inferring semantic cues and leverage both semantic and motion cues for the understanding of the environment. Different methods targeting the problem of semantic understanding, have primarily focused on either object detection [67, 85, 89, 90, 132] or semantic segmentation [11, 20, 59, 68, 92, 127, 128] and some methods have focused on both [47]. In the case of object detection, the objective is to localize the location of an object, estimate the bounding box coordinates encompassing the targeted object and also infer the semantic class to which the object belongs. Having such information allows the estimating of the number of instances of objects belonging to a semantic category and can potentially aid the tracking of objects [96, 99], obstacle avoidance and safe navigation [29]. The objective of semantic segmentation is to have a dense prediction i.e. inferring the semantic category either pixel or point wise. Having such a dense prediction can help us understand the fine details about the environment. Segmentation can also be used as a preprocessing step for object detection [37], and for robust visual localization [78, 87] among many other applications.

In this chapter, we propose a novel **D**eep **C**onvolutional **N**eural **N**etwork (DCNN) architecture for classifying points in a 3D LiDAR scan into the following semantic categories: *car*, *pedestrian* or *bicyclist*. Recently, deep neural network based methods have lead to breakthroughs in several vision tasks, such as classification [45, 54, 104], detection [67, 85, 89, 90, 132] and segmentation [11, 20, 59, 68, 92, 114, 115, 116, 127, 128]. The majority of these methods are based on camera images [11, 20, 45, 54, 67, 68, 89, 90, 92, 104] and few methods have focused on using 3D LiDAR scans [85, 127, 128, 132]. In Chapter 3, we proposed a dense blocks based DCNN for learning local feature descriptors and now we use it again for the task of semantic segmentation. Our proposed architecture is inspired by a fully convolutional dense blocks based DCNN proposed by Jégou et al. [59], but with some vital modifications. To reduce the number of parameters, we replace the standard convolution layers

with depth separable convolution layers for dense blocks in the decoder. This allows us to reduce the number of parameters by a significant amount while still having competitive performance. To down-sample the feature maps, they proposed a transition down block comprising of a composite function implementing different operations. We replace this block with a single max-pooling operation and show that instead of a composite function, this single operation is sufficient.

The input to our proposed architecture is a 2D image, which is generated after projecting the 3D LiDAR scan onto a spherical plane. This image has multiple channels that encode separate modalities. The resolution of the image after the projection is skewed i.e. the height of the image is smaller than the width, since the vertical field-of-view of the scanner is lower than the horizontal field-of-view. This characteristic is critical because an important operation within a DCNN is down-sampling of feature maps and the rate of down-sampling is crucial for our case because of the small height of the image. Different methods proposed for semantic segmentation of a 3D LiDAR scan using 2D images have proposed to down-sample the feature maps only along the width dimension [127] or using dilated convolutions [125]. We propose to down-sample the feature maps $4\times$ along both spatial dimensions. This helps us to utilize the benefits of down-sampling the feature maps (Section 2.4) while limiting the loss in information.

Standard DCNN architectures treat each example independently and do not use any previous or prior information. The data is rarely not sequential , especially in the case of perception in robotics. To leverage over this sequential nature of information, we propose a Bayes filter approach for making our segmentation results temporally consistent. Similar to the objective of our Hidden Markov Model based approach (Section 4.4) for inferring the motion state, our current objective is also to make our inference robust towards sporadic erroneous DCNN predictions. To enable such temporally consistent inference, dense data association between points in consecutive scans is required and to this end we extend our method of estimating pointwise motion to pointwise tracking, as discussed in Section 4.4. Different deep learning methods like Recurrent Neural Network (RNN) [39] or Long Short Time Memory (LSTM) [51] exist for processing sequential data with the same objective of having temporally consistent prediction but in this thesis we do not explore such methods.

Combining the motion cues with the learned semantic cues, in this chapter we also propose a novel method for classifying points in a scan into the following three classes: *non-movable*, *movable* or *dynamic*. Using the semantic cues, we can distinguish non-movable structures, like buildings, from movable structures, like vehicles, pedestrians, among others. In contrast to the semantic cues, the motion cues can help in separating the dynamic parts of the scene from the static. Since both of these cues entail different and complimentary information, combining them allows a better understanding of the environment.

Combined knowledge of these classes highlights the underlying dynamics of the environment and enables the robot to tackle different challenges pertaining to navigation in a non-static or changing environment. The distinction between these classes lies mainly in the time window within which change in the environment is observed. For instance, pose or state of a dynamic object changes in consecutive scans and therefore has non-zero motion within this time window. A movable object, for e.g. a parked vehicle, has zero motion between consecutive scans but is expected to change its location in few hours or days. Among the three classes, non-movable parts of the environment are unexpected to remain unchanged for longest period of time.

Considering navigation, knowledge of these classes is critical because it can potentially allow uninterrupted navigation for longer periods of time. For example, points classified as dynamic or movable can either be filtered out during mapping or, as shown by Ruchti and Burgard [93], this information can also be incorporated within the mapping framework. Points classified as non-movable can be used for both efficient localization and mapping. In the case of localization, non-movable points have higher chances of being re-observed and therefore can be easily matched against a known or a mapped environment. Similarly, knowledge of this class can help in avoiding frequent mapping runs of a changing environment. In the case of robust mapping and localization, knowledge of non-movable points is essential, but for safe and robust path planning, knowing which objects in the scene are dynamic is critically important. Separating movable objects from dynamic is also necessary, as it can aid in understanding different semantics of the environment. For instance, knowledge of parked vehicles can be seen as a necessary precursor for detecting parking spots and similarly knowing which movable objects are dynamic can be used for distinguishing between the bike lane from a sidewalk, among other things.

To infer the desired semantic classes, we propose an HMM based approach. This approach is similar to the HMM based approach we proposed in Chapter 4 but instead of inferring only the motion state (static or dynamic), we now focus on inferring the motion and the semantic state together. We gain the semantic knowledge, using the proposed DCNN for semantic segmentation and the Bayes filter approach for making the prediction from the neural network temporally consistent. This knowledge is then incorporated within the HMM along with the motion cues. Using an HMM allows us to seamlessly integrate the motion and the semantic cues together. Furthermore, within this approach we incorporate the previous measurements, which allows us to properly utilize the sequential nature of the input data.

Other methods [30, 88, 121] for similar semantic classification have been mainly proposed for RGB images. For LiDAR data, separate methods exist for both object detection [22, 23, 28, 65], semantic segmentation [127, 128] and for distinguishing between static and dynamic objects in the scene [74, 82]. In the context of combining

the motion and the semantic cues, Vaquero et al. proposed a deep learning based method for estimating the vehicle motion only using 3D LiDAR scans. The main difference between our approach and their approach is that they are focusing on estimating the motion model, while we are focusing on inferring the motion state (dynamic vs static).

The contributions of this chapter include a DCNN for semantic segmentation of LiDAR scans into the classes: *car*, *pedestrian* or *bicyclist*. We compare our DCNN with state-of-the-art DCNNs [125, 127, 128] proposed to solve the same task. To justify different architecture design choices and gain further insight towards them, we also present an ablation study. This study includes a comparison with the architecture from Jégou et al. [59], analyzing the contribution of depth separable convolutions and understanding the effect of different down-sampling choices. Our next contribution is a Bayes filter approach for making the predictions of the neural network temporally consistent. We evaluate our approach on the entire KITTI tracking benchmark and use our proposed neural network as a baseline method.

The last contribution is a method for semantic classification of a LiDAR scan for learning the distinction between *non-movable*, *movable* and *dynamic* parts of the scene. To highlight the importance of integrating the semantic knowledge for the inference of the motion state, we report a comparison with our approach proposed in Section 4.4. The KITTI tracking dataset only provides ground-truth annotation for different objects and the associated tracking ID, but not the ground-truth motion state, i.e. whether an object is static or dynamic. To evaluate the inferred motion state, we extended the existing ground-truth annotation by adding the information regarding the motion state. The details regarding the method for estimating the ground-truth annotation of the motion state are discussed in Section 5.4.2.

5.2 Semantic Segmentation of a 3D LiDAR Scan

In Figure 5.1, we illustrate the complete framework for semantic segmentation of a LiDAR scan. The first step is to project the scan onto different 2D images with each such image encoding a specific modality. These images are then stacked together and are passed through our proposed DCNN for semantic segmentation. The segmentation mask predicted by the DCNN is then projected back to the LiDAR scan to infer pointwise semantic labels. The different steps within our framework are described in detail in the following sections.

5.2.1 Generating training data

We use the data from the KITTI benchmark for our task of semantic segmentation of 3D LiDAR scans. For LiDAR scans, the benchmark does not include this task, but only object detection and therefore the dataset only provides the ground-truth 3D bounding box annotation for the objects belonging to different semantic categories.

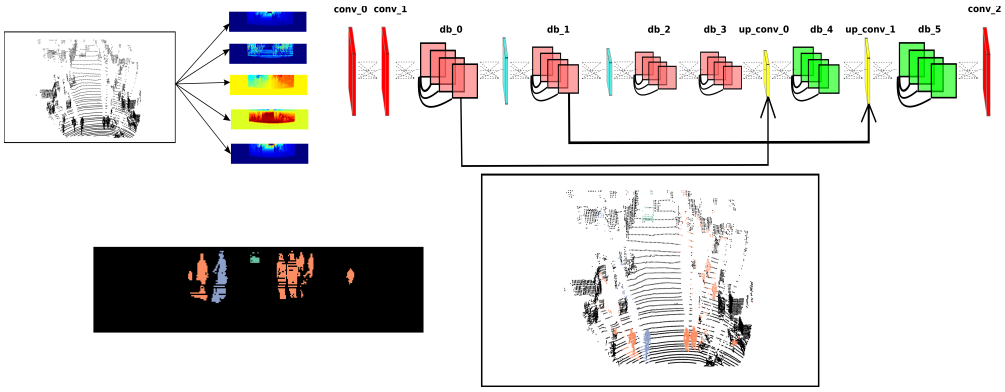


Figure 5.1: Our proposed semantic segmentation framework. In the first step, we project a LiDAR scan onto multiple 2D images and encode a specific modality in each image. These images are then stacked together, fed into the proposed CNN architecture, and the output is the predicted segmentation mask (bottom left). The segmentation information is then projected back to the scan to infer the semantic labels for each point in the scan.

Unlike the case for 2D images, the 3D bounding boxes belonging to different objects do not overlap and therefore all the points within a bounding box have the same label. Using this cue we obtain the pointwise annotation required for the task of semantic segmentation from the available object level ground-truth annotation.

One of the primary challenges pertaining to different learning based methods using 3D data is the decision regarding the data representation [108]. Unlike 2D images, which are organized in a grid structure, 3D data is often unorganized and sparse, thereby making the problem of data representation non-trivial. Different commonly used representations are 3D points [83, 84], voxel grid representation [72, 91, 132], using LiDAR measurements (range and angles) for generating 2D images [125, 127, 128] and rendering a 3D scene from multiple viewpoints and projecting it onto 2D planes [60, 107]. Depending on the task, different representations are favored. Voxel grid representation or rendering multiple view points are favored in the case of object detection or classification [60, 107] [91, 132]. 3D points have been successfully used for object classification and detection, and semantic segmentation [83, 84, 85] and 2D images have been used for semantic segmentation [125, 127, 128] and object detection as well [23, 65]. Methods learning from a group of 3D points have primarily shown results for small indoor environments where the environment is perceived using a RGB-D sensor. In contrast to that, we are focusing on using LiDAR scanners for segmenting urban outdoor environments. These environments are generally larger compared to a room in an office or in a house. Also, the data from commonly used LiDAR scanners is sparser, especially for far away points (points farther than$\sim$30m) in comparison to the data from RGB-D sensors.

For segmenting LiDAR scans, we use 2D images generated from LiDAR measure-

ments. The two key advantages of using such representation is that large outdoor environments are represented in a dense and compact fashion and secondly, different learning methods proposed for 2D images are applicable for this representation as well. In comparison to other 3D representations, like voxel grid or 3D points, a shortcoming for using such 2D representations for learning a 3D environment is that the neighborhood information is lost, since far away points can be neighboring pixels in the image.

For different experiments, we use the dataset from Wu et al. [127] and our own dataset created from the KITTI tracking sequence. The first dataset is also created from the KITTI benchmark but from the raw sequences. Since the KITTI benchmark only provides ground-truth annotation for objects in front of the camera, only the 3D points in the front of the camera are projected on a sphere for generating the 2D image. To calculate the pixel location for points in a scan, for each 3D point (in front of the camera) $\mathbf{p}$, we first calculate the azimuth θ and polar ϕ angle as following:

$$\theta = \arctan(\frac{\sqrt{x^2 + y^2}}{z}), \tag{5.1}$$

$$\phi = \arctan(\frac{y}{x}), \tag{5.2}$$

and then calculate the pixel coordinate $\tilde{\theta}, \tilde{\phi}$ as following:

$$\tilde{\theta} = \frac{\theta}{\Delta\theta}, \tag{5.3}$$

$$\tilde{\phi} = \frac{\phi}{\Delta\phi}, \tag{5.4}$$

where $\Delta\theta$ and $\Delta\phi$ are the angle resolutions. The height of the image is identical to the number of channels of the LiDAR scanner. The LiDAR scanner used in the KITTI benchmark has 64 channels (Velodyne HDL-64E) and therefore the height of the image is 64. The width of the image is dictated by the angle resolution $\Delta\theta$. In the dataset by Wu et al. [127], the 2D images have $64\times512\times5$ resolution. The last dimension corresponds to the number of channels in the image. The first channel encodes the range value $r = \sqrt{x^2 + y^2 + z^2}$, the second channel encodes surface reflectance values and the last three channels encode the 3D coordinates (x, y and z) for a point $\mathbf{p}$. For our dataset generated from the KITTI tracking benchmark, we use the implementation from Moosmann and Stiller [74] for generating the 2D images of $64\times324\times5$ resolution. We use the first dataset to evaluate the performance of our proposed DCNN for the task of semantic segmentation. We use the scans from the KITTI tracking benchmark to evaluate the performance of our proposed Bayes filter framework. Figure 5.2 illustrates the individual channels of the 2D image generated from a 3D LiDAR scan. This is an illustration for a scan from the KITTI tracking sequence. Each individual channel encodes different information, which can be readily extracted from the raw pointcloud data.

