Karlsruhe Institute of Technology

Anomaly Detection in Lidar Data by Combining Supervised and Self-Supervised Methods

Bachelor thesis

Finn Sartoris

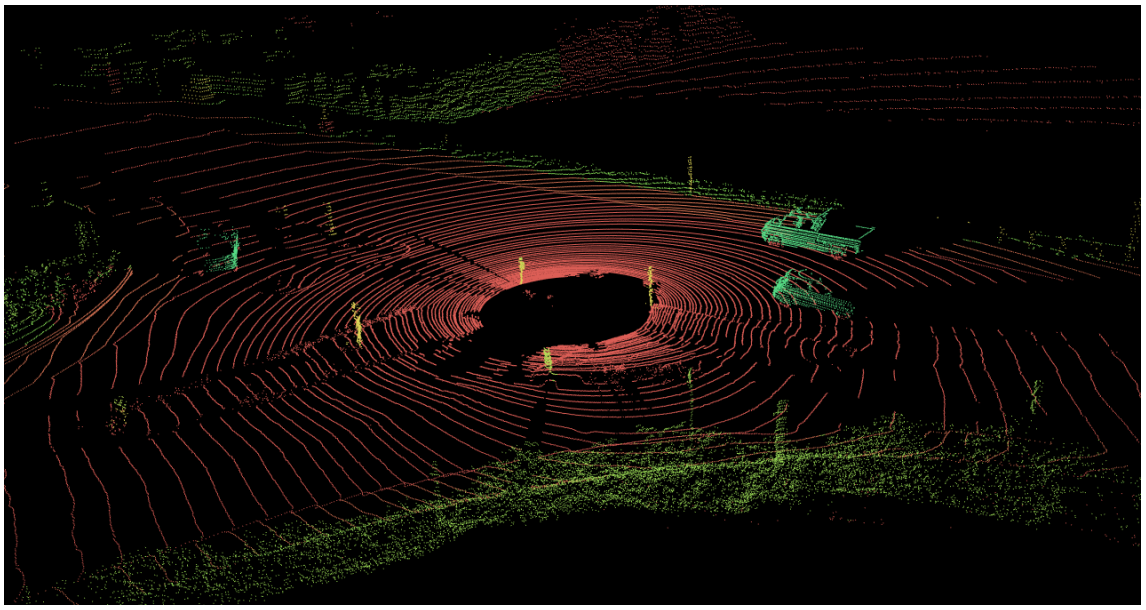
Department of Economics and Management
Institute of Applied Informatics and Formal Description Methods
and
FZI Research Center for Information Technology

Reviewer:	Prof. Dr.–Ing. J. M. Zöllner
Second reviewer:	Prof. Dr.–Ing. A. Oberweis
Advisor:	M.Sc. Daniel Bogdoll

Research Period: 01. December 2021 – 01. June 2022

Anomaly Detection in Lidar Data by Combining Supervised and Self-Supervised Methods

by
Finn Sartoris



Bachelor thesis
June 2022

FZI

Bachelor thesis, FZI
Department of Economics and Management, 2022
Reviewers: Prof. Dr.–Ing. J. M. Zöllner, Prof. Dr.–Ing. A. Oberweis

Department Technical Cognitive Systems
FZI Research Center for Information Technology

Affirmation

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, *Finn Sartoris*
June 2022

Abstract

To enable safe autonomous driving, a reliable and redundant perception of the environment is required. In the context of autonomous vehicles, the perception is mainly based on machine learning models that analyze data from various sensors such as camera, Radio Detection and Ranging (radar), and Light Detection and Ranging (lidar). Since the performance of the models depends significantly on the training data used, it is necessary to ensure perception even in situations that are difficult to analyze and deviate from the training dataset. These situations are called corner cases or anomalies.

Motivated by the need to detect such situations, this thesis presents a new approach for detecting anomalies in lidar data by combining Supervised (SV) and Self-Supervised (SSV) models. In particular, inconsistent point-wise predictions between a SV and a SSV part serve as an indication of anomalies arising from the models used themselves, e.g., due to lack of knowledge. The SV part is composed of a SV semantic segmentation model and a SV moving object segmentation model, which together assign a semantic motion class to each point of the point cloud. Based on the definition of semantic motion classes, a first motion label, denoting whether the point is static or dynamic, is predicted for each point. The SSV part mainly consists of a SSV scene flow model and a SSV odometry model and predicts a second motion label for each point. Thereby, the scene flow model estimates a displacement vector for each point, which, using the odometry information of the odometry model, represents only a point's own induced motion. A separate quantitative analysis of the two parts and a qualitative analysis of the anomaly detection capabilities by combining the two parts are performed. In the qualitative analysis, the frames are classified into four main categories, namely *correctly consistent*, *incorrectly consistent*, *anomalies detected by the SSV part*, and *anomalies detected by the SV part*. In addition, weaknesses were identified in both the SV part and the SSV part.

Contents

Acronyms

lidar Light Detection and Ranging

radar Radio Detection and Ranging

SV Supervised

SSV Self-Supervised

MLUC Metric Learning with Unsupervised Clustering

VAE Variational Autoencoder

KL Kullback–Leibler

VoxCF Voxel-Carries-Flows

LSTM Long Short-Term Memory

DeepSAD Deep Semi-Supervised Anomaly Detection

MLP Multilayer Perceptron

RNN Recurrent Neural Networks

IoU Intersection-over-Union

CNN Convolutional Neural Network

AF2M Attentive Feature Fusion Module

AFSM Adaptive Feature Selection Module

SPVConv Sparse Point-Voxel Convolution

3D-NAS 3D Neural Architecture Search

kNN k-Nearest-Neighbor

MC Monte Carlo

ADF Assumed Density Filtering

FT3D FlyingThings3D

KITTI-SF KITTI Scene Flow

BEV Bird's-Eye-View

GNSS Global Navigation Satellite System

GAN Generative Adversarial Network

ICP Iterative Closest Point

GRU Gated Recurrent Unit

1 Introduction

Critical situations in road traffic occur every day. These situations can be caused by bad weather, a pedestrian suddenly coming from behind an obstacle, or even a lane change. To establish safe autonomous driving, it is necessary to detect those situations before they get dangerous. In the literature, these situations are often referred to as corner cases or anomalies [1]. This thesis, like other works [1, 2], follows the definition for corner cases of Bolte et al. according to which corner cases are given "if there is a non-predictable relevant object/class in relevant location" [3]. This means, only situations that cannot be predicted by the autonomous driving system because the situations deviate too much from the learned ones and where the ego-vehicle is directly involved or will be can be understood as corner cases. To cope with these corner cases, they have to be detected and classified as such. Detected corner cases can serve two main purposes: (1) the online use case, where the identified anomalies are communicated to the autonomous driving system to prevent the situations from becoming critical and (2) the offline use case, where machine learning models can be explicitly trained only on the corner cases found and thus improve the systems' performance in critical situations in general [3].

Perceiving the movement of other road users and thus distinguishing between dynamic and static objects is one of many essential tasks of self-driving cars. Especially in urban areas, where pedestrians, cyclists, and many other road users participate in road traffic, reliable prediction is necessary. For perception, an autonomous vehicle has different sensors to perceive the environment so that the disadvantages of one sensor can be compensated by the advantages of the other one. Cameras provide high-resolution colored images and allow the identification of, for example, road signs and traffic lights, but fail in low light and visibility conditions [4, 5]. radar sensors use high-frequency electromagnetic waves to perceive obstacles and are independent of light and weather conditions. However, the coarse resolutions and the sensibility to target reflectivity are some drawbacks of radar sensors [4, 5]. lidar sensors create a depth representation through 3D point clouds by measuring the time it takes for a laser beam to return after being reflected by an obstacle [6]. The laser beam can be disturbed by different weather conditions, and the detection rate decreases for dark or specular objects. Moreover, the 3D points get sparse for distant objects [6].

Motivated by the need to detect anomalies in road traffic, the goal of this work is to find anomalies in lidar data by testing point-wise motion predictions from SV and SSV models for consistency. Inspired by the use of different sensors for redundant, complementary data in autonomous vehicles, e.g., by Mobileye [7], this work explores the redundancy and complementarity of results from two very different types of models on the same data source. Following the above definition by Bolte et al. [3], in the context of this thesis I refer to an anomaly as an object whose motion

label could not be correctly predicted. Inconsistencies between the supervised and self-supervised models can serve as an indication for an incorrect prediction of one model. For the model that made the incorrect prediction, the motion label of this object is "non-predictable" and can thereby show the limits of the respective models.
In summary, my method can be described as follows: on the one hand, I used a SV semantic segmentation model and a SV motion segmentation model, which assign a class to each point and infer the motion of the point based on the definition of the class. On the other hand, I used a SSV scene flow model and a SSV odometry model to obtain a second motion label for each point. This label is based on the displacement vector predicted by the scene flow model and the ego-motion estimate of the odometry model. Inconsistencies between these two motion labels serve as the basis of my method. To the best of my knowledge, this work is the first to use a combination of SV and SSV models to detect anomalies at the *method level* in lidar data.

In Chapter 2, I provide an overview of existing work in the field of anomaly detection in lidar data, formulate my contribution to the existing research gap, and explain the current state-of-the-art models of components I used. Subsequently, in Chapter 3 I present my method, the individual components, and their interactions with each other. In Chapter 4, I evaluate my method using quantitative and qualitative analysis. The quantitative analysis analyzes the individual components, whereas the qualitative analysis evaluates if anomalies can be found by combining SV and SSV methods.