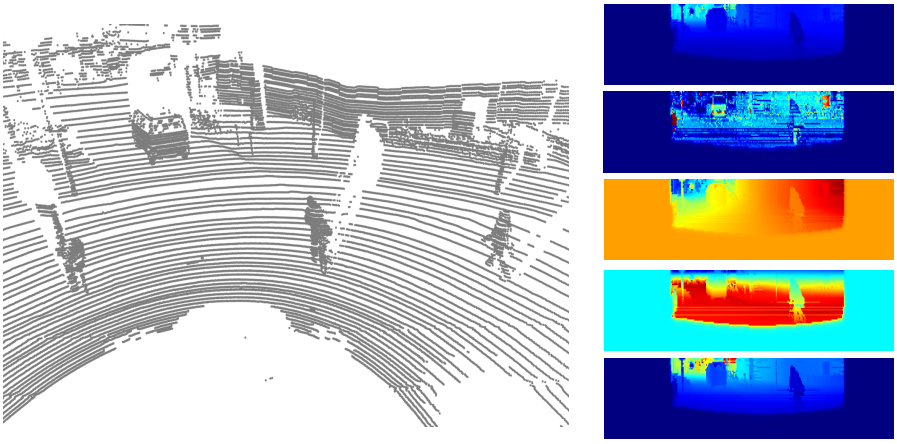


Figure 5.2: An illustration of the input data used of semantic segmentation. Figure on the left shows an example LiDAR scan from the KITTI tracking sequence and on the right we show the individual channels for the 2D images generated from this scan. The top image shows the encoding of the range values, the image below that shows the encoding of the surface reflectance values and the bottom three images shows the encoding of the 3D coordinate information (x, y and z) respectively. Each of these images have a single channel and are colorized for the sake of visualization.

5.2.2 Network Architecture

For the task of semantic segmentation, we propose a novel fully convolutional DCNN architecture called DBLiDARNet. Our architecture is based on dense blocks (Section 2.4.2) and is shown in Figure 5.3. Similar to other DCNN architectures proposed for the task of semantic segmentation [59, 68, 92], our network is also comprised of an encoder for learning the features required for the task while down-sampling the feature map size, and a decoder to up-sample the feature maps so that the last hidden layer has the same spatial resolution as the input image. In the encoder, we have two convolution layers (conv_0 and conv_1), three dense blocks (db_0, db_1 and db_2) and two max-pooling layers to down-sample the feature maps $4\times$ in comparison to the spatial resolution of the input image. In the decoder, we use two up-convolution layers to up-sample the feature maps and use two more dense blocks with depth separable convolution layers (Section 2.4.1). To limit the number of learnable parameters in the decoder, similar to the architecture proposed by Jégou et al. [59], in our proposed architecture the input to the up-convolution layers is the feature maps learned by the dense block prior to the up-convolution layer instead of all of the features maps learned until that point. For instance, the input to the layer up_conv_0 is only the feature maps learned by the dense block db_2. To recapture the information lost during up-sampling, we use skip connections to concatenate feature maps from the encoder to the output of the up-convolution layers.

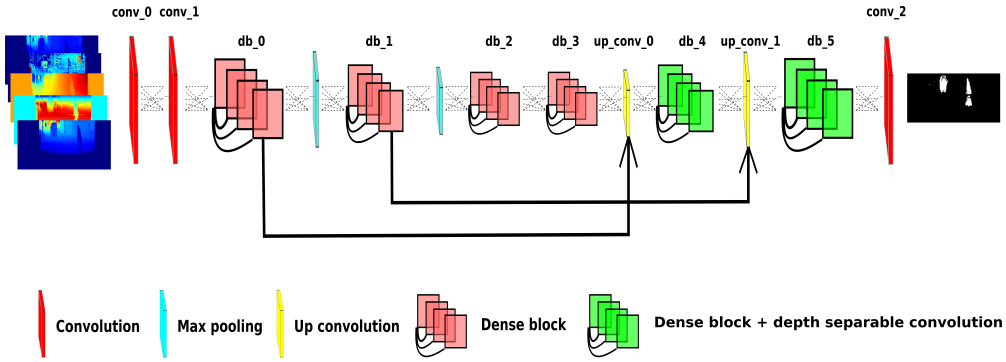


Figure 5.3: Our proposed architecture for semantic segmentation. In the encoder, we have two convolution layers (conv_0 and conv_1), two max-pooling layers and three dense blocks (db_0, db_1 and db_2). In the decoder, we use two up-convolution layers to up-sample the feature maps, two dense blocks (db_3, db_4) with depth separable convolution and one convolution layer (conv_2). We also use skip connection to concatenate feature maps from the encoder, in order to recapture the details lost due to up-sampling.

As already mentioned, the height of the input image is the same as the number of channels in the LiDAR scanner. In comparison to the data captured from standard RGB cameras, where the height and the width of the image are similar, in our the case the height (64) is $8\times$ smaller than the width (512) of the image. To limit the number of operations within different layers, make the architecture memory efficient and increase the receptive field, a common practice is to down-sample the feature maps, where the down-sampling rate varies from $16\times$ to $32\times$, depending on the task.

For the task of semantic segmentation where dense pixel wise prediction is required, a large down-sampling rate can lead to a decrease in performance [21]. Since the height of our input image is 64, reducing the feature map size 16 or $8\times$ will result in a feature map with a height dimension as 8 or 4, thereby resulting in a significant loss of information. Therefore, to arrest the information lost due to down-sampling operation, we only reduce the feature map size $4\times$. Other methods that are learning from images of same resolution have proposed down-sampling the feature maps $8\times$ ([127, 128]) but only along the width dimension, keeping the height dimension unchanged or down-sampling $4\times$ and using dilated convolutions in the last layers of the encoder [125]. In the ablation study, we report results for a network where we down-sample $8\times$ along both spatial dimensions to justify our decision of down-sampling $4\times$. We also train a network where we down-sample only $4\times$ but only along the width dimension, to compare with the down-sampling method proposed in [127, 128].

The complete details regarding the dimensions of each layer or block and different associated hyper-parameters is reported in Table 5.1. The kernel size of the filter for all the convolution and up-convolution layers except conv_2 is 3×3. In the

Table 5.1: Architecture

Layer name	Dimension (H×W×C)	Repetition	Depth separable
conv_0	$64 \times 512 \times 48$	-	No
conv_1	$64 \times 512 \times 48$	-	No
db_0	$64 \times 512 \times 144$	6	No
db_1	$32 \times 256 \times 272$	8	No
db_2	$16 \times 128 \times 432$	10	No
db_3	$16 \times 128 \times 240$	15	No
up_conv_0	$32 \times 256 \times 240$	-	No
db_4	$32 \times 256 \times 128$	8	Yes
up_conv_1	$64 \times 512 \times 128$	-	No
db_5	$64 \times 512 \times 96$	6	Yes
conv_2	$64 \times 512 \times 4$	-	No

last layer, we use a filter of size 1×1 to reduce the number of feature maps to the number of classes. The *stride* for each convolution layer is set to 1 and the *stride* for up-convolution layer is set to 2. For all dense blocks, the *growth rate* parameter (Section 2.4.2) is set to 16. The number of features learned within a dense block is *growth rate* times the *repetition*, where *repetition* is the number of times the composite function within a block is repeated. As mentioned before, the input to an up-convolution layer is only the number of feature maps learned in the previous dense block and therefore the output of the db_3 only contains the feature maps learned within the block (16×15), in contrast to the output of db_2, which consists of feature maps learned within the block (16×10) concatenated with the number of input feature maps (272). We also use a skip connection as shown in Figure 5.3. The input to the dense blocks in the decoder (db_4 and db_5) is concatenation of the feature maps learned by the previous up-convolution layer and the output of the dense block (in the encoder) with the same spatial resolution. In this case, the input to db_4 is the output of up_conv_0 concatenated with the output of db_1.

5.2.3 Loss Function

We formulate our problem of semantic segmentation as a multi-class segmentation problem. We define a set of training images $\Gamma = (\mathbf{X}_n, \mathbf{Y}_n), n = 1, \ldots, N$, where $\mathbf{X_n} = \{x_k, k = 1, \ldots, |\mathbf{X_n}|\}$ is a set of pixels in an example input image and $\mathbf{Y}_n = \{y_k = \{0, 1, 2, 3\}, k = 1, \ldots, |\mathbf{Y_n}|\}$ is the corresponding ground-truth segmentation mask image, where different values of y_k correspond to *background, car, pedestrian* and *bicyclist* classes respectively. The function learned by our proposed neural network is $\mathbf{f}(\mathbf{X}_n, \theta) \in \mathbb{R}^{H \times W \times C} \rightarrow \mathbb{R}^{H \times W \times C_k}$, where C_k corresponds to the number of classes and θ are the parameters learned by the network. The network learns these parameters by minimizing the cross-entropy (softmax) loss (Section 2.4.5.1)

in Eq. (5.5),

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{c \in \{0,1,2,3\}} y_c \log \hat{y}_c \tag{5.5}$$

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N \times |\mathbf{Y_n}|} \sum_{n=1}^{N} \sum_{l=1}^{|\mathbf{Y_n}|} \mathcal{L}\left(\mathbf{y}_n^l, \hat{\mathbf{y}}_n^l\right), \tag{5.6}$$

where, $\mathbf{y}_k^l \in \mathbb{R}^4$ is one-hot encoding of a pixel l in the image $\mathbf{Y}_n$ and $\hat{\mathbf{y}}_n^l \in \mathbb{R}^4$ is the predicted softmax probabilities (Eq. (2.10)) for the same pixel l.

5.2.4 Training

Our complete network architecture is implemented in TensorFlow [8]. We use the dataset provided by Wu et al. [127], consisting of 8057 images for training and 2791 images for testing. As mentioned before, the dimension of the image is $64 \times 512 \times 5$, where 5 channels include depth, intensity and 3D coordinate information. We train the network with a batch size of 2 and use the Adam optimizer [62] with a learning rate of $1e^{-4}$ and weight decay of $5e^{-4}$. Among the three classes, the point measurements from cars is significantly more than the measurements from either pedestrians or bicyclists, mainly because of the inherent difference in the size of the geometrical structure. This leads to the problem of class imbalancing, where some classes in the training data overwhelm the classes that are under represented. To tackle this, we use a weight balancing technique and assign larger weights to points belonging to the class *pedestrian* and *bicyclist* in comparison to points belonging to the class *cars* and *background*.

5.2.5 Results

To evaluate our proposed DCNN, we use the test set from the dataset provided by Wu et al. [127]. We report class wise IoU and compare our results with two DCNN proposed by Wu et al. ([127],[128]) and the network architecture proposed by Wang et al. [125]. The first architecture proposed by Wu et al. [127] is based on the SqueezeNet [56] architecture. They use *fire* modules, which first involve squeezing the feature maps using 1×1 filters and then expanding these squeezed feature maps in parallel using filters of size 1×1 and 3×3 and concatenating their outputs at the end. Using three max-pool layers they down-sample the feature maps only along the width dimension and to up-sample the feature maps they again use *fire* modules in the decoder. The last layer of their neural network architecture is a recurrent CRF and the complete architecture is trained end-to-end. They further improve this architecture in [128] by using a binary mask as an additional input channel. This mask indicates existence of a LiDAR measurement corresponding to a pixel location. Along this they also introduce a novel context aggregation module to limit the error

introduced by missing LiDAR measurements and furthermore in order to tackle the class imbalancing problem they use focal loss [66] for training their DCNN. The last method we compare with is the DCNN proposed by Wang et al. [125]. Similar to the neural network architectures proposed by Wu et al. [127], their network architecture is also based on SqueezeNet. They also Squeeze Excitation blocks [53] after the initial *fire* modules and at the end of the encoder they use an *enlargement* layer, which is based on the Atrous Spatial Pyramid Pooling [21].

In Figure 5.4, we show qualitative semantic segmentation results. Our proposed DCNN is able to segment objects of different classes successfully (top row) and is able to tackle cases where an object is heavily occluded (middle row). We also illustrate a case where our method sometimes over segments a bicyclist into classes *pedestrian* and *bicyclist*. This primarily happens because a person is part of both classes, in one case a person is walking and in the other case a person is riding a bicycle.

In Table 5.2, we report the class wise IoU and mean IoU for different methods. Our proposed DCNN outperforms the existing state-of-the art DCNNs proposed for the same task and has a better IoU for all of the three classes. In the case of *pedestrian*, the increase in IoU is around 70%, for the class *bicyclist* the increase is around 17%, with an overall increase in mean IoU of 16%. These results indicate a remarkable improvement over the existing DCNNs proposed to solve the same task. Comparing the runtime, our DCNN has the largest runtime but considering that scan rate for LiDAR scanners is around 10Hz, our method still provides real time performance.

Comparing the inter class performance, the highest IoU is achieved for the class *car*, whereas the performance for *pedestrian* and *bicyclist* are comparable. A similar trend is evident for other methods as well. This difference in performance has three main reasons, firstly the number of instances of *pedestrian* and *bicyclist* is lesser in comparison to *car*. Secondly, the object in both these classes has a smaller size in comparison to cars and therefore the number of points sampled from their surface is significantly lower in comparison to points sampled from the surface of cars. Due to these reasons, these two classes are under represented, and as mentioned before, we use weight balancing in order to have a large penalty for misclassifying points in these classes. To understand the contribution of this weight balancing, in our ablation study we report results for a model trained w/o weight balancing. The last reason is the over segmentation of points on a bicyclist into classes *bicyclist* and *pedestrian* as shown in Figure 5.4. This misclassification is not a common occurrence but still hampers the overall performance.

5.2.5.1 Ablation Study

In this ablation study, we justify the network design choices thet we mentioned in Section 5.2.2. We first discuss the differences between our dense blocks based

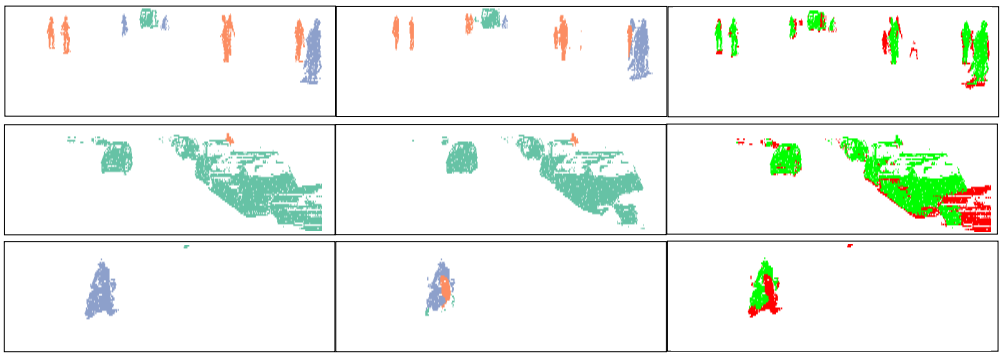


Figure 5.4: An illustration of the semantic segmentation results. In the left column, we show the ground-truth segmentation masks where points belonging to the class *car*, *pedestrian* or *bicyclist* are show in color green, orange and blue respectively. In the middle column, we show the predicted segmentation masks with the same color scheme as the ground-truth masks. To clearly visualize the differences between the ground-truth and predicted masks, in the last we show the correctly segmented points in green color and the misclassified points in color red. The top row illustrates the case where our proposed DCNN is able to successfully segment objects of different classes. The middle row shows a hard case, where a pedestrian is walking behind the cars and is heavily occluded and our method is still able to correctly segment the pedestrian. The bottom row illustrates a case where our method under performs. In some cases, bicyclists are over segmented into the classes *bicyclist* and *pedestrian* due to presence of a person in both classes.

Table 5.2: A comparison with other DCNNs proposed for semantic segmentation of a LiDAR scan. For each method, we report class wise and mean IoU

Method	Car	Pedestrian	Bicyclist	meanIoU	t [ms]
SqueezeSeg [127]	60.9	22.8	26.4	36.7	**8.7**
SqueezeSeg w/ CRF [127]	64.6	21.8	25.1	37.1	13.5
PointSeg [125]	67.4	19.2	32.7	39.7	12
PointSeg w/ RANSAC [125]	67.3	23.9	38.7	43.3	14
SqueezeSegV2 [128]	73.2	27.8	33.6	44.8	-
DBLiDARNet (Ours)	**75.1**	**47.4**	**45.4**	**56.0**	40

Table 5.3: Results for ablation study. For each method we report class wise and mean IoU.

Method	Car	Pedestrian	Bicyclist	meanIoU	t [ms]
100 Layer Tiramisu [59]	74.2	48.7	43.7	55.5	41
TD block [59]	72.2	48.3	41.2	53.9	43
Down-sample 8×	74.1	43.8	39.7	52.5	43
Down-sample width 4×	74.7	45.0	38.6	52.8	66
db_3 depth separable	74.2	**49.2**	36.8	53.4	41
db_3 + db_2 depth separable	73.6	41.2	33.2	49.3	**40**
W/o weight balance	72.4	40.9	35.1	49.5	41
DBLiDARNet	**75.1**	47.4	**45.4**	**56.0**	41

fully convolutional network and the architecture proposed by Jégou et al. [59].

1. Their architecture consists of transition-down block for down-sampling the feature maps. This block implements a composite function comprising of batch normalization, ReLU activation, convolution layer (1×1), dropout and max-pooling. We replace this transition-down block with a max-pooling layer. This decision is based on our empirical findings, which showed that replacing this block, which contains learnable parameters by a max-pooling layer, helps in reducing the parameters while maintaining similar performance.

2. As already mentioned and shown in Figure 5.3, we use depth separable convolution layers instead of convolution layers for dense blocks in the decoder. This again helps in reducing the learnable parameters significantly without reducing the performance. The number of parameters for our proposed network is 2.8M and their architecture is 3.6M. This large difference between the parameters is mainly attributed to depth separable convolution.

The architecture proposed by Jégou et al. [59] consists of five transition down blocks for down-sampling the feature maps 32×. They, therefore, use five up-convolution layers in the decoder along with the same number of dense blocks. Such a high down-sampling rate will result in significant loss of information for reasons discussed before (Section 5.2.2). Therefore, in our implementation of their architecture we only use two transition down blocks instead of five. In Table 5.3 we report both class wise and mean IoU for their architecture and also for a model where we use our architecture but replace the max-pool layers with transition down (TD) blocks. Our proposed architecture outperforms their architecture marginally while using fewer parameters. Using transition down blocks instead of a max-pooling layer leads to a slight decrease in performance as well. These results clearly indicate that our proposed changes help in reducing the parameters while improving the performance.

We trained two different models in order to compare the different down-sampling strategies. For the first model, we down-sample $8\times$ instead of $4\times$ and for the second model we down-sample $4\times$ but only along the width dimension while keeping the height unchanged, similar to [127]. As reported in Table 5.3, for the first model (down-sample $8\times$), the IoU for the class *car* remains comparable but a decrease in performance is observed for the other classes. In comparison to cars, pedestrians and bicyclists are smaller and therefore a large down-sampling rate adversely affects these classes in comparison to other classes. For the second model, similar to the first, a noticeable decrease in performance is observed for both *pedestrian* and *bicyclist* classes. Even though a large down-sampling rate can hamper the performance, especially for the task of semantic segmentation, it is still required for increasing the receptive field as well as making the model efficient considering both the memory and computational requirements. Our proposed strategy of down-sampling the feature maps $4\times$ allows us to exploit the advantages of such operations without losing the crucial information necessary for predicting accurate segmentation masks.

Depth separable convolution is an ingenious way of reducing parameters, but excessively using it can potentially decrease performance. To justify this, we train two models, using a depth separable convolution in the last dense block of the encoder (db_3) and then in last two dense blocks together (db_3 + db_2). This decreases the number of parameters from 2.8M (DBLiDARNet) to 1.9M and 1.4M respectively. In both cases the performance decreases, especially for the second case wherein the decrease is substantial.

To limit the detrimental impact of class imbalancing on the overall performance, we use weight balancing. In the loss function, the contributions made by the under represented classes are multiplied by a large weight, thereby incurring a large penalty if points from these classes are incorrectly classified. The lowest weight is assigned to the *background* class, while weights in increasing order are assigned to classes *car*, *pedestrian* and *bicyclist* respectively. To analyze the impact of weight balancing, we trained a model where we did not use balancing and report results for this in Table 5.3. The decrease in performance is evident for all the classes, where the most under represented class suffers the most with performance decreasing for the class *bicyclist* by 22% and for class *pedestrian* by 5%. These results highlight the necessity of using the weight balancing technique.

5.3 Object Bayes Filter

In Section 5.2, we proposed a novel DCNN architecture for semantic segmentation of a LiDAR scan into different categories. The output of the network is the predicted softmax probabilities of a point in a scan belonging to different categories. Since this prediction is performed independently for different scans, in this section we

introduce a novel Bayes Filter approach to make our pointwise prediction temporally consistent. This approach assumes the scans are sequential with significant overlap and the objective is to leverage over the sequential nature of this information and make our prediction robust to isolated erroneous predictions from the neural network.

The output of the neural network is category wise softmax probability for each point and now we estimate the category wise belief for each point using a binary Bayes filter with *static* state. For each point, we use three separate binary Bayes filters to estimate the belief for each class independently. In Section 4.4, we proposed an HMM based approach for classifying points in a scan as static and dynamic with the same objective of making the predictions temporally consistent. In that approach, we considered transition between the states since objects can switch their state from static to dynamic and vice-versa. In contrast to that, for the case of semantic segmentation, the semantic state of a point is *static*, i.e. it remains the same over time and therefore the current belief only depends on the measurements, which in this case is the neural network output.

To estimate the belief for a class c, for a point $\mathbf{p}^t$ in a scan at time t , we first define a binary random state variable $O_c^t = \{0, 1\}$, where $O_c^t = 1$ indicates that the point belongs to the class c and $O_c^t = 0$ indicates the opposite. Without loss of generality, from now on, we would write $Bel(O_c^t = 1)$ as $Bel(O_c^t)$ and $Bel(O_c^t = 0)$ as $Bel(\neg O_c^t)$. The current belief $Bel(O_c^t)$ depends only on the predictions of the neural network, $\xi_c^{1:t}$, for the class c as shown in Eq. (5.7).

$$Bel(O_c^t) = P(O_c^t \mid \xi_c^{1:t}), \tag{5.7}$$

where, $\xi_c^{1:t}$ are softmax scores for the class c. We define such binary random variables for each class and estimate the belief for each class independently.

Using Bayes rule and Markov assumption, we can rewrite the Eq. (5.7) as following:

$$P(O_c^t \mid \xi_c^{1:t}) = \frac{P(\xi_c^t \mid O_c^t) P(O_c^t \mid \xi^{1:t-1})}{P(\xi^t \mid \xi^{1:t-1})}. \tag{5.8}$$

Using Bayes rule for the term $P(\xi_c^t \mid O_c^t)$, Eq. (5.8) can be modified as following:

$$P(O_c^t \mid \xi_c^{1:t}) = \frac{P(O_c^t \mid \xi_c^t) P(\xi_c^t) P(O_c^t \mid \xi^{1:t-1})}{P(O_c^t) P(\xi^t \mid \xi^{1:t-1})}. \tag{5.9}$$

Similarly, $P(\neg O_c^t \mid \xi_c^{1:t})$ can be written as:

$$P(\neg O_c^t \mid \xi_c^{1:t}) = \frac{P(\neg O_c^t \mid \xi_c^t) P(\xi_c^t) P(\neg O_c^t \mid \xi^{1:t-1})}{P(\neg O_c^t) P(\xi^t \mid \xi^{1:t-1})}. \tag{5.10}$$

We now introduce the log odds notation, where odds of an event x is defined

in Eq. (5.11) and the log odds are defined in Eq. (5.12)

$$\frac{p(x)}{\neg p(x)} = \frac{p(x)}{1-p(x)}, \tag{5.11}$$

$$l(x) = \log\frac{p(x)}{1-p(x)}. \tag{5.12}$$

The odds for a point $\mathbf{p}^t$ having the semantic class c can be estimated by dividing Eq. (5.9) by Eq. (5.10). The odds is defined in Eq. (5.13) and the log odds are defined in Eq. (5.15),

$$\frac{P(O_c^t \mid \xi_c^{1:t})}{P(\neg O_c^t \mid \xi_c^{1:t})} = \frac{P(O_c^t \mid \xi_c^t)}{P(\neg O_c^t \mid \xi_c^t)} \frac{P(O_c^t \mid \xi^{1:t-1})}{\neg P(O_c^t \mid \xi^{1:t-1})} \frac{P(\neg O_c^t)}{P(O_c^t)}, \tag{5.13}$$

$$= \frac{P(O_c^t \mid \xi_c^t)}{1-P(O_c^t \mid \xi_c^t)} \frac{P(O_c^t \mid \xi^{1:t-1})}{1-P(O_c^t \mid \xi^{1:t-1})} \frac{1-P(O_c^t)}{P(O_c^t)}, \tag{5.14}$$

$$l_t(O_c^t) = \log\frac{P(O_c^t \mid \xi_c^t)}{1-P(O_c^t \mid \xi_c^t)} + l_{t-1}(O_c) - l_0(O_c), \tag{5.15}$$

where, the current measurement is defined as following:

$$P(O_c^t \mid \xi_c^t) = \xi_c^t. \tag{5.16}$$

In Eq. (5.15), $l_t(O_c^t)$ are the log odds for the belief at time t, the first term on the right side in Eq. (5.15) are the log odds for the current measurement, $l_{t-1}(O_c)$ are the log odds for the previous belief and $l_0(O_c)$ are the log odds for the initial belief. Through this formulation, our inference not only depends on the current measurement ($P(O_c^t \mid \xi_c^t)$), but also on the previous measurements, incorporated through the recursive term $l_{t-1}(O_c)$. To enable this recursive behavior, data association between points in consecutive scans is required and for this we use our method of estimating pointwise motion as proposed in Chapter 4. For inferring the motion state using HMM (Section 4.4), we performed data association by aligning scans using the estimated motion and choosing the nearest point on the basis of Euclidean distance as the corresponding point. For this case, we follow the exact same approach for associating points in consecutive scans. As mentioned before, we estimate $l_t(O_c^t)$ for each class separately and for the inference we choose the class with the largest odds.

5.3.1 Results

To evaluate our proposed Bayes filter approach, we use the KITTI tracking benchmark. The benchmark contain 20 sequences and to evaluate our approach on all of the sequences, we split the sequences into two different sets. We train our network on both sets separately and use the other set for testing i.e. we train a model on the first set and test the learned model on the second set and then train on the

Table 5.4: Splitting of sequences in KITTI tracking benchmark

Seq. ID	# of scans	Cars	Pedestrians	Bicyclists
Set 1				
0	153	528	21	153
4	313	908	65	60
5	296	1307	0	139
6	269	661	0	0
8	388	1334	0	0
9	802	3135	29	0
10	293	673	30	14
11	372	3579	197	0.0
19	1058	1411	6595	306
Set 2				
2	229	1127	177	75
3	143	38	0	0
7	799	2488	67	0
12	77	142	64	42
13	339	123	1096	237
14	105	523	120	0
15	375	899	751	537
16	208	832	2019	271
17	144	0	776	100
18	338	1413	0	0
20	836	6244	0	0

second set and test on the first set. While splitting the sequences we assure the number of scans and the instances of the different classes have similar distribution. In Table 5.4, we report the number of scans in each sequence and the number of instances of each class in a given sequence. Among the three classes, the number of instances of class *bicyclist* is the minimum and instances of class *car* in large numbers is consistently prevalent across sequences. As mentioned before, the number of point measurements from the surface of pedestrians and bicyclists is significantly less in comparison to the measurements from cars. Therefore, due to the limited instances and smaller size, segmenting these classes is challenging.

For training the network, we use our proposed network with the exact same parameters as discussed in Section 5.2.4, with one difference. In this case, the input resolution of the images are $64 \times 324 \times 5$, in comparison to $64 \times 512 \times 5$. For evaluating the proposed Bayes filter method, we use our network as the baseline method and report comparison with the segmentation results from the network. In Chapter 4, we proposed a method for estimating pointwise motion, which is used

in the proposed Bayes filter approach for the task of estimating data association between points in two consecutive scans. This data association is required for estimating the state of a point in a recursive fashion. Our method of estimating pointwise motion relies on the correspondences between sparse set of keypoints, and in Section 4.3.2.4, we compared the accuracy of the estimated motion, where we used our proposed learned feature descriptor in Chapter 3 and a handcrafted descriptor, for estimating the motion. Across different metrics, the motion estimated using our learned descriptor was more accurate.

To learn this feature descriptor, we used the first ten sequences of the KITTI tracking benchmark (Section 3.2.3), where surface patches around randomly sampled keypoints were part of the training set. Using the same learned feature descriptor for estimating pointwise motion on the same ten sequences makes the performance biased towards the learned descriptor, even though a large overlap between randomly sampled keypoints that were part of the training set and the keypoints used for the estimating motion is unlikely. To compare the performance between the learned and the handcrafted descriptor, we show results for both cases. A superior performance for the learned descriptor is expected on the first ten sequences for the aforementioned reasons but for the remaining sequences, this comparison is unbiased.

In Figure 5.5, we illustrate the differences in the segmentation results for a sequence of six consecutive scans. In the top two rows, we show results for our proposed neural network and in the bottom two rows we show results for our proposed Bayes filter approach. In the case of the neural network, the points on a car are correctly classified in the first scan but in the next few scans, points on the same car are misclassified as background. For the same scans, our proposed Bayes filter is able to consistently classify points on the car correctly. These results clearly illustrate that our Bayes filter approach successfully leverages over the sequential nature of the input data to correct the segmentation, thereby making our predictions temporally consistent. For these results, the pointwise motion required for data association was estimated using the learned descriptor and the scans are from a sequence that was not used for generating the training data for learning the descriptor.