2 Related Work

This chapter provides the context of the presented method. What types of anomalies exist, and how can they be categorized? What work has been done so far in the field of anomaly detection on lidar data, and what research gap results from this? Subsequently, the contribution of this work to the research gap is shown. The method of this thesis consists of two main types of models. The background of these types of models and recent approaches are given in the last two parts of this chapter.

2.1 Anomaly Detection

In connection with corner cases, the terms “anomaly”, “novelty”, and “outlier” often occur in literature and are closely related. A clear distinction between those terms is not possible, because of overlapping meanings [1] and various applications. An anomaly is a situation where the involved objects do not behave as expected [3] and is partly being used as a synonym for corner cases [1]. A novelty describes any data, as an object, that differs from the already seen data [3]. An outlier refers typically to an extreme value [1]. In this thesis, the terms “corner cases” and “anomalies” are used interchangeably, and thus follow Heidecker et al. [1].
The term corner case is very ambiguous and requires a precise specification to be able to develop suitable detection methods. There are different sensor-specific corner cases, that can be classified in different levels [1, 8, 2]. Heidecker et al. distinguish between *single-source* and *multi-source* corner cases. The former refers to anomalies resulting from a single sensor, while the latter describes anomalies resulting from the fusion of multiple sensors. They categorize the sensor-specific *single-source* corner cases into three layers, namely the *sensor*, *content*, and *temporal layer* and order them based on their detection complexity. The *sensor layer* refers to any corner case that arises from the sensor itself and represents the layer with the lowest detection complexity. Within this layer, Heidecker et al. distinguish between the *hardware level*, which refers to corner cases resulting from a faulty setup, and the *physical level*, which includes any corner case resulting from specific drawbacks of the sensor used. The *content layer* can be subdivided into a *domain level*, *object level*, and *scene level*. The *domain level* refers to anomalies resulting from a domain shift, whereas the *object level* describes anomalies that are associated with a particular object. *Scene-level* anomalies address the context of a frame and refer to the behavior within that scene. *Scenario-level* anomalies, on the other hand, refer to multiple scenes and the behavior over time. Therefore, *scenario-level* anomalies are represented separately at the *temporal layer*. An overview of the categorization by Heidecker et al. including specific examples for the different levels can be found in Figure 2.1. In addition to the sensor-specific corner cases, Heidecker et al. introduce a *method level*, which refers to corner cases resulting from the applied method, e.g., due to a lack

of knowledge. According to [1], a clear distinction between a *method level* anomaly and a *single-source* anomaly is difficult, since they can occur together.

	Sensor Layer		Content Layer			Temporal Layer
	Hardware Level	**Physical Level**	**Domain Level**	**Object Level**	**Scene Level**	**Scenario Level**
LiDAR-based corner cases	Laser Error • *Broken mirror* • *Misaligned actuator*	Beam-Based Corner Case • *Black cars disappear* • ...	Domain Shift on Single Point Cloud • *Shape of Road markings*	Single-Point Anomaly on Single Point Cloud • *Dust cloud* • ...	Contextual/Collective Anomaly on Single Point Cloud • *Sweeper cleaning the sidewalk*	Corner Cases on Multiple Point Clouds and Frames • *Person breaks traffic rule* • *Overtaking a cyclist* • *Car accident* • ...
Camera-based corner cases	Pixel Error • *Dead pixel* • *Broken lense*	Pixel-Based Corner Case • *Dirt on lense* • *Overexposure*	Domain Shift on Single Frame • *Location (EU-U.S.A.)* • ...	Single-Point Anomaly on Single Frame • *Animal* • ...	Contextual/Collective Anomaly on Single Frame • *People on a billboard* • ...	
RADAR-based corner cases	Impulse Error • *Low voltage* • *Low temperature*	Impulse-Based Corner Case • *Interference* • ...	Domain Shift on Single Point Cloud • *Weather, e.g., snow, rain, etc.*	Single-Point Anomaly on Single Point Cloud • *Lost objects* • ...	Contextual/Collective Anomaly on Single Point Cloud • *Demonstration* • *Tree on street*	

Figure 2.1: Categorization of single-source corner cases based on used sensor [1]

In recent years, a lot of research has been published in the field of anomaly detection, especially regarding images [2, 9, 10, 11, 12, 13]. Research on anomaly detection in radar and lidar data is not as advanced and still leaves many gaps [13]. Since an anomaly detection approach on lidar data is presented in this thesis, only related methods based on lidar sensors will be described in the following. Methods for anomaly detection at the *sensor layer*, like in [14], are also not considered, since the focus of this work lies on the detection of external corner cases.

Classical deep learning object detection methods are based on the closed-set assumption. This means that at test time, the models can only output classes on which they have been trained. As a result, unknown objects are falsely classified as known during testing. The closed-set assumption is not viable under real conditions, since the entirety of all possible objects cannot be included in the training. Unknown objects can thus always appear during testing. Wong et al. [15] and Cen et al. [16] tackled this problem by publishing an open-set 3D object detector which can classify both known and unknown classes/objects in lidar data. Since the work of Cen et al. [16] achieves state-of-the-art performance and outperforms the approach of Wong et al. [15], only their method is explained in more detail. The presented method is called Metric Learning with Unsupervised Clustering (MLUC). At first, it finds regions with unknown objects through metric learning. Then it refines the bounding boxes with an unsupervised clustering algorithm. The method uses Euclidean distance-based probability to define the loss so that the embeddings of unknown objects are located in the center of the embedding space. This is because the unknown objects do not resemble any of the predefined prototypes of the known objects. To distinguish between known and unknown objects, the Euclidean distance sum is calculated. The sum should be smaller for unknown objects in the center of the embedding space than that of known objects. If a defined threshold is not reached, the boxes are classified as unknown. Subsequently, the bounding boxes of the regions containing possible unknown objects are refined with an unsupervised clustering algorithm.

Motivated by anomaly detection methods for images, Masuda et al. [17] applies a reconstruction-based method to find anomalies in lidar data. The general idea behind using a Variational Autoencoder (VAE) for anomaly detection is that the difference between the input and the reconstructed

output is relatively small for normal data during training and relatively large for data that deviates from the normal data during testing. Masuda et al. [17] use the Chamfer distance and the Kullback–Leibler (KL) divergence as metrics for the training loss to compare the input and the reconstructed point cloud. The network does not work with whole lidar scans, but with single point clouds of objects from the ShapeNet dataset [18].
Iqbal et al. [19] published a method to detect abnormal motion in point clouds by estimating scene flows and learning motion features. The term abnormal motion refers to, for example, a pedestrian crossing the road in front of a car. The points with the estimated scene flows are clustered based on distance metrics. Subsequently, the closest object with respect to the autonomous vehicle is selected and converted into a 3D grid structure, called Voxel-Carries-Flows (VoxCF). This VoxCF is used to extract dynamic features which describe the direction of an object's motion. The difference between the dynamic features and the prediction by a Long Short-Term Memory (LSTM) network serves as an abnormality measurement. The method can detect abnormal motion with an accuracy of 77.11%.
As already mentioned in Chapter 1, the laser beam of a lidar scanner can be disturbed by different weather conditions. In the case of rain, there are three interactions that cause degradation of laser scans: absorption, reflection, and refraction [20, 21, 22]. Zhang et al. [23] address the problem of lidar degradation quantification in rainy weather by using an anomaly detection model. The lidar point clouds are transformed into a 2D image. The total number of pixels is the same as the total number of laser beams per scan emitted from the sensor. Consequently, a change in the laser beam intensity value due to degradation can be assigned to exactly one pixel. Deep Semi-Supervised Anomaly Detection (DeepSAD) [24] is used as the anomaly detection model and learns a hypersphere by transforming the input images into a latent space. Non-degraded images are mapped into this hypersphere and degraded images are mapped away. During testing, the distance between the hypersphere center and the mapped image serves as a degradation score.
At the moment, there are not many datasets specifically designed for anomaly detection in lidar data. In contrast, there are many more datasets for anomaly detection in the image domain [25, 26, 27, 28, 29, 30]. To the best of my knowledge [31, 32], only one dataset designed for anomaly detection in lidar data exists, namely CODA [33]. The entire CODA dataset was published just before this work was submitted. Therefore, we could no longer consider it during evaluation.

2.2 Research Gap

As stated in [13], all methods described in the previous chapter, except the one by Iqbal et al. [19], detect anomalies on the *domain* or *object level*. The open-set approaches [15, 16] can classify unknown objects and are able to detect anomalies on the *object level*. Similarly, the reconstruction based method by Masuda et al. [17] detects anomalous objects. The domain shift between normal and rainy weather is addressed by Zhang et al. [23]. Iqbal et al. [19] are working on the *scenario level* by finding abnormal motions over time. No work on anomaly detection on lidar data that deals with the *scene level*, could be found.
The existing anomaly detectors on the *object level* are working either with reconstruction-based

methods [17] or with an embedding space [15, 16]. The reconstruction-based method by Masuda et al. [17] is not applicable for autonomous driving, since it does not work on entire point clouds of a scan. The approach by Iqbal et al. [19] only considers the closest object in relation to the autonomous vehicle to detect abnormal movements. No other objects are considered, which is a big disadvantage. Compared to anomaly detection on camera data, the lidar area is still far behind and needs further research so that corner case situations can be detected and safe driving in such situations can be guaranteed.
The approach in this thesis is not based on reconstructions [17], nor does it work with the embedding space [15, 16] and thus represents a new approach to anomaly detection in lidar data. In particular, this thesis presents an approach to anomaly detection at the *method level* by identifying limitations of applied methods. This is done by comparing point-wise motion labels between SV and SSV methods and finding inconsistencies. In contrast to the method of Iqbal et al. [19] I am not interested in abnormal motion, but in the motion of each point itself to decide whether a point is dynamic or static.
The terms “SV” and “SSV” define the way in which a model is trained. During SV training, the network learns the mapping between a given pair of input and output data. The supervision is obtained by the previously labeled data, through comparison with the prediction of the model. A loss value is calculated based on the found discrepancies. Providing this manually labeled data can be very labor-intensive. With SSV training, no labeled data is needed, since the supervision is taken over by the model itself. SSV training is a part of unsupervised training, however, it is used for similar tasks like SV learning methods. Since no labels are needed, SSV learning requires a completely different way of training. To obtain their own supervisory signals, SSV methods use the structure of the underlying data. The potential of SSV models is very high, considering that it is easy to collect tons of data compared to manual labeling. Tesla alone has a team of 1000 people taking care of manual labeling [34].
The potential of using the two different training concepts for anomaly detection in lidar data is demonstrated in this thesis using variously SV and SSV models. The approach of this work can be divided into a SV and a SSV part. In the SV part, semantic motion labels are predicted for each point of a point cloud by combining a semantic segmentation model and a motion segmentation model. In the SSV part, a scene flow model and an odometry model are used to predict a motion label for each point of a point cloud. Thus, two labels are predicted for each point. Inconsistencies between the two labels per point serve as the basis for the anomaly detection. The three main tasks, namely SV semantic segmentation, SSV scene flow, and SSV odometry, are described in the following chapters.

2.3 Supervised Semantic Segmentation on Lidar

In order for an autonomous vehicle to understand the environment based on the generated data, infrastructure and traffic objects must be identified. One way to do this is to perform semantic segmentation. Semantic segmentation assigns a class label, such as car, pedestrian, road, etc., to each pixel in images or to each point in point clouds. In recent years, many deep learning ap-

proaches have been published that learn semantic segmentation on images or point clouds [35, 36, 37, 38, 39]. Since lidar "point clouds are relatively sparse, unstructured, and have non-uniform sampling" [38], four different approaches have evolved to perform semantic segmentation on lidar data, namely projection-based, discretization-based, point-based and hybrid methods [40]. Projection-based methods first project the point cloud into a temporary 2D image, such as a multi-view or spherical images. Discretization-based methods project a point cloud into a discrete representation, which can either be dense or sparse, like a volumetric or permutohedral lattice. At the end of these methods, the temporary representation with the predicted labels is projected back to a raw point cloud. Projection-based methods allow the use of 2D convolutions of the well-developed field of image based computer vision [41], but lose and alter the 3D geometric structure [42] which leads to information loss [40]. The sparse representation of discretization-based methods comes with information loss due to low resolution, while the dense representation strongly depends on memory capacity and computational power [40, 43]. Point-based methods work directly on the raw point clouds, without any prior processing steps. Within point-based methods, a distinction can be made between point-based Multilayer Perceptron (MLP) methods, point-based convolutional methods, Recurrent Neural Networks (RNN) based methods, and graph-based methods [40]. The bottleneck of point-based methods is that they partially waste up to 80% of their runtime for structuring the unstructured point clouds [44]. Hybrid methods combine voxel-based, projection-based and/or point-based approaches [39]. A spherical projection of a 3D point cloud with associated semantic labels onto a 2D image can be seen in Appendix D.1. A visualized labeled raw point cloud is shown in Appendix D.2. Each color represents a different class.

In order to evaluate and compare the performance of semantic segmentation models, the *mean* $Intersection-over-Union$(IoU) metric, also called the mean Jaccard Index, is used by almost all publications [38, 39, 45]:

$$meanIoU = \frac{1}{C}\sum_{c=1}^{C}\frac{TP_c}{TP_c + FP_c + FN_c} \qquad [2.1]$$

C is the number of classes and TP_c, FP_c, and FN_c refers to the number of true positive, false positive and false negative points per class c, respectively [45].

There are several datasets with annotated semantic labels for point clouds [45, 46, 47, 48]. One of the most popular is the SemanticKITTI dataset [45], where all sequences of the KITTI Vision Odometry Benchmark [49] are annotated.

In the following, some of the latest approaches are described. As of May 26, 2022, the model $(AF)^2$-*S3Net* [39] achieves the state-of-the-art performance on the SemanticKITTI dataset. This model is an end-to-end encoder-decoder 3D sparse Convolutional Neural Network (CNN), which especially considers fine details of smaller objects. This is achieved by introducing two attention blocks, namely Attentive Feature Fusion Module (AF2M) and Adaptive Feature Selection Module (AFSM). The former brings local and global features together by combining mutually exclusive learnable masks in a weighted manner. They cover either small, medium, or large instances. The latter learns the relationship between the feature maps of the AF2M and improves the general-

izability. Tang et al. [43] argue that point-based methods are wasting most of their time sorting the unstructured point clouds and voxel-based methods suffer from information loss due to low resolution, which leads to small instances being poorly represented. Because of that, they introduced a novel lightweight 3D module, called Sparse Point-Voxel Convolution (SPVConv), capable of recognizing small objects. In addition, they presented 3D Neural Architecture Search (3D-NAS), an architecture search framework, which automatically finds the best network design, e.g., channel numbers or network depth, for a predefined search space. The resulting model, named *SPVNAS*, is lightweight, accurate, and fast. Zhu et al. [42] found out, that the usual division of point clouds by 3D voxelization methods using uniform cubes leads to an imbalanced distribution. This has to do with the fact that nearby areas have a significantly higher density of points than more distant areas. A cylindrical partitioning results in a more balanced representation in relation to the varying density. Combining cylindrical partition and asymmetrical 3D convolution networks, Zhu et al. [42] developed the *Cylinder3D* model.

Reasoning about the prediction uncertainty of models has gained attention in recent years [9, 50, 51, 52]. The model uncertainty, also called epistemic uncertainty, "describes the uncertainty experienced by a model when it has a lack of knowledge, such as when it encounters conditions not represented by the training data" [52]. The data uncertainty, called aleatoric uncertainty, "refers to uncertainty arising from noise or randomness present in the data" [52]. Cortinhal et al. [38] were the first to develop an uncertainty-aware semantic segmentation model for lidar point clouds. The proposed model, named *SalsaNext*, projects the point cloud onto a spherical surface in order to get a 2D range view image as input. This allows the use of standard 2D convolutions. In order to measure the epistemic uncertainty, Monte Carlo (MC) sampling is used during inference. Dropout layers serve as regulators and are originally used to prevent overfitting during training by randomly dropping neurons [53]. MC dropout, applied during testing, "introduces stochasticity into the model, and repeated testing of an input yields variations in predictions that represent epistemic uncertainty" [52]. However, due to the repeated forwarding of the same input, the MC dropout method is rather slow. To be able to measure the aleatoric uncertainty, Assumed Density Filtering (ADF), introduced by [54] is applied. Here, the activation functions, the input, and the outputs are replaced by probability distributions, so that the network's output consists of a prediction μ and σ_A as the associated aleatoric uncertainty [38, 54]. Instead of using spherical projection to trans-

Method	Mean IoU↑	Code
$(AF)^2$-S3Net [39]	69.7	✗
SPVNAS [43]	66.4	✓
Cylinder3D [42]	67.8	✓
SalsaNext [38]	59.5	✓
KPRNet [41]	63.1	✓

Table 2.1: Comparison of SV semantic segmentation models evaluated on SemanticKITTI test set [45]. A ↑ shows that a higher value is better, and the last column indicates whether the code has been published. All results are taken from the respective research paper.

form the point clouds into a 2D range image, *KPRNet* [41] applies the scan unfolding method, described in [55], which leads to a smoother projection and prevents systematic occlusion. In order to project the 2D image back to a 3D point cloud, Kochanov et al. [41], replaced the typical k-Nearest-Neighbor (kNN) based technique with a single KPConv layer, introduced by [56]. A compact overview of the performance of the described methods can be found in Table 2.1, where the last column indicates whether the code has been published.

2.4 Self-Supervised Scene Flow on Lidar

For self-driving cars, it is essential to recognize and anticipate the movement of other road users in order to understand the scene and react appropriately. In 3D lidar data, scene flow addresses this task by predicting the movement for each point between consecutive time frames in the form of a displacement vector [57]. The counterpart in 2D is called optical flow, where the pixel motion of adjacent frames is predicted [58].
Formally, scene flow can be described as follows: two point clouds X_t and Y_{t+1} consist of a certain number of points $p_i \in \mathbb{R}^3$ and $q_j \in \mathbb{R}^3$, such that each point cloud can be described as $X_t = \{p_1, ..., p_{n_1}\}$ and $Y_{t+1} = \{q_1, ..., q_{n_2}\}$, where n_1 and n_2 refer to the number of points for point cloud X_t and Y_{t+1}, respectively. The goal of scene flow is to estimate a vector $f_i \in \mathbb{R}^3$ for each point $p_i \in X_t$ to transfer the point p_i from timeframe t to its corresponding position p_i' at timeframe $t+1$, so that $p_i' = p_i + f_i$. It needs to be considered that the number of points in X_t does not have to be equal to the number of points in Y_{t+1}. Additionally, p_i' may not be existent in Y_{t+1}, due to sparsity of the point cloud and occlusion [59, 60]. For a better understanding, an example of 3D scene flow is given in Figure 2.2.