In Table 5.5, we report class wise IoU for different sequences, for both our DCNN and the Bayes filter approach (handcrafted and learned descriptors). In the cases where no instances of a class is observed, we do not report results as well (Indicated by a dash sign). Analyzing the neural network predictions, our DCNN is consistently able to segment *cars* in comparison to the other classes. Since the LiDAR scanner is mounted on a vehicle, it shares the same space where other vehicles operate, in comparison to pedestrians or bicyclists, which are either walking or biking on a sidewalk or a bike lane. This also explains why the instances of cars outnumbers pedestrians or bicyclists by a significant margin. In the cases of pedestrians, a

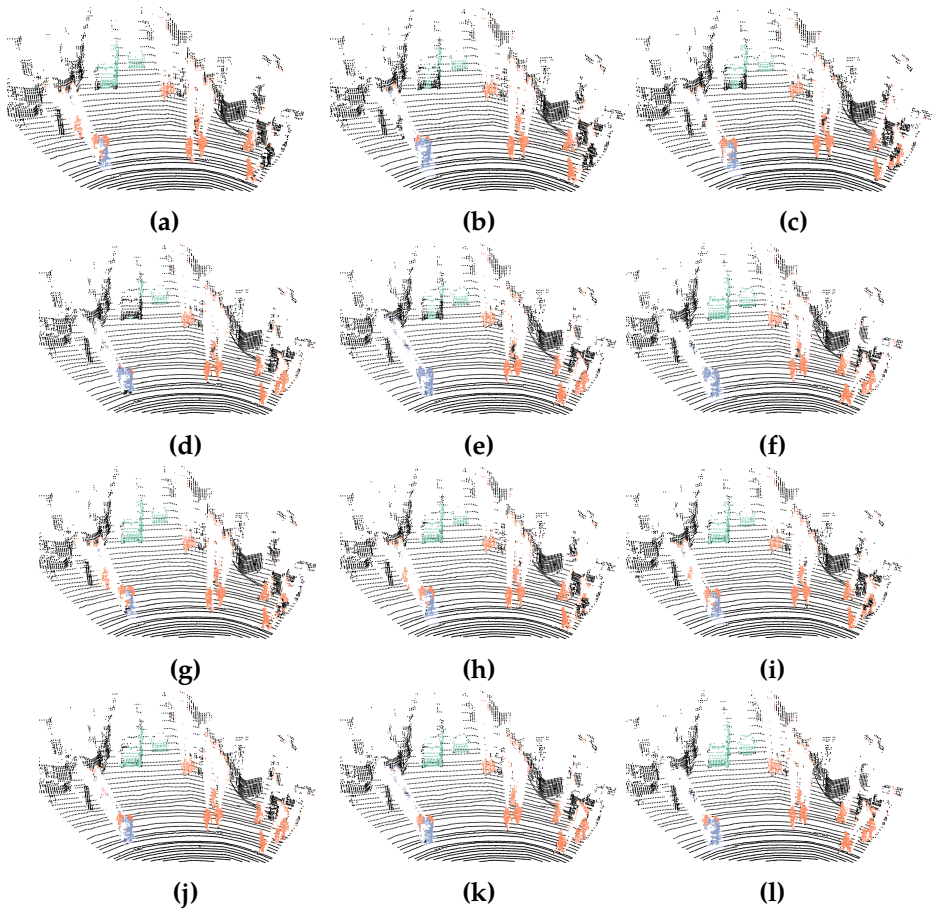


Figure 5.5: Illustration of semantic segmentation with the object Bayes filter. In the top two rows ((a)-(f)), we show the output of our proposed DCNN for six consecutive scans. In the top left image, the points on a car (top left) are correctly classified, but in subsequent scans, the points on the same car are first partially ((b)-(c)) and then completely ((d)) misclassified as background. In the bottom two rows, we show the output of our proposed binary Bayes filter for the same six consecutive scans. For all of the six scans, the points on the same car are correctly classified. These results clearly illustrate that our proposed Bayed filter method is able to successfully mitigate the sporadic erroneous predictions from the neural network.

Table 5.5: Class wise IoU for DCNN and the binary object Bayes filter (learned and handcrafted descriptor)

Seq. ID	DBLiDARNet			Object Bayes Filter (SHOT descriptor)			Object Bayes Filter (learned descriptor)		
	Cars	Pedestrians	Bicyclist	Cars	Pedestrians	Bicyclist	Cars	Pedestrians	Bicyclist
0	76.2	2.0	**29.6**	79.2	2.0	23.6	**79.3**	2.0	24.7
2	54.9	37.0	0.0	**55.3**	**46.9**	0.0	55.2	46.4	0.0
3	75.2	-	-	75.5	-	-	**75.8**	-	-
4	66.6	40.8	35.2	69.1	**47.4**	**53.2**	**69.2**	47.3	52.9
5	70.1	-	-	70.0	-	-	70.0	-	-
6	87.2	-	-	87.1	-	-	87.1	-	-
7	83.2	28.2	-	83.5	32.7	-	83.6	**34.9**	-
8	66.9	-	-	69.9	-	-	**70.8**	-	-
9	71.9	18.6	-	72.9	**21.6**	-	**73.1**	21.4	-
10	72.4	0.0	0.0	**75.1**	0.0	0.0	74.1	0.0	0.0
11	88.4	15.6	-	**89.6**	15.3	-	**89.6**	**16.8**	-
12	51.5	0.0	**4.0**	**58.5**	0.0	1.6	58.5	0.0	1.8
13	24.2	50.7	39.5	**31.3**	50.6	41.1	31.1	**51.7**	**44.2**
14	**89.6**	40.2	-	86.3	42.6	-	86.4	**43.4**	-
15	83.9	70.1	5.0	**85.7**	72.5	5.0	85.4	**73.3**	5.0
16	63.8	75.3	54.5	**64.1**	77.0	**60.7**	63.5	**77.1**	60.6
17	-	81.8	0.0	-	**83.7**	0.0	-	**83.7**	0.0
18	84.7	-	-	84.7	-	-	84.8	-	-
19	68.4	66.2	36.9	74.0	66.1	37.8	**74.1**	**66.3**	**39.9**
20	69.1	-	-	69.4	-	-	69.3	-	-

high IoU is achieved for the cases where pedestrians are in close proximity of the vehicle collecting the sensor data, for instance on a crowded small street or at an intersection. For some sequences, the IoU for the class bicyclist is zero. In these cases, the majority of times these objects are either far from the sensor or occluded and in rare cases they are misclassified as pedestrians. In the case of LiDAR data, with the increase in distance the data gets sparser and especially in the case of bicyclists or pedestrians, the surface is smaller in comparison to cars and therefore there are not enough point measurements to a make an accurate prediction.

Comparing the DCNN results with the Bayes filter approach, across different sequences and classes, an improvement in IoU is consistently observed after using the Bayes filter approach. This improvement is achieved irrespective of whether learned or handcrafted descriptor is used. For most cases the improvement in IoU is around 4% to 9% but an improvement of 27% is achieved for class *pedestrian* in sequence 2 and a staggering improvement of 51% is achieved for class *bicyclist* in sequence 4. For a couple of isolated cases, a decrease in IoU is observed after using the filter approach. The implicit assumption of our Bayes filter approach is that the predictions from DCNN is seldom wrong and for these cases, the filter uses previous knowledge to correct those predictions. In the rare cases where this assumption is violated, the information accumulated by the filter spurs from incorrect measurements and therefore the filter approach needs multiple correct predictions from DCNN to improve its knowledge in comparison to a single prediction needed by DCNN. For instance, in the sequence 0, points on a bicyclist were labeled as pedestrian often, causing the Bayes filter to accumulate the incorrect predictions.

Analyzing the impact of the feature descriptor used for estimating pointwise motion on the final segmentation results, we now compare the performance of our learned descriptor with the handcrafted one. In the case of the handcrafted descriptor, we use SHOT, which among the multiple handcrafted descriptors that we analyzed in Chapter 3, showed the most encouraging performance. For the learned descriptor, we use our descriptor learned without metric. As shown in Chapter 3, this descriptor offered the right balance between the computation time complexity and performance. These descriptors are used for estimating motion for aligning a pair of consecutive scans, and aligned scans are then used for estimating data association required for our Bayes filter approach. The performance is either comparable or attaining a marginal improvement after using the learned descriptor. Focusing on the sequences that were not used for learning the descriptor (10-20), a consistent and noticeable improvement is evident, especially for the classes *pedestrian* and *bicyclist*. As shown in Table 4.14 in Chapter 4, the improvement in alignment accuracy is marginal after using the learned descriptor and therefore similar improvement is expected for this case as well.

Through these qualitative and quantitative results, we show the importance of

our proposed static binary Bayes filter approach. We exploit the sequential nature of the input data, making our predictions temporally consistent and report a persistent improvement in IoU across different sequences and classes. A further improvement is attained after incorporating our learned descriptor in our method for estimating data association between points in consecutive scans.

5.4 Combining Motion and Semantic Cues

In Figure 5.6, we illustrate the framework of our proposed method of classifying points in a 3D LiDAR scan into the classes *non-movable*, *movable* or *dynamic*. The input to our method is a pair of consecutive LiDAR scans, where the first scan is input to the DCNN architecture thet we proposed for semantic segmentation in Section 5.2.2. The output of the DCNN, is the confidence score of a point belonging to an object semantic category (*cars*, *pedestrians* or *bicyclist*), which we collectively call the objectness score. Both scans are inputs to our method of estimating pointwise motion proposed in Section 4.3. The output of this method is a dense motion field, which we then compare with the odometry to estimate the dynamicty score, i.e. the confidence score of point being either dynamic or static. The pointwise objectness and dynamicity score is input to the HMM based method that we propose in this section and the output of this method is the classification of a LiDAR scan into the desired classes.

In Section 4.4, we proposed an HMM based approach for inferring the motion state of points in a 3D LiDAR scan. We now extend that approach for inferring the motion and the semantic states together for points in a LiDAR scan. For a point $\mathbf{p}^t$ in a scan at time t, we define a random state variable $X^t = \{\mathtt{n}, \mathtt{m}, \mathtt{d}\}$, where $\mathtt{n}$, $\mathtt{m}$ and $\mathtt{d}$ are the possible values for X representing the states: **n**on-movable, **m**ovable and **d**ynamic respectively. The objective is to estimate the belief of the current state.

$$Bel(X^t) = P(X^t \mid \boldsymbol{\tau}^{1:t}, \gamma^{1:t}), \tag{5.17}$$

where, the current belief $Bel(X^t)$ depends on the motion measurements $\boldsymbol{\tau}^{1:t}$ and object measurements $\gamma^{1:t}$. In Section 5.3, we proposed a binary state Bayes filter with *static* state for accumulating the neural network predictions $\xi_c^{1:t}$ with the objective of having temporally consistent inference of the semantic states and the variable $O_c^{1:t}$ represents the output of this binary Bayes filter. The object measurements are the output of this Bayes filter. In Eq. (5.18), we show the simplification of Eq. (5.17) using the Bayes rule and the Markov assumption. Since the motion ($\boldsymbol{\tau}$) and the semantic cues (ξ_c and O_c) are independent, we can further simplify Eq. (5.18) as Eq. (5.19). Since X is a discrete random variable, Eq. (5.19) can be rewritten as Eq. (5.20).

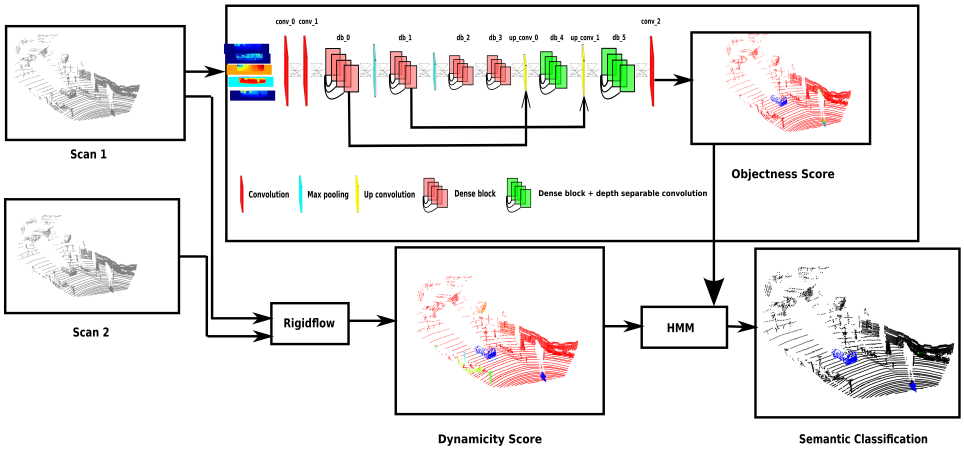


Figure 5.6: An illustration of the framework of the proposed method for semantic classification of a LiDAR scan into classes *non-movable*, *movable* or *dynamic*. The input to our method is a pair of consecutive LiDAR scans. The first scan is the input to our proposed DCNN the output of which is the pointwise objectness score. This score indicates the confidence measure of a point being *non-movable* or *movable*. Both scans are input to our method of estimating pointwise motion from Chapter 4 and, using this motion, we estimate the dynamicity score. This score indicates the confidence measure of a point being dynamic or static. The pointwise objectness and dynamicity score is input to our proposed HMM based method, which combines the two scores to classify points in the first scan into the classes *non-movable*, *movable* and *dynamic*.

$$Bel(X^t) = P(\boldsymbol{\tau}^t, \gamma^t \mid X^t) \int P(X^t \mid X^{t-1}) Bel(X^{t-1}) dX^{t-1}, \tag{5.18}$$

$$= P(\boldsymbol{\tau}^t \mid X^t) P(\gamma^t \mid X^t) \int P(X^t \mid X^{t-1}) Bel(X^{t-1}) dX^{t-1}, \tag{5.19}$$

$$= P(\boldsymbol{\tau}^t \mid X^t) P(\gamma^t \mid X^t) \sum_{X^{t-1}=\{\mathtt{m},\mathtt{n},\mathtt{d}\}} P(X^t \mid X^{t-1}) Bel(X^{t-1}). \tag{5.20}$$

Here, $P(\boldsymbol{\tau}^t \mid X^t)$, which we call the dynamicity score, is defined in the exact same way as in Eq. (5.21) in Chapter 4,

$$P(\boldsymbol{\tau}^t \mid X^t) = \mathcal{N}(\boldsymbol{\tau}^t; \hat{\boldsymbol{\tau}}^t, \Sigma), \tag{5.21}$$

where, $\hat{\boldsymbol{\tau}}^t$ is the odometry and Σ is the covariance. In Section 5.3, we the used

log-odds formulation for estimating the current belief of O_c^t as following:

$$Bel(O_c^t) = P(O_c^t \mid \xi_c^{1:t}), \tag{5.22}$$

$$l_t(O_c) = \log \frac{P(O_c^t \mid \xi_c^t)}{1 - P(O_c^t \mid \xi_c^t)} + l_{t-1}(O_c) - l_0(O_c), \tag{5.23}$$

where, $P(O_c^t \mid \xi_c^t)$ is defined as following:

$$P(O_c^t \mid \xi_c^t) = \xi_c^t. \tag{5.24}$$

This belief was defined separately for each class c. In the current formulation, our objective is to distinguish non-movable (background) from movable (*car*, *pedestrian* or *bicyclist*). Therefore, we introduce a binary random variable $O^t = \{0, 1\}$, where $O^t = 1$ indicates that a point is movable i.e. it belongs to one of the object classes and $O^t = 0$ indicates that a point is non-movable. For estimating $Bel(O^t)$, we choose the maximum value from beliefs of different classes.