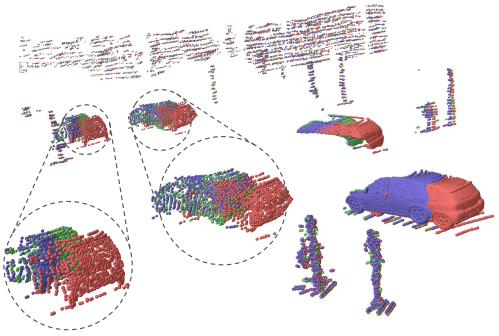


Figure 2.2: Scene flow between two consecutive point clouds. Point cloud X is shown in red, point cloud Y in green and the point cloud X predicted with the scene flow is shown in blue [61]

In recent years, a lot of research has been done in the field of scene flow, especially with SSV approaches [57, 59, 60, 62, 63, 64]. The first work on SSV scene flow has been published in 2019 by Wu et al. [62], where the self-supervision is obtained through the Chamfer distance, smoothness constraints and Laplacian regularization. Following in 2020, Mittal et al. introduced the nearest neighbor loss in combination with an anchored cycle consistency loss in order to compensate for missing ground truth flow [59]. Even though, state-of-the-art SSV scene flow models have achieved the accuracy of early SV models, there is still an accuracy gap between these two

approaches [65]. An additional challenge is a possibility to process raw point clouds, which are directly generated by a lidar scanner without prior down-sampling [65]. Almost all networks require down-sampling from about 120,000 points to a maximum of 8,192 points [57]. Training with more than 8,192 points leads to a significant increase in training time and memory requirements for most methods. Inference on more than the proposed number of points is possible but leads to a domain shift, which is reflected in a reduced performance [57]. Additionally, some methods, like [62], rely on the assumptions of a bijective mapping between point clouds, i.e., every point in point cloud X has a corresponding point in point cloud Y [57, 66]. This assumption cannot be used in real-world applications due to, for example, occlusions [66].
Because of the lack of publicly available annotated scene flow datasets, almost all methods are trained and evaluated based on the same two datasets, the synthetic FlyingThings3D (FT3D) dataset [67] and the real-world KITTI Scene Flow (KITTI-SF) dataset [68, 69]. FT3D consists of stereo and RGB-D images with randomly moving 3D models from ShapeNet [18], ground truth disparity, and optical flow maps [61]. A common pre-processing step is to create point clouds from the disparity maps and the associated scene flow from the optical ground truth flow [60, 61, 62, 70]. The KITTI-SF dataset is a set of 400 dynamic scenes from the KITTI raw data annotated with precise 3D CAD models for all moving vehicles [68, 69]. A very usual pre-processing step is to remove the ground in the point clouds and to only use the points that do not exceed a distance of 35 meters [57, 59, 60, 62, 71]. This is due to the fact that "the ground is a large flat piece of geometry with little cue to tell its motion" [61].
In order to evaluate and compare the performance of scene flow models, common metrics are used by almost all publications [59, 60, 64, 70]:

- **EPE3D** (m): average end-point-error $\|F_{pred} - F_{gt}\|_2$ with the flow $F = \{f_1, ..., f_{n_1}\}$ once from the prediction ($pred$) and once from the ground truth (gt)
- **Acc3DS**: strict version of accuracy, the percentage of points whose **EPE3D** $< 0.05m$ or relative error $< 5\%$
- **Acc3DR**: relaxed version of accuracy, the percentage of points whose **EPE3D** $< 0.1m$ or relative error $< 10\%$
- **Ouliers3D**: percentage of points whose **EPE3D** $< 0.3m$ or relative error $> 10\%$

The model SLIM by Baur et al. [57] currently appears to be the state-of-the-art among the SSV scene flow approaches with regard to the metrics just mentioned. SLIM [57] performs scene flow and motion segmentation together in an end-to-end system. Motion segmentation refers to the task of classifying each point as either "moving" or "stationary". The model is characterized by the fact that it can be trained and evaluated on roughly 64,000 points, without drastically increasing computation time and memory requirements. The inputs of the network are two point clouds with removed ground points encoded into a Bird's-Eye-View (BEV) pseudo-image which is limited to $70 * 70$ meters. Tishchenko et al. [63] decomposed the scene flow into a non-rigid part, which represents the motion caused by a moving object, and an ego-motion part, which refers to the motion

caused by an observer. As described by [57] and as can be seen in Table 2.2, the model suffers from high outlier rates. FlowStep3D [60] has a recurrent architecture, where the initial flow is

Method	Dataset	#points	EPE3D↓	Acc3DS↑	Acc3DR↑	Outliers3D↓	Code
SLIM [57]	KITTI-SF	8192 64000	0.1207 0.0668	0.5178 0.7695	0.7956 0.9342	0.4024 0.2488	✓
Ego-motion & Non-rigid flow [63]	FT3D KITTI-SF	8192	0.1696 0.4154	0.2532 0.2209	0.5501 0.3721	0.8046 0.8096	✓
FlowStep3D [60]	FT3D KITTI-SF	8192	0.0852 0.1021	0.5363 0.7080	0.8262 0.8394	0.4198 0.2456	✓
Occlusion-guided [64]	FT3D KITTI-SF	8192	0.3373 0.2091	0.1232 0.2107	0.3593 0.4904	0.9104 0.7241	✓
SFGAN [71]	KITTI-SF	2048	0.098	0.3022	0.6823	0.5584	✗

Table 2.2: Evaluation comparison of SSV scene flow models with common evaluation metrics, where ↑ shows that a higher value is better, and ↓ indicates that a lower value is better. The column "#points" specifies the number of points per point cloud during inference.

refined in each iteration by an iterative refinement process. The approach of Ouyang et al. [64] focuses on the prediction of scene flow for point clouds with occlusions. "Since the occluded regions usually produce misleading information for the flow estimation" [64], they calculate an occlusion probability and use an occlusion-weighted cost volume. The authors of [71] used a Generative Adversarial Network (GAN) consisting of a generator and a discriminator. The generator generates scene flow with which fake point clouds are created. The discriminator distinguishes between the fake and the real point cloud. Adversarial training brings the fake and real point cloud closer and closer together and increases the accuracy of the scene flow prediction.
For a compact overview of the described SSV scene flow models, see Table 2.2. The column "Dataset" refers to the data used for evaluation, and the last column indicates whether the code has been published.

2.5 Self-Supervised Odometry on Lidar

In addition to detecting and predicting the movement of other road users, it is essential for an autonomous vehicle to know about its own movement and pose. Especially in situations where external signals, like the Global Navigation Satellite System (GNSS), are too inaccurate or not available at all, it is necessary to determine the ego-motion and pose in a different way [72]. For this purpose, different self-contained odometry methods were established which can be categorized based on the sensor used into wheel, inertial, laser, radar, and visual odometry [73]. Since this work deals with lidar data, only the field of laser odometry will be discussed in more detail.
Given two consecutive lidar point clouds X_t and Y_{t+1}, the task of an odometry model is to estimate a 4-by-4 matrix to transform point cloud Y_{t+1} into the frame of point cloud X_t by estimating relative changes in ego-motion. The transformation matrix $Tr_t^{t+1} : Y_{t+1} \rightarrow X_t$ consists of a rotation matrix R and a translation vector t [72], as shown in Equation 2.2.

$$\mathrm{Tr} = \left[\begin{array}{ccc|c} & \mathbf{R} & & \mathbf{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \quad [2.2]$$

Learning odometry from lidar data in an unsupervised or SSV manner is a very new field, pioneered in 2019 by Cho et al. with their model DeepLO [74]. The proposed network takes projected point clouds in the form of 2D vertices and normal maps as input and extracts features from the maps through two separate feature extractors, called *VertexNet* and *NormalNet*. The subsequent *PoseNet* combines the features of each frame and estimates the relative motion between the two frames. For unsupervised training Iterative Closest Point (ICP) and a field of view loss is integrated. The field of view loss stabilizes the training process and serves for regularization.
Similarly, the DeLORA model [75] and the SSLO model [76] both use spherical projection to create range images as input to their network. One difference between the DeLORA and SSLO model to the DeepLO model is that the correspondence search during training is not performed in 2D on the basis of pixels, but in 3D by applying a KD-Tree. Thereby, the nearest neighbor of the source, transformed with the estimated matrix, and the target is searched. The pairs serve as input for a combination of geometric losses. The SSLO model weights larger distances between corresponding points low to give dynamic points and outliers less influence. This incentivizes the model to consider the static points more.
In contrast to the projection-based models, the SelfVoxeLO model [77] works with 3D convolutions for feature extraction. SelfVoxeLO operates on voxelized point clouds instead of the upstream 2D projection, and thus preserves the geometric information in 3D space. Xu et al. introduced a two-stage odometry estimation network with their model RSLO [78]. In the first stage, high dimensional features are encoded, which serve as a separation into so-called subregions. For each subregion, the rigid transformation is estimated. In the second stage, the estimated rigid transformations are used to vote for the ego-motion. This allows the network to focus more on the regions that are relevant and mitigates the influence of dynamic points.
An overview of the performance of the described SSV odometry models is given in Table 2.3.