The likelihood of the object measurements is defined as following:

$$P(\gamma^t \mid X^t) \propto \begin{cases} Bel(\neg O^t) & \text{if } X^t = \mathtt{n} \\ Bel(O^t) & \text{if } X^t = \mathtt{m} \\ s \cdot Bel(O^t) & \text{if } X^t = \mathtt{d} \end{cases} \tag{5.25}$$

As the neural network is trained to predict the non-movable (*background*) and movable (*car*, *pedestrian* or *bicyclist*) classes the first two cases in Eq. (5.25) are straightforward. For the case of dynamic objects, we scale the prediction of the movable class by a factor $s \in [0, 1]$. This scaling factor approximates the ratio of number of dynamic objects in the scene to the number of movable objects. This ratio is environment dependent, for instance, on a *highway*, the value of s will be close to 1, since most of movable objects will be dynamic. For our experiments, through empirical evaluation, we chose the value of $s = 0.8$.

5.4.1 Dynamic Log-Odds

In the previous section, we proposed an HMM based approach for classifying points in a LiDAR scan into the classes *non-movable*, *movable* or *dynamic*. The current belief was dependent on the current motion measurements and the semantic information accumulated by the static binary Bayes filter proposed in Section 5.3. In Section 5.3.1, we showed the necessity of such a filtering approach, where for multiple sequences a consistent improvement was achieved. In this section, we explore the option of using a binary Bayes filter with *static* state for motion measurements. As mentioned in Section 5.3, the *static* binary Bayes filter relies on the *static* world assumption i.e. the state of a random variable is expected to remain unchanged for long periods of time. This assumption is completely satisfied in the case of semantic cues, since the

transition between the semantic categories is highly unlikely. In the case of motion measurements indicating towards the motion state of an object i.e. whether an object is static or dynamic, this assumption is violated when the state switches from static to dynamic or vice-versa. Since the number of times that the state switches is often less when compared to the times that the state remains the same, in this section we introduce a binary random state variable $D^t = \{0, 1\}$, where $D^t = 1$ indicates a point $\mathbf{p}^t$ in a scan at time t is *dynamic* and $D^t = 0$ indicates the opposite.

The current belief $Bel(D^t)$ depends only on the motion measurements $\boldsymbol{\tau}^{1:t}$ as shown in Eq. (5.26).

$$Bel(D^t) = P(D^t \mid \boldsymbol{\tau}^{1:t}), \tag{5.26}$$

where, $\boldsymbol{\tau}^{1:t}$ is the estimated pointwise motion. Following the detailed derivation of the binary state Bayes filter in Section 5.3, a similar log-odds formulation for estimating the $Bel(D^t)$ is shown in Eq. (5.27).

$$l_t(D) = \log\frac{P(D^t \mid \boldsymbol{\tau}^t)}{1 - P(D^t \mid \boldsymbol{\tau}^t)} + l_{t-1}(D) - l_0(D). \tag{5.27}$$

To estimate $P(D^t \mid \boldsymbol{\tau}^t)$ we use the Bayes rule like following:

$$P(D^t \mid \boldsymbol{\tau}^t) = \frac{P(\boldsymbol{\tau}^t \mid D^t)P(D^t)}{P(\boldsymbol{\tau}^t \mid D^t)P(D^t) + P(\boldsymbol{\tau}^t \mid \neg D^t)P(\neg D^t)}, \tag{5.28}$$

where, $P(\boldsymbol{\tau}^t \mid D^t)$ is the likelihood of the measurement which is estimated using the Eq. (5.21) and $P(D^t)$ is the prior.

The log-odds value for a particular state increases as more measurements supporting that state are observed. For instance, the log-odds value of a point being *dynamic* will increase continuously, if the estimated motion of a point is significantly different than the odometry for multiple scans. For increasing the log-odds value for the opposite state i.e. to switch the state, again, multiple measurements suggesting the opposite will be required. Due to this, the switching of states in this formulation is a slow process and to overcome this we use a threshold for a possible maximum and minimum log-odds value. This threshold enables our method to switch states faster than the original formulation but also makes our method robust to isolated erroneous measurements.

The dynamic log-odds formulation is incorporated within the original HMM model by modifying Eq. (5.17) in the following way.

$$Bel(X^t) = P(X^t \mid \gamma^{1:t}, \zeta^{1:t}), \tag{5.29}$$

where, $\zeta^{1:t}$ is the output of the Bayes filter proposed above. Similar to before, using the Bayes rule, the Markov assumption and assuming independence between the variables associated with the motion and the semantic measurements, Eq. (5.29) can be simplified as follows:

$$Bel(X^t) = P(\gamma^t, \zeta^t \mid X^t) \int P(X^t \mid X^{t-1}) Bel(X^{t-1}) dX^{t-1}, \tag{5.30}$$

$$= P(\gamma^t \mid X^t) P(\zeta^t \mid X^t) \int P(X^t \mid X^{t-1}) Bel(X^{t-1}) dX^{t-1}, \tag{5.31}$$

$$= P(\gamma^t \mid X^t) P(\zeta^t \mid X^t) \sum_{X^{t-1}=\{\mathtt{m},\mathtt{n},\mathtt{d}\}} P(X^t \mid X^{t-1}) Bel(X^{t-1}), \tag{5.32}$$

where, $P(\zeta^t \mid X^t)$ is estimated as following:

$$P(\zeta^t \mid X^t) \propto \begin{cases} Bel(\neg D^t) & \text{if } X^t = \mathtt{n} \\ Bel(\neg D^t) & \text{if } X^t = \mathtt{m} \\ Bel(D^t) & \text{if } X^t = \mathtt{d} \end{cases} \tag{5.33}$$

For the state $X_t = d$, we use the current belief $Bel(D^t)$ and for the other states, where the object is expected to be static, we use $Bel(\neg D^t)$.

5.4.2 Results

To evaluate our proposed approach, we use the sequences from the KITTI tracking benchmark. In Section 4.4, we proposed a method for classifying points in a scan into the classes *static* and *dynamic*. Using that method, we were able to classify majority of dynamic parts of the environment correctly (high recall) but our method also misclassified a small set of static points as *dynamic*, resulting in low precision and IoU. By combining the motion and semantic cues in this chapter, we aim to reduce the misclassification of static points and therefore use our method from Section 4.4 as a baseline for comparison. We proposed two different ways of using the motion measurements, i.e. to use the dynamicity score from the current scan (Eq. (5.17)) or the dynamic log-odds value (Eq. (5.33)) and we report the comparison between these cases.

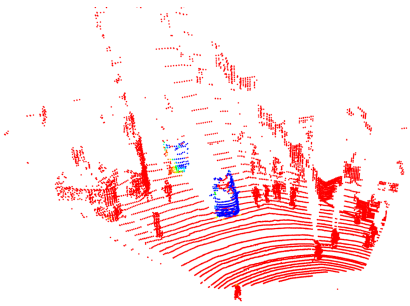
In Chapter 4, we proposed a method for estimating pointwise motion, which is used for two different tasks in our method of semantic classification proposed in this chapter. Firstly, it provides the motion cues, i.e. the dynamicity score, where the estimated motion is part of the likelihood term (Eq. (5.21)) indicating the likelihood of a point being dynamic and secondly, it is used for providing the dense data association between points in consecutive scans. This data association is required for estimating the state of a point in a recursive fashion. We report classification results for both descriptors. The performance on the first ten sequences is expected to be in the favor of the learned feature descriptor for the reasons mentioned in Section 5.3.1 and, similar to before, the comparison is unbiased for the remaining sequences. In this comparison, we use the dynamic log-odds value as a source of motion cues instead of the dynamicity score.

In this chapter, we proposed a novel method for classifying points in a LiDAR scan into the classes *non-movable*, *movable* or *dynamic*. The KITTI tracking benchmark provides a ground-truth 3D bounding box for objects of different categories, which can be collectively labeled as *movable*. To get pointwise ground-truth labels, we use the same approach as discussed in Section 5.3.1, where all of the points inside the bounding box are given the same label as the bounding box.

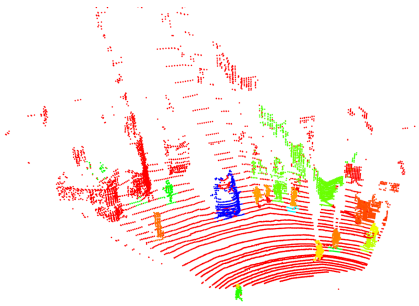
The benchmark only provides the ground-truth semantic category not the ground-truth motion state i.e. whether an object is *dynamic* or *static*. Hand labeling all of the objects in the entire benchmark is an exhausting task and therefore we developed a method for augmenting the labels provided by the benchmark with the ground-truth motion state. Our method relies on the ground-truth odometry provided by the benchmark along with the ground-truth tracking ID. Using the ground-truth odometry $\mathcal{O}_t^{t+1}$, we project points P_t^k for an object k in the scan at time t to the scan at time $t+1$. We then calculate the IoU between the projected points $\hat{P}_{t+1}^k$ and the ground-truth object, P_{t+1}^k. A high IoU value indicates that objects in consecutive scans can be aligned using the odometry and therefore they are *static* and a low value indicates the opposite. This method provides an initial set of ground-truth labels, which are then further refined by human intervention. Since our method implicitly relies on the assumption that the motion of a dynamic object is large enough to cause a mismatch after projecting the points using the odometry, only movable objects like cars and bicyclists can be labeled reliably. Therefore, only points belonging to an object labeled as *car* and *bicyclist* by the benchmark are labeled *movable*. The KITTI tracking benchmark also provides ground-truth bounding box annotation for other movable object categories, like *tram* or *truck*, among others. Due to the limited number of instances of these objects, they are not evaluated on the benchmark, and in our evaluation, we also ignore objects of these categories. The remaining points, i.e. points not belonging to any object for which the benchmark provides annotation is considered *non-movable* or *static*.

Among different sequences in the benchmark, we report results only for the sequences that consists of ground-truth *dynamic* objects, since evaluation for different *movable* objects is already presented in Section 5.3.1. In sequence 20 of the benchmark, objects are moving at a slow speed and are regularly switching states between *static* and *dynamic* and, therefore, estimating a reliable ground-truth label using our method is non-trivial. Therefore, we ignore this sequence for evaluation due to the lack of reliable ground-truth labels.

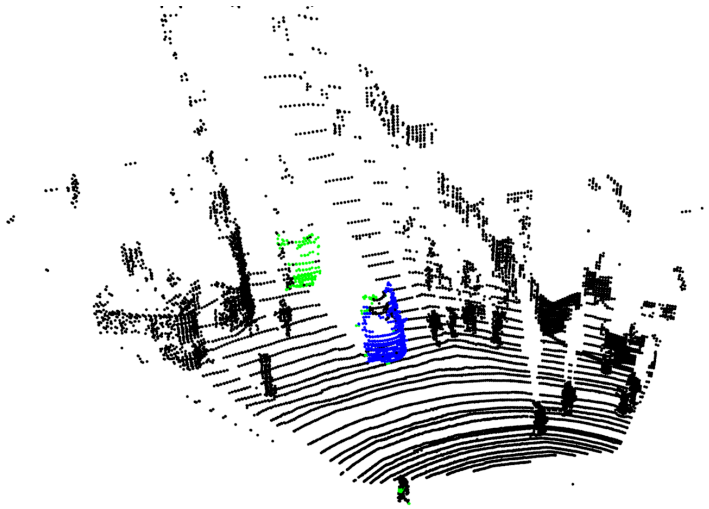
In Figure 5.7, we illustrate the semantic classification of a LiDAR scan into the classes *non-movable* (black), *movable* (green) and *dynamic* (blue). In Figure 5.7a & 5.7b, we illustrate the objectness and the dynamicity score, respectively, which are then combined together in our proposed method to estimate the desired classification, illustrated in Figure 5.7c. In Table 5.6, we report class wise IoU for our method proposed in Section 4.4, where we classified points in a scan into the classes *static*



(a) Objectness score



(b) Dynamicity score



(c) Semantic classification of a 3D LiDAR scan

Figure 5.7: An illustration of the semantic classification of a LiDAR scan from the KITTI tracking benchmark, using the proposed method. In figure (a), we show the objectness score, which is the output of our proposed DCNN. The blue color indicates a high confidence value and the red color indicates the opposite. In figure (b), we illustrate the dynamicity score, which is estimated by comparing the pointwise motion with the odometry measurements. Similar to the objectness score, a high confidence value is indicated by the color blue and a low value is indicated by the color red. In figure (c), is the output of our method, where points in a LiDAR scan are classified into the the classes *non-movable* (black), *movable* (green) or *dynamic* (blue). Points with high objectness and dynamicity score are correctly classified as *dynamic* and points with high objectness but low dynamicity score are correctly classified as movable. Points belonging to non-movable structurs that have a high dynamicity score are correctly labeled as non-movable, thereby illustrating the importance of combining motion and semantic information.

and *dynamic*, using only the motion cues and our method proposed in this chapter, where we combine learned semantic cues with the motion cues to classify points into the classes *non-movable*, *movable* or *dynamic*. For the latter method, we compare the performance between the two sources of motion cues i.e. the dynamicity score and the dynamic log-odds formulation and also between the learned and handcrafted descriptors. For the descriptor comparison, we use the log-odds formulation. For the same set of experiments, we also report the class wise precision and recall in Table 5.7 & 5.8 respectively. For all of these experiments, only the parts of a LiDAR scan that overlaps with the field-of-view of the camera is used for the evaluation.