Method	Seq 09		Seq 10		Code
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	
DeepLO [74]	4.87	1.95	5.02	1.83	✗
DeLORA [75]	6.05	2.15	6.44	3.00	✓
SSLO [76]	2.00	0.88	2.27	0.92	✗
SelfVoxeLO [77]	3.01	1.14	3.48	1.11	✗
RSLO [78]	2.75	1.01	3.08	1.23	✓

Table 2.3: Overview of SSV odometry models evaluated on sequences 9 and 10 of the KITTI odometry dataset [49]. t_{rel} and r_{rel} refer to the translational and rotational errors for all possible subsequences between 100 m and 800 m.

3 Method

In this chapter, the approach of this thesis is presented. First, a general overview is given and the purpose of each component and its interaction with other components are explained. Then, the selected models and their implementations are described in more detail. In the last part of this chapter, the datasets used for training are presented, followed by the explanation of the training process of particular models.

3.1 Method Overview

The method of this thesis works on lidar data and combines variously SV and SSV models. The method consists of five models, of which three are SV and two are SSV. The goal of this method is to find inconsistencies between the semantic class for a point, predicted on the basis of SV models, and the motion label of the same point, predicted on the basis of SSV models. A schematic overview of the approach with its individual components is shown in Figure 3.1.

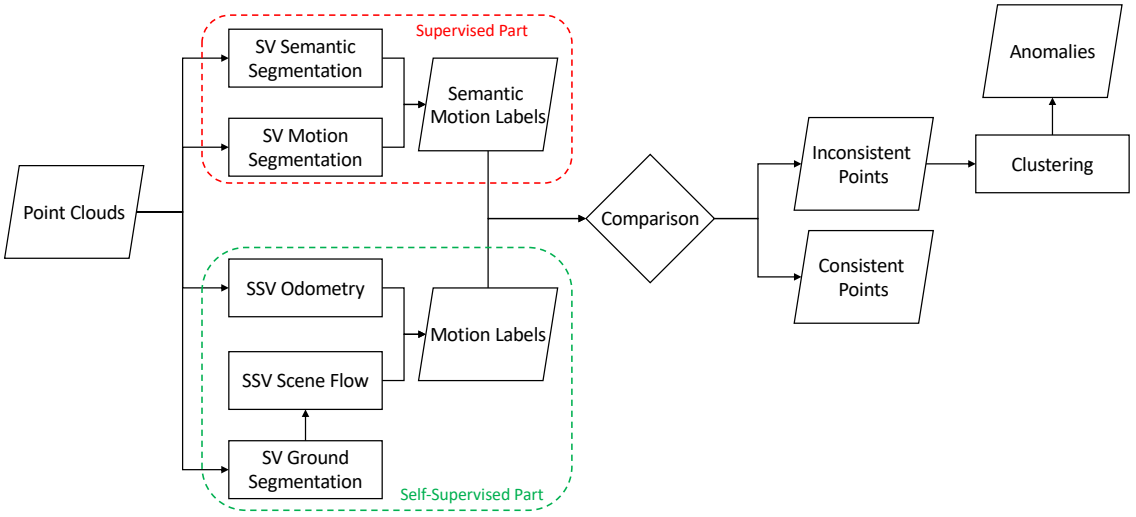


Figure 3.1: Overview of the approach. In the SV part, a semantic motion class is assigned to each point by combining a SV semantic segmentation model and a SV motion object segmentation model. By definition, the class specifies the motion state of the point. In the SSV part, a displacement vector is predicted for each non-ground point by a SSV scene flow model. By combining with the ego-motion predicted by a SSV odometry model, another motion label can be predicted for each point. By comparing the two labels per point, a distinction is made between consistent and inconsistent points. The inconsistent points are clustered and these clusters serve as an indication for anomalies.

The SV part, marked red in Figure 3.1, contains a SV semantic segmentation model and a SV motion segmentation model. The SV semantic segmentation model assigns a class label to each point. In certain cases, it is not possible to draw conclusions about the status of the point's move-

ment on the basis of the defined semantic classes, since some classes can occur in both static and dynamic states. For example, points of the class *person* can be both static and dynamic. In the case of a pedestrian crossing the street, the points of a waiting person are initially static. Once an opportunity presents itself and the person crosses the street, the points are dynamic. A point assigned to the class *building*, on the other hand, is static by definition. For semantic classes to be used to infer the movement of points within that class, the potentially moving classes must be further subdivided. For this purpose, a SV motion segmentation model is used, which classifies each point in the point cloud as either static or dynamic. With these motion labels, the class *person*, for example, can be subdivided into *moving person* and *standing person*. Semantic classes that are static by definition remain unchanged. In Figure 3.1 I refer to the unchanged and newly created classes as *semantic motion classes*.

The SSV part, marked green in Figure 3.1, contains three models, namely a SV ground segmentation, SSV scene flow, and a SSV odometry model. As already mentioned in Chapter 2.4, a common pre-processing step of scene flow models is to remove the points that belong to the ground. The majority of SSV scene flow models remove all points that fall below a certain threshold [57, 60, 63]. The threshold is not learned, but is set once, usually to 0.3 m, and applied in a general way to all point clouds. This removal by height has the consequence that not only ground points are removed, but also points from objects that are close to the ground, e.g., car tires. Even points from small objects, like a soccer ball, could be removed this way. In addition, uneven roads may result in either incomplete removed ground points or removal of non-ground points. Therefore, I decided to use a model that learns to classify each point as *ground* or *non-ground*, in order to exclude the classified ground points.
Two consecutive point clouds with removed ground points serve as an input for the SSV scene flow model, which returns a 3D displacement vector for each point in the first point cloud. By the use of these vectors, a point cloud at time t can be transformed into time $t+1$ and represent the movement of individual points. To distinguish between static and dynamic points from data recorded by a moving vehicle, it is necessary to distinguish between the ego-motion of the observed vehicle and the movement by the point itself. The estimated vectors from the scene flow model do not distinguish between the ego-motion of the observed vehicle and the motion induced by the point itself. In order to differentiate between these two motions, a SSV odometry model is applied. This model estimates a relative rigid body transformation, which allows to reason about the ego-motion between two consecutive point clouds. With this transformation, a point cloud Y_{t+1} can be transformed into the frame of point cloud X_t by compensating the ego-motion. Together with the estimated scene flow, the points can be divided into static or dynamic, by taking a point $p_i \in \mathbb{R}^3$ of point cloud $X_t = \{p_1, ...p_{n_1}\}$ and apply the estimated flow $f_i \in \mathbb{R}^3$ resulting in a point $p_i' = p_i + f_i$ at time $t+1$. By using the rigid body transformation Tr_t^{t+1}, the resulting point p_i'' adjusted for the ego-motion is in the same frame as the initial point p_i, only shifted by the own induced motion of the point. If the length of the flow vector, compensated by the ego-motion, exceeds a certain threshold, the point is considered dynamic, otherwise as static.

The labels of the two parts are compared point-wise. Inconsistencies occur when the semantic class is static, but the motion label is dynamic, and vice versa. Since individual inconsistent points are not indicative of an anomaly, the inconsistent points are clustered. In this way, entire inconsistent objects can be detected, serving as an indication of an anomaly in the context of this work.

3.2 Method Components

In the following chapters, I describe the used models for each component of my method. For this purpose, I divided the components into SV and SSV components.

3.2.1 Supervised Components

This chapter specifies the selected models for the SV components, i.e., models that learn the mapping between a given input and output during training and thus rely on the availability of ground truth data.

3.2.1.1 Supervised Semantic Segmentation Model

I chose SalsaNext [38], a fast, uncertainty-aware, and projection-based network, as the SV semantic segmentation model. Although SalsaNext has the lowest mean IoU among the models in Table 2.1, I selected it because of two main reasons. First, SalsaNext comes with an uncertainty version, which allows to reason about the data uncertainty and how uncertain the model's predictions are. This uncertainty version can be placed on top of the already trained network and does not require any additional training. Second, a SV motion object segmentation model based on SalsaNext [79], see Chapter 3.2.1.2, enables to further divide semantic classes into dynamic and static, resulting in *semantic motion classes*. Thereby, semantic classes that can be both static and dynamic can be subdivided.
SalsaNext builds upon the SalsaNet model [80] and achieves over 14% more accuracy by introducing a new context module, capturing more spatial features, adding pixel-shuffle layers, applying central dropout, inserting average pooling, and extending the loss.
SalsaNext takes a 2D range view image as input, allowing the use of 2D convolutions. This is done by projecting a 3D lidar point (x,y,z) onto a spherical projection, moving the origin to the top left corner of the image, and scaling the image based on the sensor used [81]. This results in 2D coordinates (u,v), calculated by the following equation

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\left[1-\arctan(y,x)\pi^{-1}\right] w \\ \left[1-\left(\arcsin\left(z,r^{-1}\right)+f_{\text{down}}\right) f^{-1}\right] h \end{pmatrix} \quad [3.1]$$

with h and w representing the height and width of the image, r the range of each 3D point computed by $r=\sqrt{x^2+y^2+z^2}$, and f the sensor specific vertical field of view corresponding to $f=|f_{down}|+|f_{up}|$ [38]. The result of the projection is an $[w \times h \times 5]$ image, where the lidar coordinates, the intensity values of the received lasers, and the range of the points are stored as separate channels in the image. An example of a 2D projection is shown in Figure D.1.