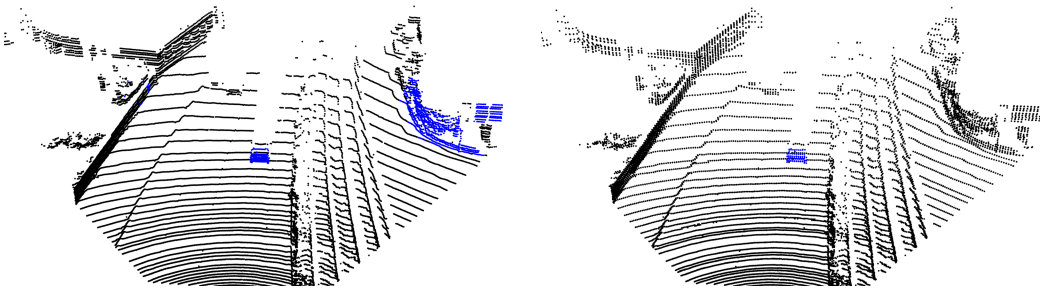
The results of our method proposed in Section 4.4 for the KITTI tracking benchmark is similar to the results reported for dataset from Moosmann and Stiller in Section 4.4.1. For the majority of the sequences in the benchmark, dynamic points are correctly classified (high recall) but the precision and IoU is low due to the misclassification of static points as *dynamic*. For 13 out of 18 sequences, the IoU for the *static* class is greater than 85%, which shows only a small fraction of static points are misclassified. Since the number of dynamic points are significantly lower than the number of static points, this small fraction leads to a significantly low precision and IoU for the dynamic class. For the sequence with low static IoU (lower than 85%), the estimated motion for the points used for evaluation is erroneous. Our method of estimating motion is a feature based method and tends to perform sub-optimally when feature correspondences are plagued with large number of outliers. This mainly happens when a scan contains repeated structures, for instance, a thin metal railing dividing the road lanes or when scan points are far away and the data gets sparser.

One of the main objectives of combining the semantic and motion cues is to limit the misclassification of non-movable structures like metal railings or building walls as *dynamic*. As reported in Table 5.6, the IoU for dynamic class is consistently improved after incorporating the learned semantic cues. This improvement is mainly attributed to increased precision (Table 5.7) in classification of dynamic points, as misclassification of non-movable structures significantly decreases. Comparing the recall values, for majority of the sequences, the performance is either improved or remained comparable after incorporating the semantic information. In Figure 5.8, for a single scan, we compare the classification results for our method proposed in Section 4.4 and our method (dynamic log-odds) proposed in this chapter. These quantitative and qualitative results clearly highlight the advantages of combining semantic and motion cues, as a consistent noticeable improvement is achieved for all the sequences, except one. For the first sequence, the IoU decreases after incorporating the semantic cues. In this sequence, a bicyclist is misclassified as a pedestrian for majority of the scans and since we ignore pedestrians in this evaluation due to lack of reliable ground-truth labels, the bicyclist is not classified dynamic. In Chapter 5 similar results were reported for the same sequence. After

using our proposed Bayes filter approach for making predictions of the neural network temporally consistent, the IoU for the class *bicyclist* decreased (Table 5.5). The results from this sequence highlight an advantage of model-free approaches that we proposed in Chapter 4. Using such approaches, arbitrary different dynamic objects can be correctly classified.

For the classes *dynamic* and *movable*, a higher IoU is consistently achieved when we use dynamic log-odds as motion cues instead of dynamicity score, whereas for *non-movable* class, the performance is comparable. This increase in IoU is again mainly attributed to improved precision, as shown in Table 5.7. For dynamic class, across all of the sequences, a better IoU is achieved after using the log-odds formulation. Similarly for movable class, the performance improves for the majority of the sequences after using the dynamic log-odds formulation. In Figure 5.9, we compare results for a sequence of four consecutive scans for both methods. These quantitative and qualitative results clearly illustrate that using the log-odds formulation to combine information from previous scans, makes our method robust to isolated erroneous measurements. Comparing the recall values, using dynamicty score results in a better performance for majority of the sequences. This is mainly due to a key characteristic of *static* binary Bayes filter, which is the assumption that state of a variable remains unchanged for longer periods of time. To enable swift switch in states, we use a threshold for maximum and minimum log-odds value, but using such filtering approach reduces the necessary flexibility required for switching states. The decrease in IoU for *movable* class for sequence 8 is also because of this. A movable object is misclassified as *dynamic* due to erroneous motion and when the motion is correctly estimated, the state switches to *movable* faster for the dynamicity score in comparison to the log-odds formulation.

For all of the above discussed experiments, we used our learned feature descriptor proposed in Chapter 3, for estimating data association between sparse keypoints, required for estimating the motion. As already discussed before, for the first ten sequences (0-9), the performance is expected to be in favor of the learned descriptor because keypoints used for learning the descriptor were randomly sampled from these sequences. Even though a large overlap between randomly sampled keypoints used for estimating the motion and the keypoints used for training the descriptor is unlikely, for this comparison, we primarily focus on the evaluation of the 9 sequences from sequence number 10 to 19. For *non-movable* class, the performance for both descriptors are comparable. Comparing the IoU of *movable* class, for 3 out of 8 sequences (no movable object in sequence 12), results are better for the SHOT descriptor and for the remaining sequences, the learned feature descriptor either performs better or the results are comparable. For sequence 10, an improvement of 78% is achieved after using the learned descriptor. This sequence contains a single movable object that is misclassified as dynamic, in the case when SHOT descriptor is used for estimating the motion. This misclassification is reflected in the reported



(a) Objectness score **(b)** Dynamicity score

Figure 5.8: In Section 4.4, we proposed a HMM based method for inferring the motion state of points in a scan, solely based on motion cues. The image on the left shows the output of that method, where points on the wall on the right are incorrectly classified as *dynamic*. On the right, we show the output of our method proposed in this chapter. By combining the learned semantic cues with the motion cues, points on the wall are correctly classified as *non-movable*.

recall values. For sequence 16, SHOT descriptor performs significantly better (+33%). The sequence contains a couple of heavily occluded movable objects and for one such object estimated motion is erroneous when our learned descriptor is used. The low recall values reflect this case as well. Comparing the IoU for dynamic class, for 2 out of 9 sequences, performance is better for the SHOT descriptor, and for the remaining 7 sequences, performance of our learned descriptor is either better or comparable. For sequences 13 and 14, an improvement of over 70% is achieved after using the learned descriptor. For both sequences, the motion estimation is erroneous after using the SHOT descriptor, causing movable objects to be misclassified as *dynamic* and thereby resulting in low precision.

For some isolated cases our method performs poorly. In sequence 3, movable points are not detected. The majority of movable objects in the sequence are dynamic, which are correctly classified. The remaining movable objects are far away and for most scans they are not segmented by the DCNN and in the rare cases when they are segmented, they are classified as dynamic due to lack of enough keypoint correspondences, resulting in incorrect motion estimation. The classification of dynamic points for sequence 12 is low across different methods. This sequence contains a dynamic car and a bicyclist. Among these two objects, bicyclist is not segmented by the DCNN and therefore it is not classified as dynamic. The car is segmented by the DCNN but is classified as static for most scans, partially due to its slow speed while turning. For the last part of the sequence, it is correctly classified as dynamic, which is reflected in non-zero IoU value for the case of dynamicity score. In the case of log-odds, the transition between the states is slower and therefore the car is always misclassified as dynamic.

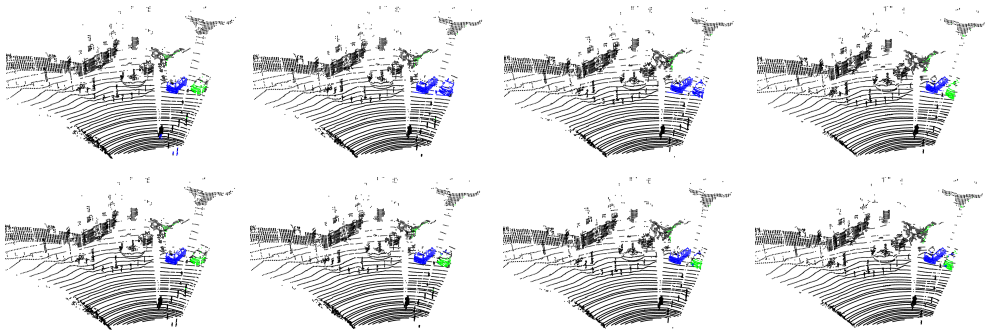


Figure 5.9: In the top row, we show classification results for four consecutive scans, where our proposed method used the motion measurement from the current scan (Section 5.4). A parked car is correctly classified as *non-movable* in the first scan (top left image) but incorrectly classified as *dynamic* in the next two scans due to erroneous motion measurements. In the bottom row, we show classification results for the same four consecutive scans. In this case, we use the proposed dynamic log-odds formulation (Section 5.4.1) and the parked vehicle is now correctly classified as *non-movable*. These results clearly support our initial claim that having such a formulation makes our method robust to isolated erroneous measurements.

5.5 Related Work

One of the main focus of this chapter was semantic segmentation of a 3D LiDAR scan. With the advent of deep neural networks, a significant progress has been made towards solving a variety of tasks, including the task of semantic segmentation. Regarding 2D images, a plethora of research has been done in last few years [11, 20, 59, 68, 92], pushing the boundary of state-of-the-art results to the limit. A similar progress has not been in the field of semantic segmentation of 3D pointcloud data due to inherent differences in the two data modalities. In the case of 2D images, the input data to the network is fixed but in the case of 3D data, multiple representations are possible (Section 5.2). Regarding the current task, the most commonly used representation are either a collection of 3D points or projecting the pointcloud on a 2D image. For the first representation, the PointNet architecture proposed by Qi et al. [83] is a popular choice for learning from unordered pointcloud. They propose to use a multi layer perceptron, for learning features from individual points and then use a symmetric function to combine features learned from points, as a *global* representation. A symmetric function is necessary in this case, in order to make the learned representation invariant to the permutations of the input point set. For the task of classification, the learned global representation is sufficient but for segmentation they propose to combine the global representation with learned local features. Extending PointNet, they proposed PointNet++ [84]. The extension include hierarchical learning, where a set of points (centroids) are sampled from the input point set and then points in the neighborhood of the centroids are grouped together, which is then followed by the PointNet architecture. The grouping of

points in the metric space, enable learning of local contextual information. They have shown results primarily on indoor sequence for the data collected from RGB-D sensors. In our case, we use a LiDAR scanner for segmentation of urban outdoor environments. The data from LiDAR scanner is sparser in comparison to the RGB-D sensor and the outdoor environment is more spread out in comparison to confined indoor spaces.

Table 5.6: Results for KITTI tracking benchmark. We report class wise IoU for our HMM based approach proposed in Section 4.4, our method proposed in this chapter (dynamicity score and dynamic log-odds) and a comparison between the descriptors for dynamic log-odds case

Seq. ID	HMM (Section 4.4)		HMM (dynamicity score)			HMM (dynamic log-odds + SHOT)			HMM (dynamic log-odds)		
	Stat.	Dyn.	N. mov.	Mov.	Dyn.	N. mov.	Mov.	Dyn.	N. mov.	Mov.	Dyn.
0	94.7	**66.5**	95.1	65.6	43.6	**95.6**	64.6	52.4	95.3	**66.8**	49.1
2	67.2	25.1	97.5	**56.6**	35.3	**98.2**	49.3	41.6	98.1	54.5	**43.4**
3	89.6	40.4	97.9	0.0	52.7	**99.3**	0.0	73.6	99.0	0.0	**75.0**
4	83.6	25.5	97.4	28.4	38.3	**98.7**	37.0	**57.7**	**98.7**	**42.0**	**57.7**
5	76.2	34.0	97.3	2.1	43.3	**98.6**	**5.5**	60.9	**98.6**	3.2	**62.1**
6	84.0	54.7	98.0	-	62.1	**98.8**	-	74.1	**98.8**	-	**74.8**
7	97.9	23.1	97.3	70.6	28.9	97.4	**71.9**	**55.7**	**97.4**	71.3	38.6
8	64.0	15.0	96.2	**43.9**	24.1	**99.0**	20.1	53.5	**99.0**	22.9	**54.2**
9	84.2	2.3	96.8	**68.9**	5.4	97.8	66.2	16.2	**98.0**	68.5	**18.3**
10	59.6	16.9	95.3	35.3	24.6	99.2	23.6	70.1	**99.2**	**42.1**	**75.9**
11	90.2	42.8	96.0	70.0	47.7	**96.6**	**73.1**	**57.4**	96.5	71.5	55.6
12	98.2	2.3	**98.9**	-	**3.5**	**98.9**	-	2.4	**98.9**	-	0.0
13	97.6	12.9	98.3	23.9	19.0	**98.6**	**25.3**	22.1	**98.6**	**25.3**	**39.6**
14	97.5	33.0	98.3	76.4	45.9	98.4	65.3	33.5	**98.4**	**77.3**	**58.7**
15	94.4	6.7	97.9	76.0	8.7	**98.5**	**80.2**	14.9	98.3	78.1	**20.0**
16	97.9	21.8	98.3	24.9	24.1	98.6	**35.7**	33.1	**98.4**	26.8	**34.8**
18	77.6	46.9	97.5	3.8	51.9	98.2	4.7	**62.3**	**98.1**	**5.2**	62.1
19	98.0	20.4	97.5	30.6	24.5	97.5	28.3	31.6	**97.7**	**31.6**	**35.8**

Table 5.7: Results for KITTI tracking benchmark. We report class wise *precision* for our HMM based approach proposed in Section 4.4, our method proposed in this chapter (dynamicity score and dynamic log-odds) and a comparison between the descriptors for dynamic log-odds case

Seq. ID	HMM (Section 4.4)		HMM (dynamicity score)			HMM (dynamic log-odds + SHOT)			HMM (dynamic log-odds)		
	Stat.	Dyn.	N. mov.	Mov.	Dyn.	N. mov.	Mov.	Dyn.	N. mov.	Mov.	Dyn.
0	99.2	87.9	96.0	88.9	85.8	**96.4**	89.2	87.5	96.1	**89.9**	**93.6**
2	96.5	30.1	**99.3**	**68.7**	43.7	**99.3**	56.9	57.8	99.2	65.7	**58.6**
3	99.8	43.3	99.5	0.0	57.8	**99.6**	1.6	84.0	**99.5**	0.0	**86.2**
4	98.1	30.5	**99.3**	30.7	49.0	99.2	43.4	83.5	99.2	**46.3**	**84.9**
5	99.7	36.9	**99.3**	31.6	47.9	**99.1**	**74.5**	**79.5**	99.0	40.4	82.7
6	99.0	61.2	99.5	-	68.8	**99.5**	-	**85.6**	**99.5**	-	85.3
7	99.2	34.3	**98.5**	84.3	47.0	98.4	**86.9**	**69.8**	98.4	84.9	67.9
8	99.9	16.2	**99.3**	**90.3**	27.0	**99.3**	85.5	73.3	99.2	84.2	**75.9**
9	99.7	2.4	**98.5**	**92.1**	5.8	**98.5**	89.3	21.7	**98.5**	**92.1**	**25.4**
10	100.0	17.3	**99.6**	59.3	25.5	99.5	**70.5**	82.5	**99.6**	67.8	**87.6**
11	97.2	55.2	97.6	86.2	63.1	**97.8**	86.7	**79.3**	97.6	**87.4**	76.8
12	98.6	28.8	**99.2**	-	38.3	**99.2**	-	82.6	**99.2**	-	**100.0**
13	100.0	18.8	**98.8**	54.1	35.2	**98.8**	58.6	81.5	**98.8**	**62.4**	**84.3**
14	99.6	44.8	**99.1**	90.4	60.0	**99.1**	89.5	40.1	**99.1**	**90.6**	**82.9**
15	98.1	14.9	**98.6**	90.0	22.2	**98.8**	90.6	65.6	98.6	**93.1**	**67.1**
16	100.0	58.0	**98.9**	37.5	66.2	**99.0**	**48.2**	92.8	98.9	41.9	**93.2**
18	87.0	68.2	99.0	3.9	78.3	**99.1**	4.7	89.8	98.9	**5.2**	**90.1**
19	98.8	54.1	**98.1**	51.8	63.6	97.9	52.6	**93.3**	**98.1**	**54.9**	92.9

Table 5.8: Results for KITTI tracking benchmark. We report class wise *recall* for our HMM based approach proposed in Section 4.4, our method proposed in this chapter (dynamicity score and dynamic log-odds) and a comparison between the descriptors for dynamic log-odds case

Seq. ID	HMM (Section 4.4)		HMM (dynamicity score)			HMM (dynamic log-odds + SHOT)			HMM (dynamic log-odds)		
	Stat.	Dyn.	N. mov.	Mov.	Dyn.	N. mov.	Mov.	Dyn.	N. mov.	Mov.	Dyn.
0	95.3	**73.2**	99.0	71.4	47.0	**99.1**	70.1	56.6	**99.1**	**72.2**	50.8
2	68.8	60.5	98.1	76.2	**64.7**	**98.8**	**78.7**	59.8	**98.8**	76.2	62.5
3	89.7	85.9	98.3	0.0	**85.7**	**99.4**	1.8	85.5	**99.4**	0.0	85.2
4	84.9	60.6	98.1	79.6	63.7	**99.5**	71.4	**65.1**	**99.5**	**81.9**	64.3
5	76.3	81.2	98.0	2.2	**81.9**	**99.5**	**5.6**	72.2	**99.5**	3.4	71.4
6	84.7	83.8	98.4	-	**86.5**	**99.2**	-	84.7	**99.2**	-	85.8
7	98.6	41.5	98.7	81.3	42.9	**98.9**	80.7	**73.3**	**98.9**	**81.7**	47.1
8	64.0	66.8	96.8	**46.1**	**69.4**	99.6	20.8	66.5	**99.7**	24.0	65.4
9	84.4	43.4	98.2	73.2	**43.5**	99.3	71.9	39.4	**99.4**	**72.8**	39.6
10	59.6	87.4	95.6	46.7	**86.8**	**99.6**	26.1	82.3	**99.6**	**52.6**	85.0
11	92.6	65.6	98.2	78.8	66.2	98.7	82.3	**67.6**	**98.8**	**79.7**	66.9
12	99.5	2.5	**99.7**	-	**3.7**	**99.7**	-	2.5	**99.7**	-	0.0
13	97.6	29.4	99.4	29.9	29.2	**99.7**	**30.8**	23.3	**99.7**	**29.9**	**42.7**
14	97.9	55.7	99.1	83.0	66.0	**99.3**	70.7	**66.9**	**99.3**	**83.9**	66.7
15	96.1	11.0	99.2	83.0	12.6	**99.6**	**87.4**	16.2	**99.6**	82.8	**22.1**
16	97.9	25.8	99.3	42.6	27.5	**99.5**	**57.8**	33.9	**99.5**	42.6	**35.8**
18	87.7	60.1	98.4	93.6	60.7	99.0	93.1	**67.1**	**99.1**	**93.5**	66.7
19	99.2	24.7	99.3	**42.8**	28.5	99.5	38.1	32.3	**99.5**	42.7	**36.8**

In our case, we use the second representation i.e. projecting the 3D LiDAR scan on to a 2D image. As mentioned before, this allows us to represent a LiDAR scan in a compact fashion and furthermore the advancements made in the field of semantic segmentation using 2D images can be used as well.

Focusing on the task of semantic segmentation using 2D images, one of the initial architectures was proposed by Long et al. [68]. They proposed an encoder-decoder style, fully convolutional network (FCN) architecture and other architectures since then have followed the same paradigm. In the presented ablation study, we showed that reducing the spatial resolution of the feature maps size using pooling operations leads to loss of information (Down-sample $8\times$ in Table 5.3) but such operation is necessary for increasing the receptive field (Down-sample width $4\times$ in Table 5.3). Targeting this problem, Chen et al. [20] proposed using Atrous convolution to increase the receptive field without requiring the lossy pooling operations. One of the methods [125] we compare with, uses the Atrous convolution for the same reasons. We compare our proposed architecture, with the architecture proposed in [127], [128] [125]. All of these architecture are based on SqueezeNet architecture proposed in [56]. A detailed explanation of different characteristics of each of these architectures is mentioned in Section 5.2.5.

Some of the non deep learning based methods for semantic segmentation of LiDAR data are [26, 75, 123]. Moosmann et al. [75], proposed a method for segmenting a 3D LiDAR scan based on the notion that objects are locally convex. Wang et al. [123] proposed a method of segmenting a LiDAR scan into the same classes as us. Their method involves decomposing a scan into a set of segments using a graph-based segmentation method. For each such segment, they extract a feature vector, which is then classified using Euclidean minimum spanning tree algorithm. Dohan et al. [26] proposed a hierarchical approach for segmenting a LiDAR scan into multiple classes. Their method involves removing structural elements like roads, curbs etc. and then clustering the remaining points into smaller such segments. Similar to [123], these segments are represented by a handcrafted descriptor. They propose a probabilistic approach to estimate the probability of a segment belonging to a particular semantic class and the probability of two nearby segments having the same class. Since our proposed approach is based on a DCNN, we only report comparison with similar approaches.

Besides, semantic segmentation of a LiDAR scan, in this chapter we also focus on inferring the semantic and motion states together. Methods focusing on the same problem have been proposed but primarily for images [30, 43, 88, 101, 121]. Reddy et al. [88] proposed a dense CRF based method, where they combine semantic, geometric and motion constraints for joint pixel-wise semantic and motion labeling. Similar to them Fan et al. [30] proposed a neural network based method, where they combine motion cues with learned semantic cues. For motion segmentation, they use a stereo camera and leverage over disparity information and combine this

with semantic segmentation mask, using a dense CRF. More recently proposed methods, focus on end-to-end learning. Siam et al. [101] proposed a method of object detection and motion segmentation. Input to their method is a RGB image and a corresponding motion flow image. Features learned from each of these modalities are combined together to predict the bounding box coordinates for the detection and the motion segmentation mask. Similar to [101], Haque et al. [43] also use RGB image and a motion flow image, to jointly infer the semantic and motion states together. Unlike the other methods, which uses a motion flow image as input data, Vertens et al. [121], focus on learning the motion flow and the semantics together and then combining them for motion segmentation.

5.6 Conclusions

In this chapter, we concentrated on learning the semantic knowledge of the 3D environment and combining this knowledge with the motion cues for understanding the environment. We proposed a DCNN to segment points in a 3D LiDAR scan into multiple semantic categories. Our proposed architecture is based on dense blocks and uses depth separable convolution to reduce the parameters while still maintaining competitive performance. It significantly outperforms state-of-the-art neural network architectures, with an average improvement of around 16% across different classes. In the presented ablation study, we justify our architecture choices. The neural network predicts the segmentation mask for each scan independently and to make these predictions temporally consistent, we proposed a Bayes filter method. Through extensive evaluation on the KITTI tracking benchmark, we report a consistent improvement across classes and sequences. These results clearly show the need of such an approach, especially when the input data is sequential, which is rarely not true in the case of robotic perception. Motion and semantic cues entail necessary and complementary information and to leverage this, we propose a HMM based method for combining these cues. This method classifies points in a scan into classes: *non-movable*, *movable* or *dynamic*. In the previous chapter, our method of semantic classification was plagued by misclassification of non-movable structure as dynamic. In this chapter, we overcome this shortcoming and show a significant improvement in precision of classification of dynamic points. Both the Bayes filter and HMM methods, require accurate pointwise motion for different tasks. We show how our learned descriptor from Chapter 3 enables accurate estimation of motion, thereby further improving the classification performance.

Chapter 6

Conclusion

In this thesis, we proposed different novel methods for scene understanding using 3D LiDAR data. Targeting automation of vehicles on roads, we primarily focused on the understanding of urban outdoor environments. For the proposed methods, we describe various theoretical contributions, report qualitative and quantitative results through extensive experimental evaluation and discuss related works.

The environment in which a robot is operating evolves regularly and this evolution is closely related to its motion and semantic characteristics. In the first part of the thesis, we focus on motion characteristics. A key requirement for estimating motion is finding the association between the same structures of the environment. To this end, we proposed a local feature descriptor learned from 3D LiDAR scans. To learn the descriptor, we proposed a 2D image and a 3D voxel based representation for the local surface patches and two DCNN architectures. Using the first architecture, the descriptor and the metric for matching the descriptors are learned simultaneously, and for the latter, the descriptor is learned using the Euclidean metric. For different experiments, we reported superior performance of our learned descriptors in comparison to commonly used handcrafted descriptors and descriptors learned from different architectures. To gain further insight, we also presented an analysis of the discriminative power of various descriptors.

To understand the dynamic characteristics of various objects in the environment, we proposed a method for detection and tracking of dynamic objects, and a method for estimating dense rigid motion field. The first method assumes that an environment is a collection of rigid objects, and we estimate motion for these objects in an iterative fashion using RANSAC. To associate points in a scan to an estimated motion model, we proposed a Bayesian segmentation approach. Assuming the majority of the structure in the environment is static, the first motion model describes the sensor motion and using the subsequent motion models we detect various dynamic objects. For tracking, we use the motion model for data association. This method allows us to detect and track various arbitrary different objects belonging to various semantic categories. We evaluated our approach on a publicly available dataset and substantially outperformed a popular method.

In contrast to the assumption that an environment is a collection of rigid bodies, our proposed novel method for estimating dense rigid motion field relies on

the assumption that objects are locally rigid. To estimate the motion field we used graph based optimization techniques. To build the graph, we used a factor graph representation. Through two different factor nodes, the graph defines the connection between the corresponding keypoints in consecutive scans and the neighboring points in the first scan. To realize these connections, we proposed a method for estimating robust data association between keypoints and a method for estimating neighboring points. We also proposed two factor graph representations. In the first representation, the first factor only uses the measurement from a single pair of associated keypoints. In the latter representation, multiple pairs of corresponding keypoints are used. We evaluate our approach on publicly available real and simulated datasets consisting of urban outdoor environments and a dataset containing pedestrians collected by us. For all the experiments we used ICP algorithm as the baseline method. Through extensive analysis, we showed how our method estimates motion models of various dynamic objects and seamlessly adapts to the case of non-rigid bodies. Among different representations, the second factor graph representation performs better. We also incorporated our learned descriptor and reported results for that.

Having a dense motion field paves the way for scene understanding. To this end, we proposed a HMM based method for inferring the motion state of points in a 3D LiDAR scan. Having such a formulation, allows us to neatly incorporate the motion field and furthermore enable temporally consistent predictions. As a baseline, we use our method of detection and tracking dynamic objects. Even though this method successfully classifies dynamic points but misclassified parts of the static structure as dynamic, resulting in a high false positive rate.

In the second part of the thesis, we shifted our focus from motion to semantic characteristics. We proposed a DCNN architecture for semantic segmentation of LiDAR scan into the classes *car*, *pedestrian* or *bicyclist*. Our architecture is based on dense blocks and uses depth separable convolution for arresting the parameters. We substantially outperform state-of-the-art methods, while maintaining real time performance. To make the neural network predictions temporally consistent, we proposed a Bayes filter method. This method recursively estimates the semantic state of point using the current prediction and the prediction from previous scans. This recursive estimation requires data association between points in a scan, and to this end, we use our method of estimating the motion field for solving this task. Through extensive experiments, we showed how this method helps in mitigating isolated erroneous predictions from the network and consistently improves the performance across different classes.

In the last part of the thesis, we proposed a HMM based method for classifying points in a 3D LiDAR scan into classes *non-movable*, *movable* or *dynamic*. This method neatly combines the learned semantic cues with motion cues and allows temporally consistent predictions. Our HMM based method for estimating pointwise motion

states suffered from a high false positive rate and we showed by incorporating the semantic information this rate can be reduced substantially.

In summary, in this thesis, we proposed various novel methods to address various challenges pertaining to scene understanding. We primarily focused on non-static environmental conditions. We proposed different methods to infer dynamic characteristics of the environment, a method for semantic segmentation and a method to understand the environment by exploiting the complex interplay between motion and semantic cues.

Future Work

In this thesis we used 3D LiDAR as the only data source for scene understanding. However, robots are often equipped with variety of sensors and using multiple sensors can potentially improve the robustness of the system. LiDARs provide precise depth information and in contrast to that RGB cameras provide rich color information. In this thesis, we used DCNNs for learning a local feature descriptor and for the task of semantic segmentation. For both tasks, we used modalities that can be readily extracted from 3D LiDAR data and we think augmenting these modalaties with color information can further improve the performance.

With the advent of deep learning a variety of perception tasks can be solved using learning based methods. In our method of detection and tracking dynamic objects, for tracking we used estimated motion models for association. An interesting extension could be learning a global representation for detected dynamic objects and use the representation for association along with the motion cues. Recently Behl et al. [15] proposed an end-to-end supervised learning method for estimating pointwise motion. This method can be seen as an alternative to our proposed method of estimating pointwise motion. An interesting research avenue would be to solve this problem in a semi-supervised or unsupervised fashion. In our method, the corresponding keypoints provided the cues required for estimating motion. Similarly, the corresponding keypoints could potentially provide (weak) supervision for estimating motion in a learning based method. Having such a formulation would require focusing towards novel loss functions and ingenious architecture design choices. Another possible extension would be to incorporate semantic information and exploit the relation between semantic and motion characteristics by jointly learning representations for these tasks.