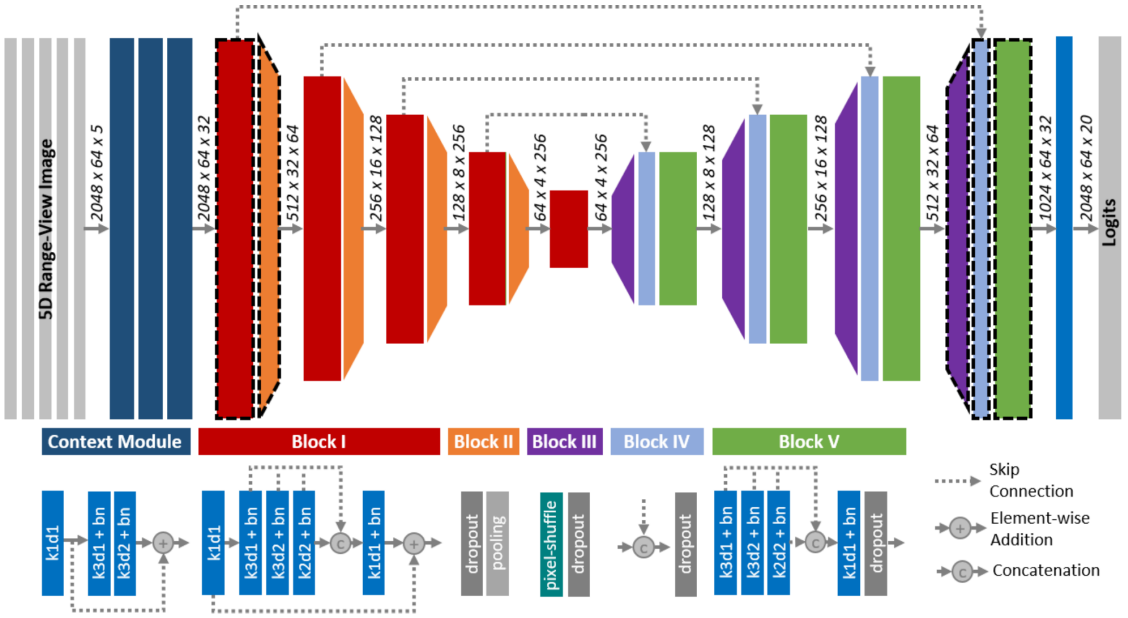


Figure 3.2: Architecture of the SalsaNext model. The network is divided into blocks, where blocks with dashed edges do not apply dropout during training. The letters *k*, *d*, and *bn* indicate the kernel size, dilation rate, and batch normalization [38].

The final output of the network are point-wise classification scores for each pixel of the image. To cope with information loss during inference by back-projecting the pixel-wise predictions to point-wise predictions, the authors applied a kNN-based post-processing step, introduced by the authors of [82]. Milito et al. [82] argue that, without this post-processing step, it is possible that points, which were stored in the same image pixel, are classified as the same class, leading to misclassifications, especially of object boundaries. The kNN-algorithm selects neighboring points for each lidar point by using a window around the corresponding pixel. From this collection of neighboring points, a certain number of the nearest points are selected to accumulate the votes from all the labels. Based on the votes, the final labels for the input point cloud are determined. The absolute difference in the range serves as a proxy for the euclidean distance and is used as the distant metric for the nearest neighbor search. This way, the algorithm can run in real-time.

The network of SalsaNext is an encoder-decoder architecture, with the encoder encoding spatial information to some kind of features, and with the decoder up-sampling and reassembling the features. The architecture of the network is shown in Figure 3.2. The total loss function of the SalsaNext model is composed of two parts. The first part, is a weighted softmax cross-entropy term that copes with the imbalanced class problem. Class imbalance occurs due to a non-balanced representation of all classes in the dataset, e.g., the class car occurs significantly more often than the class bicycle. This can lead to misclassification, especially for underrepresented classes, as the network "will typically over-classify the majority group due to its increased prior probability" [83]. For this reason, the authors of SalsaNext use the weighted softmax cross-entropy loss, where the weights are the inverse square root of the number of points per class. As the second part of the total loss, they use the Lovász-Softmax loss, presented by the authors of [84], to represent and

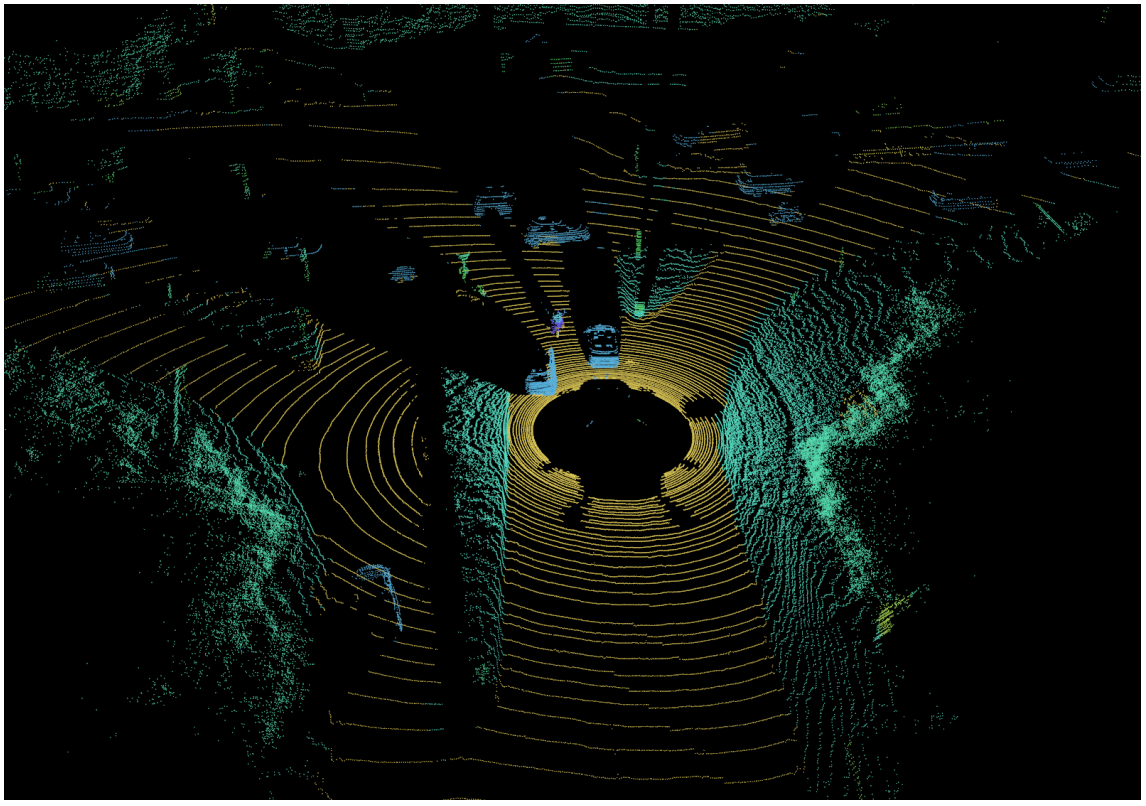


Figure 3.3: Result of a semantic segmentation by SalsaNext of a point cloud into different classes, where each class is represented by a different color. The point cloud is taken from sequence 12, frame 1019 of the KITTI odometry dataset [49].

optimize the mean IoU directly. A linear combination of both parts results in the total loss. One result of a semantic segmentation is shown in Figure 3.3.

In addition to the classical semantic segmentation, where each point is assigned a class, SalsaNext also has an uncertainty version that can be applied to the already trained model. To obtain the uncertainty of the prediction of the model, I used MC dropout. When applying MC dropout, the same input is passed through the network multiple times, while different neurons are randomly switched off. In the published code [85], the way to get the predictions for the semantic class for each pixel from the MC samplings is not complete. Therefore, I applied a voting ensemble, where the predictions for each pixel are summed across the MC samples and the most frequently occuring label is considered the prediction for that pixel. Due to the non-linear output of the softmax classifier, the mean cannot be used, and a discrete representation is given with the voting ensemble. A comparison of the predictions from the voting method with the predictions from the classical SalsaNext model without MC dropout showed that the same mean IoU is obtained. The authors of SalsaNext calculate the epistemic uncertainty by taking the average of the variance of the MC sample results. As a consequence, the uncertainty values do not represent the uncertainty of the predicted class but the uncertainty over all MC samples, regardless of the prediction made. To obtain the uncertainty for the predicted class only, I took the variance across all MC samples and only selected the uncertainty of the pixel-wise predicted class, determined in advance by ensemble

voting. This gives a value for each point that shows how uncertain the model is with respect to the predicted class.

3.2.1.2 Supervised Motion Object Segmentation Model

Most of the existing semantic segmentation models assign a class to each point, and it is not possible to distinguish between moving and static objects based on the defined classes. For example, the class *car* does not allow to infer whether the detected car is standing or driving in the case of a red light. Since the method presented in this thesis is based on inconsistencies between semantic classes and the motion label predicted by the combination of SSV models, it is important to further divide the points of potentially moving classes. Otherwise, many anomalies would appear as inconsistencies due to an insufficient class definition. For this reason, I integrated another SV model that can distinguish between moving and static objects, resulting in *semantic motion classes*. An example where a semantic class was further subdivided based on the motion labels can be seen in Figure 3.4.
I decided to use the model of Chen et al. [79], as it is based on top of existing range projection-based lidar semantic segmentation networks, like SalsaNext. The model uses residual images generated from previous scans as additional channels to the range view image to incorporate temporal information. To decide whether an object is dynamic or static, one must distinguish between the ego-motion of the observer and the actual movement of the object itself. To compensate for ego-motion, Chen et al. [79] transformed previous scans into the current local coordinate system by using odometry data from SemanticKitti [45] that was estimated with a surfel-based SLAM approach [86]. They use the estimated relative poses between consecutive scans to transform scans captured at a different time into the current frame. The idea behind this is that dynamic objects have their own motion in addition to the ego-motion, and transforming them to the same local coordinate system reveals this motion by compensating for the ego-motion of the vehicle. Subsequently, the transformed scans are projected into the current range view image, according to the Equation 3.1. The residual value for each pixel as the normalized absolute difference between the ranges of the current and the previous scans is calculated by

$$d_{k,i}^{l} = \frac{\left| r_i - r_i^{k \to l} \right|}{r_i} \qquad [3.2]$$

with $d_{k,i}^{l}$ the residual of pixel i between the current local frame l and the transformed scan of frame k. The range is defined as $r = \sqrt{x^2 + y^2 + z^2}$ where r_i represents the range value of point i located at the pixel coordinates (u_i, v_i) and $r_i^{k \to l}$ the range value of the transformed scan located at the same image pixel. Together with the already existing channels resulting from the range view image, the residual images are added as additional channels and serve as an input for the SalsaNext model. The residual images provide temporal information, while the range view images provide spatial information to the network.
Chen et al. [79] found that the more previous scans and thus the more residual images are considered, the better the result. This holds up to a number of 8 images. Each additional image provides

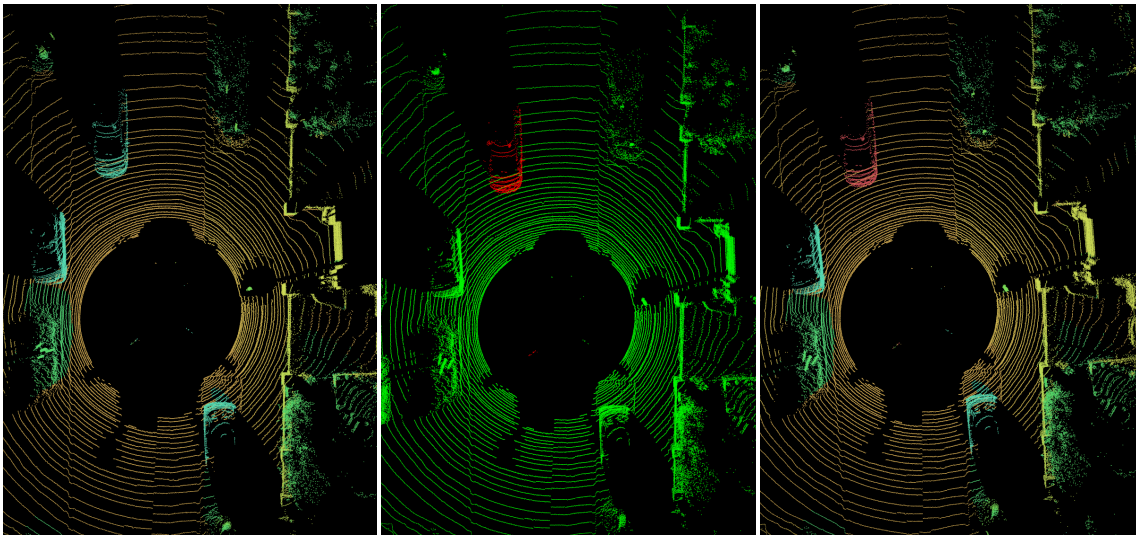


Figure 3.4: Example sequence, where the semantic class *car* has been further divided into the classes *dynamic car* and *static car*. The left image is the result of semantic segmentation and the middle image shows the result of motion segmentation, with dynamic objects colored red and static objects colored green. In the right image, the semantic labels are combined with the motion labels, so that the moving car in red now differs from the parked cars in green. The point cloud is taken from sequence 11 frame 450 of the KITTI odometry dataset [49].

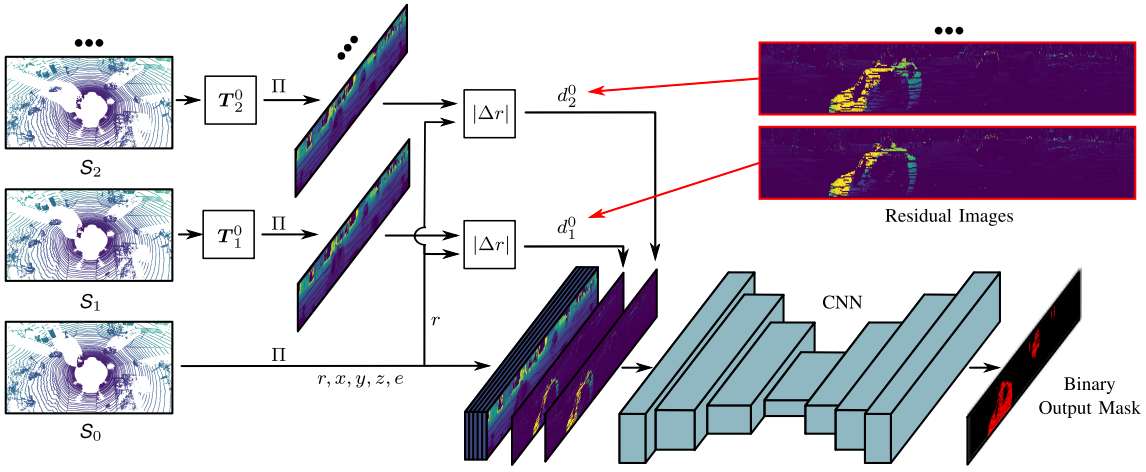


Figure 3.5: Overview of motion object segmentation model [79]. To generate the residual images, past scans are needed, as shown on the left side. In the middle at the bottom the additional channels consisting of the residual images are shown, which are used as input for the CNN, which in my case is SalsaNext.

only marginal further improvement. The output of the network is a binary mask indicating whether a point is static or dynamic. An overview of the motion object segmentation model is shown in Figure 3.5.

3.2.1.3 Ground Segmentation Model

Most SSV scene flow models, including the one I chose, remove ground points during both training and inference [57, 59, 60, 62, 63, 71]. As mentioned in Chapter 3.1, I decided to use a ground segmentation model instead of heuristics. I chose to use GndNet [87], as it is a fast ground plane estimation and ground segmentation model that operates on raw point clouds. In a first step, the model discretizes the point cloud into a 2D grid, resulting in a set of pillars, introduced by [88]. Then, PointNet [89] learns pillar-wise features and creates a 2D pseudo image. The pseudo-image serves as input to a convolutional encoder-decoder network [90] that extracts spatial features and regresses the height of the ground for each cell in the grid. With the elevation of the ground for each cell, points that are below the elevation can be classified as *ground* and those that are above can be classified as *non-ground*. For training, the authors of GndNet created annotated ground elevations for the SemanticKitti dataset. The model achieved a mean IoU of 83.6%. For my approach, I used the publicly available pre-trained model. A prediction of the pre-trained GndNet for a point cloud from the KITTI-360 dataset is shown in Figure 3.6.

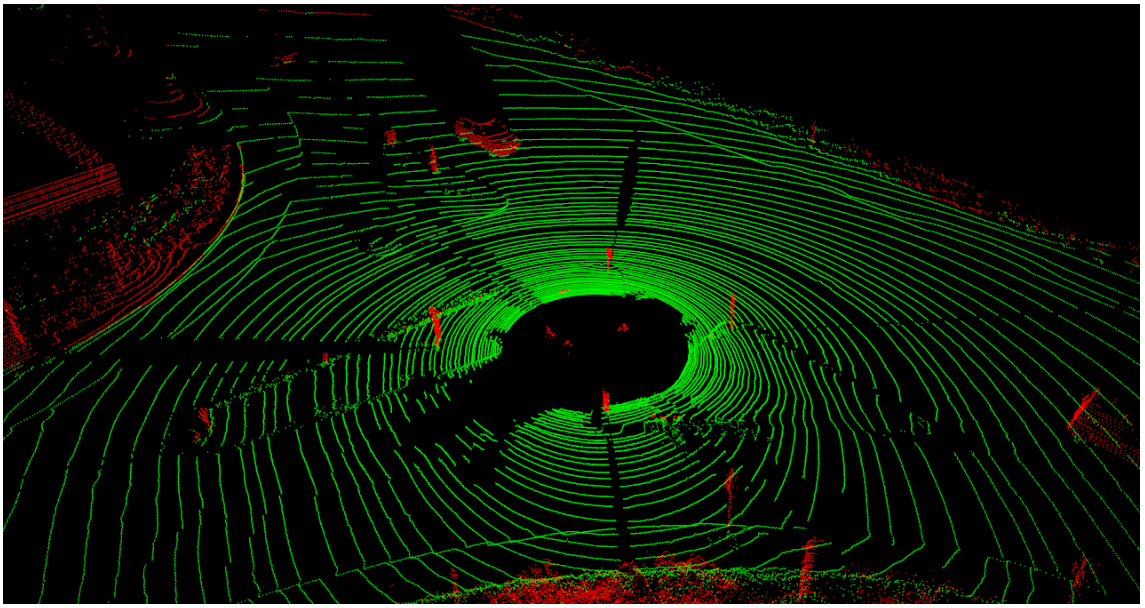


Figure 3.6: Ground segmentation results of sequence 3, frame 134 of the KITTI-360 dataset, where points that have been classified as *ground* are colored green and all others are colored red.