We proposed a HMM based method for inferring pointwise motion state and classifying points in a LiDAR scan into different classes. The first method uses estimated motion and the odometry to separate static parts of the environment from dynamic. The main shortcoming of this approach was high false positive rate and we addressed this by incorporating semantic information. An alternative to such an approach would be to jointly learn representations to estimate odometry

and motion together, with the final objective of inferring motion state. The main advantage of such an approach would be that the uncertainty in the estimation of individual quantities are combined implicitly, that can potentially improve the performance. Similarly for the second method, along with odometry and motion models, representations for semantics can be learned together. Such an approach would offer above mentioned advantages as well.

To summarize, a possible extension of different methods presented in this thesis would be learning based methods. The current DCNNs have pushed the boundaries of the state-of-the-art performance for variety of tasks, especially those related to perception. They are capable of learning complex representations to jointly solve related tasks and neatly exploit inter-task dependencies. If representations are learned separately, uncertainty estimation becomes crucial and therefore we think methods such as Bayesian deep learning could be another promising research direction. Another challenge that would have to be addressed is choosing a favorable representation of the 3D data, especially for joint learning since different representations are favored for different tasks.

List of Figures

List of Tables

Bibliography

[1] 2018 SELF-DRIVING SAFETY REPORT. https://www.gm.com/content/dam/company/docs/us/en/gmcom/gmsafetyreport.pdf, 2018. [Online; accessed 28-May-2019].

[2] The path to autonomous driving. https://www.bmw.com/en/automotive-life/autonomous-driving.html, 2019. [Online; accessed 28-May-2019].

[3] Our Approach to Automated Driving System Safety . https://www.apple.com/ads/ADS-Safety.pdf, 2019. [Online; accessed 28-May-2019].

[4] Cars with Autopilot in 2019. https://www.autopilotreview.com/cars-with-autopilot-self-driving/ , 2019. [Online; accessed 28-May-2019].

[5] AUTOMATED DRIVING AT TOYOTA: VISION, STRATEGY AND DEVELOPMENT. http://automatedtoyota.com/, 2019. [Online; accessed 28-May-2019].

[6] Waymo. https://waymo.com/, 2019. [Online; accessed 28-May-2019].

[7] Waymo: Our Mission. https://waymo.com/mission/, 2019. [Online; accessed 28-May-2019].

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[9] L. A. Alexandre. 3d descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, volume 1, page 7, 2012.

[10] A. Azim and O. Aycard. Detection, classification and tracking of moving objects in a 3d environment. In *IEEE Intelligent Vehicles Symposium (IV)*, 2012.

[11] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv: 1511.00561*, 2015. URL http://arxiv.org/abs/1511.00561.

[12] V. Balntas, E. Johns, L. Tang, and K. Mikolajczyk. Pn-net: conjoined triple deep network for learning local image descriptors. *arXiv preprint arXiv:1601.05030*, 2016.

[13] Y. Bar-Shalom and X.-R. Li. *Estimation and tracking- Principles, techniques, and software*. Artech House, 1993.

[14] Y. Bar-Shalom and X.-R. Li. *Multitarget-multisensor tracking: principles and techniques*. Yaakov Bar-Shalom, 1995.

[15] 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.

[16] J. Billington. The history of adaptive cruise control. `https://www.autonomousvehicleinternational.com/features/adas-3.htmls`, 2018. [Online; accessed 28-May-2019].

[17] F. Boniardi, T. Caselitz, R. Kümmerle, and W. Burgard. Robust lidar-based localization in architectural floor plans. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3318–3324. IEEE, 2017.

[18] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert. High accuracy optical flow estimation based on a theory for warping. In *European Conference on Computer Vision (ECCV)*. Springer, 2004.

[19] Y.-T. Chan and F. L. Jardine. Target localization and tracking from doppler-shift measurements. *IEEE Journal of Oceanic Engineering*, 15(3):251–257, 1990.

[20] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[21] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[22] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[23] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. *arXiv preprint arXiv:1611.07759*, 2016.

[24] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[25] L. Cosmo, E. Rodola, A. Albarelli, F. Memoli, and D. Cremers. Consistent partial matching of shape collections via sparse modeling. *Computer Graphics Forum*, 2016. to appear.

[26] D. Dohan, B. Matejek, and T. Funkhouser. Learning hierarchical semantic segmentations of lidar data. In *3D Vision (3DV), 2015 International Conference on*, pages 273–281. IEEE, 2015.

[27] P. S. Dragomir Anguelov, H.-C. Pang, D. Koller, and J. D. Sebastian Thrun. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In *MIT Press Conference on Neural Information Processing Systems (NIPS)*, 2005.

[28] 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. *arXiv preprint arXiv:1609.06666*, 2016.

[29] A. Ess, K. Schindler, B. Leibe, and L. Van Gool. Object detection and tracking for autonomous navigation in dynamic environments. *The International Journal of Robotics Research*, 29(14):1707–1725, 2010.

[30] Q. Fan, Y. Yi, L. Hao, F. Mengyin, and W. Shunting. Semantic motion segmentation for urban dynamic scene understanding. In *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, pages 497–502. IEEE, 2016.

[31] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. Van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[32] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

[33] C. F.Kerry and J. Karsten. Gauging investment in self-driving cars. `https://www.brookings.edu/research/gauging-investment-in-self-driving-cars/`, 2017. [Online; accessed 28-May-2019].

[34] D. Fortun, P. Bouthemy, and C. Kervrann. Optical flow modeling and computation: a survey. *Computer Vision and Image Understanding*, 134:1–21, 2015.

[35] A. Frome, D. Huber, R. Kolluri, T. Bülow, and J. Malik. Recognizing objects in range data using regional point descriptors. *Computer vision-ECCV 2004*, pages 224–237, 2004.

[36] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[37] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[38] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[39] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[40] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 273–278. IEEE, 2010.

[41] D. Haehnel, S. Thrun, and W. Burgard. An extension of the icp algorithm for modeling nonrigid objects with mobile robots. In *IJCAI*, 2003.

[42] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3279–3286, 2015.

[43] N. Haque, N. D. Reddy, and K. M. Krishna. Joint semantic and motion segmentation for dynamic scenes using deep convolutional networks. *arXiv preprint arXiv:1704.08331*, 2017.

[44] S. Hassler. Self-driving cars and trucks are on the move [spectral lines]. *IEEE Spectrum*, 54(1):6–6, 2017.

[45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[46] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[47] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[48] D. Held, J. Levinson, S. Thrun, and S. Savarese. Combining 3d shape, color, and motion for robust anytime tracking. In *Proceedings of Robotics: Science and Systems*, 2014.

[49] E. Herbst, P. Henry, X. Ren, and D. Fox. Toward object discovery and modeling via 3-d scene comparison. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[50] E. Herbst, X. Ren, and D. Fox. Rgb-d flow: Dense 3-d motion estimation using color and depth. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[51] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[52] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[53] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[54] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[55] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[56] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[57] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2462–2470, 2017.

[58] M. Jaimez, M. Souiai, J. Gonzalez-Jimenez, and D. Cremers. A primal-dual framework for real-time dense rgb-d scene flow. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 98–104, 2015.

[59] S. Jégou, M. Drozdzal, D. Vazquez, A. Romero, and Y. Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 1175–1183. IEEE, 2017.

[60] P. Jund, A. Eitel, N. Abdo, and W. Burgard. Optimization beyond the convolution: Generalizing spatial relations with end-to-end metric learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2018.

[61] R. Kaestner, J. Maye, Y. Pilat, and R. Siegwart. Generative object detection and tracking in 3d range data. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[62] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[63] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[64] R. Kümmerle, M. Ruhnke, B. Steder, C. Stachniss, and W. Burgard. Autonomous robot navigation in highly populated pedestrian zones. *Journal of Field Robotics*, 32(4):565–589, 2015.

[65] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. *arXiv preprint arXiv:1608.07916*, 2016.

[66] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[67] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[68] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[69] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza. Dronet: Learning to fly by driving. *IEEE Robotics and Automation Letters*, 2018. doi: 10.1109/lra.2018.2795643.

[70] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[71] Z. C. Marton, R. B. Rusu, and M. Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[72] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[73] M. Menze and A. Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, 2015.

[74] F. Moosmann and C. Stiller. Joint self-localization and tracking of generic objects in 3d range data. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[75] F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion. In *2009 IEEE Intelligent Vehicles Symposium*, pages 215–220. IEEE, 2009.

[76] D. Muoio. Google just made a big move to bring down the cost of self-driving cars. `https://www.businessinsider.de/googles-waymo-reduces-lidar-cost-90-in-effort-to-scale-self-driving-cars-2017-1?r=US&IR=T`, 2017. [Online; accessed 28-May-2019].

[77] T. Naseer, L. Spinello, W. Burgard, and C. Stachniss. Robust visual robot localization across seasons using network flows. In *Proc. of the AAAI Conference on Artificial Intelligence*, 2014.

[78] T. Naseer, G. L. Oliveira, T. Brox, and W. Burgard. Semantics-aware visual localization under challenging perceptual conditions. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2614–2620. IEEE, 2017.

[79] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[80] U. D. of Transportaion. 2016 Fatal Motor Vehicle Crashes: Overview. `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456`, 2017. [Online; accessed 28-May-2019].

[81] A. Petrovskaya and S. Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2-3), 2009.

[82] F. Pomerleau, P. Krusi, F. Colas, P. Furgale, and R. Siegwart. Long-term 3d map maintenance in dynamic environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[83] 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, 2017.

[84] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017.

[85] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[86] J. Quiroga, T. Brox, F. Devernay, and J. Crowley. Dense semi-rigid scene flow estimation from rgbd images. In *European Conference on Computer Vision (ECCV)*. Springer, 2014.

[87] N. Radwan, A. Valada, and W. Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018.

[88] N. D. Reddy, P. Singhal, and K. M. Krishna. Semantic motion segmentation using dense crf formulation. In *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*, page 56. ACM, 2014.

[89] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[90] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[91] G. Riegler, A. Osman Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[92] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[93] P. Ruchti and W. Burgard. Mapping with dynamic-object probabilities calculated from single 3d range scans. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, Brisbane, Australia, May 2018. URL `http://ais.informatik.uni-freiburg.de/publications/papers/ruchti18icra.pdf`.

[94] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[95] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[96] S. Scheidegger, J. Benjaminsson, E. Rosenberg, A. Krishnan, and K. Granström. Mono-camera 3d multi-object tracking using deep learning detections and pmbm filtering. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 433–440. IEEE, 2018.

[97] J. Serafin, E. Olson, and G. Grisetti. Fast and robust 3d feature extraction from sparse point clouds. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4105–4112. IEEE, 2016.

[98] J. Shackleton, B. Van Voorst, and J. A. Hesch. Tracking people with a 360-degree lidar. In *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2010.

[99] S. Sharma, J. A. Ansari, J. K. Murthy, and K. M. Krishna. Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3508–3515. IEEE, 2018.

[100] M. Sheehan, A. Harrison, and P. Newman. Self-calibration for a 3d laser. *The International Journal of Robotics Research*, 31(5):675–687, 2012.

[101] M. Siam, H. Mahgoub, M. Zahran, S. Yogamani, M. Jagersand, and A. El-Sallab. Modnet: Moving object detection network with motion and appearance for autonomous driving. *arXiv preprint arXiv:1709.04821*, 2017.

[102] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 118–126, 2015.

[103] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[104] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[105] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[106] L. Spinello, K. O. Arras, R. Triebel, and R. Siegwart. A layered approach to people detection in 3d range data. In *AAAI Conference on Artificial Intelligence*, 2010.

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

[108] J.-C. Su, M. Gadelha, R. Wang, and S. Maji. A deeper look at 3d shape classifiers. In *European Conference on Computer Vision*, pages 645–661. Springer, 2018.

[109] A. Tevs, M. Bokeloh, M. Wand, A. Schilling, and H.-P. Seidel. Isometric registration of ambiguous and partial data. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1185–1192. Ieee, 2009.

[110] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2005.

[111] G. D. Tipaldi and F. Ramos. Motion clustering and estimation with conditional random fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009.

[112] F. Tombari, S. Salti, and L. Di Stefano. Unique signatures of histograms for local surface description. In *European Conference on Computer Vision (ECCV)*. Springer, 2010.

[113] Y. Tsin and T. Kanade. A correlation-based approach to robust point set registration. In *European conference on computer vision*, pages 558–569. Springer, 2004.

[114] A. Valada, A. Dhall, and W. Burgard. Convoluted mixture of deep experts for robust semantic segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop, State Estimation and Terrain Perception for All Terrain Mobile Robots*, 2016.

[115] A. Valada, G. Oliveira, T. Brox, and W. Burgard. Towards robust semantic segmentation using deep fusion. In *Robotics: Science and Systems (RSS 2016) Workshop, Are the Sceptics Right? Limits and Potentials of Deep Learning in Robotics*, 2016.

[116] A. Valada, R. Mohan, and W. Burgard. Self-supervised model adaptation for multimodal semantic segmentation. *arXiv preprint arXiv:1808.03833*, 2018.

[117] J. Van De Ven, F. Ramos, and G. D. Tipaldi. An integrated probabilistic model for scan-matching, moving object detection and motion estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2010.

[118] L. van der Maaten and G. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[119] V. Vaquero, A. Sanfeliu, and F. Moreno-Noguer. Deep lidar cnn to understand the dynamics of moving vehicles. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–6. IEEE, 2018.

[120] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *IEEE International Conference on Computer Vision(ICCV)*, 1999.

[121] J. Vertens, A. Valada, and W. Burgard. Smsnet: Semantic motion segmentation using deep convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 582–589. IEEE, 2017.

[122] C. Vogel, K. Schindler, and S. Roth. 3d scene flow estimation with a rigid motion prior. In *IEEE International Conference on Computer Vision(ICCV)*, 2011.

[123] D. Z. Wang, I. Posner, and P. Newman. What could move? finding cars, pedestrians and bicyclists in 3d laser data. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4038–4044. IEEE, 2012.

[124] D. Z. Wang, I. Posner, and P. Newman. Model-free detection and tracking of dynamic objects with 2d lidar. *The International Journal of Robotics Research (IJRR)*, 34(7), 2015.

[125] Y. Wang, T. Shi, P. Yun, L. Tai, and M. Liu. Pointseg: Real-time semantic segmentation based on 3d lidar point cloud. *arXiv preprint arXiv:1807.06288*, 2018.

[126] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi. Towards a viable autonomous driving research platform. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 763–770. IEEE, 2013.

[127] B. Wu, A. Wan, X. Yue, and K. Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893. IEEE, 2018.

[128] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. *arXiv preprint arXiv:1809.08495*, 2018.

[129] D. J. Yoon, T. Y. Tang, and T. D. Barfoot. Mapless online detection of dynamic objects in 3d lidar. *arXiv preprint arXiv:1809.06972*, 2018.

[130] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4353–4361, 2015.

[131] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3dmatch: Learning local geometric descriptors from rgb-d reconstructions. In *CVPR*, 2017.

[132] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.