3.2.2 Self-Supervised Components

In this chapter, I describe the selected models for the SSV components, i.e., models that do not depend on the availability of ground truth data and whose supervision is performed by the model itself.

3.2.2.1 Self-Supervised Scene Flow Model

Amongst the models listed in Table 2.2, FlowStep3D has the best performance on the KITTI-SF dataset [68, 69] together with the SLIM model [57]. For this reason, I chose FlowStep3D [60] as the SSV scene flow model. The model has a significantly lower outlier rate and better accuracy than the model of Tishchenko et al. [63] and Ouyang et al. [64]. Unlike SLIM, FlowStep3D's code had already been published at the time of the decision.

FlowStep3D has a recurrent architecture and works iteratively by refining an initial flow in each step. This results in a flow sequence $\{F_1, ..., F_K\}$, where F_1 is the initial flow and F_K the final flow. A high-level overview of the model architecture can be found in Figure 3.7.

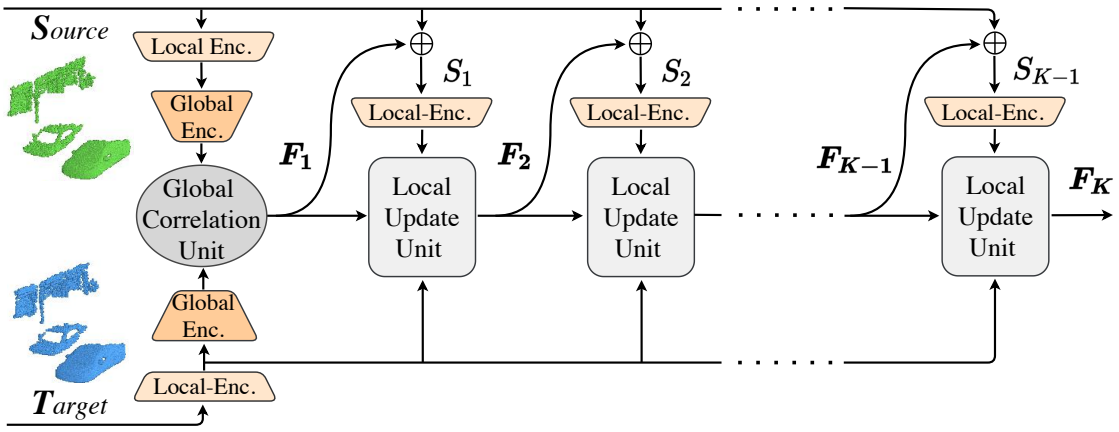


Figure 3.7: High-level overview of the chosen scene flow model, FlowStep3D [60]

In a first step, a local encoder extracts local features of the source point cloud S and the target point cloud T to preserve geometry features of the immediate neighborhood. The local encoder consists of two *set_conv* layers, introduced by the authors of [61]. Then, the local features are further encoded to obtain global features to capture the relative position in the scene. A global correlation unit estimates the initial flow F_1 by computing the cosine similarity between the feature vectors of the source and target point cloud and by constructing a coarse all-to-all correlation matrix. The initial flow is given by the average distance between the coarse points of the down-sampled source point cloud and all coarse points in the down-sampled target point cloud, weighted by the correlation matrix and up-sampled by a set of *set_up_conv* layers.

Starting from the initial flow, the flow is refined in each iteration using local information. The predicted flow from the previous iteration is used to warp the points of the source point cloud, which in turn serve as input for the local encoder that extracts new local features. The local update unit consists of a single conceptual iteration of an ICP provided with learned components. In the local update unit, the *flow_embedding* correlation layer, presented by [61], is applied to learn to correlate the warped source and the target. It generates flow embeddings for the points in the warped source by aggregating feature similarity and spatial relationships between the local features from the warped source and the target. To refine the estimated flow from the previous iteration, a gated activation unit [91] based on a Gated Recurrent Unit (GRU) [92] cell followed by two

set_up_conv layers estimate the flow refinement ΔF_k. The refined flow at the end of each iteration is computed as $F_k = F_{k-1} + \Delta F_k$.
To train their network in a SSV way, the authors of FlowStep3D implemented a Chamfer Loss and a regularization loss. The Chamfer distance ensures, that the source point cloud is shifted towards the target point cloud, according to the points that are closest to each other. The regularization loss controls the convergence of the Chamfer distance to the global minimum and makes sure that the structure of the objects is preserved during warping.

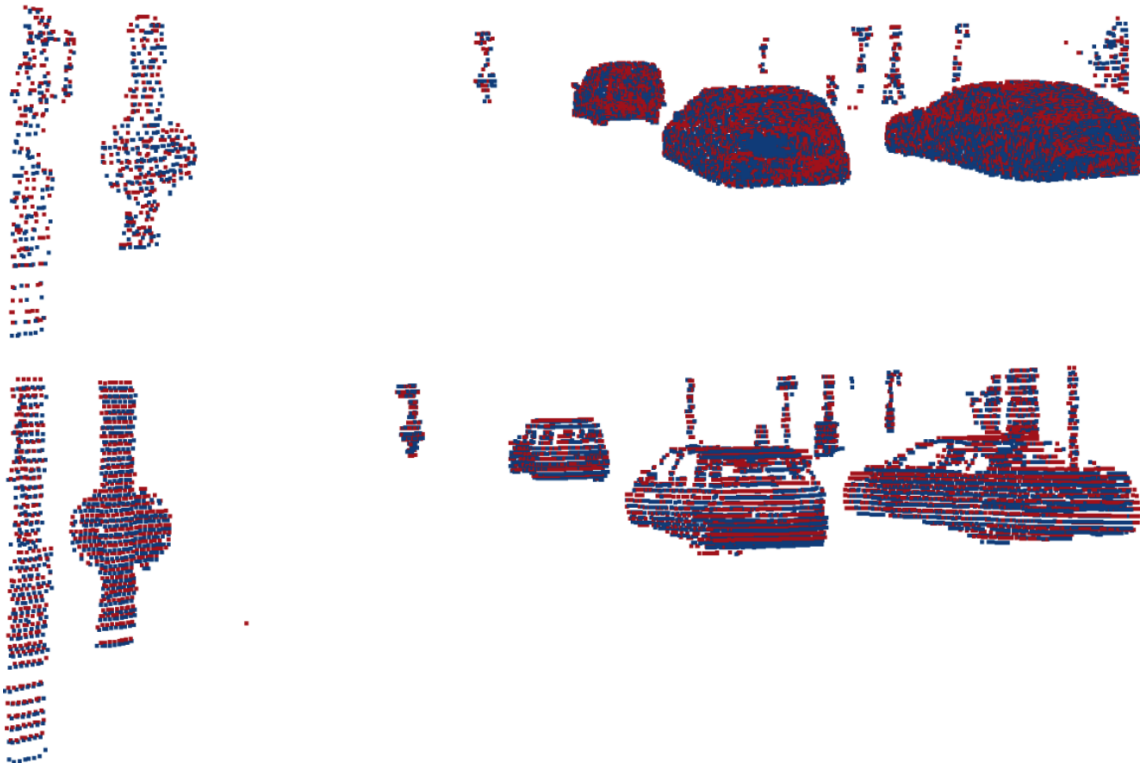


Figure 3.8: A comparison between the predicted flow of the same scene once on the scene from the KITTI-SF dataset, upper image, and once on a raw Velodyne scan from the KITTI raw dataset with the mentioned pre-processing steps, shown in the bottom image. The point cloud at time $t+1$ is shown in red, and the point cloud at time t transformed with the corresponding estimated flow is shown in blue. The scene shown is the 52nd scene in the KITTI-SF dataset and belongs to frame 107 in sequence 2011_09_26_drive_0018 of the KITTI raw dataset.

For the evaluation of FlowStep3D on the KITTI-SF dataset, the authors removed ground points with a threshold of 0.3 m and points further away than 35 m. The remaining points were down-sampled to 8,129 points. To be consistent with the training data, they rotated the x and y axis by 180°. This resulted in two consecutive point clouds in camera coordinates from which ground points were removed, a depth threshold was applied, and where the x and y axis were flipped by 180°. In addition, only points within the camera's field of view were considered.
Although FlowStep3D was trained only on synthetic data from the FT3D dataset, it showed great generalization capabilities on the real-world KITTI-SF dataset [60]. For this reason, I used the provided pre-trained model for my approach.
A raw scan from a Velodyne HDL-64E scanner provides a 360° view and consists of about 120,000

points. To apply the pre-trained model of FlowStep3D to Velodyne scans, some pre-processing steps were necessary. First, I transformed the raw point cloud from the Velodyne coordinate system to the camera coordinate system, keeping only the points that are in the field of view of the forward-facing camera. This is done by using the remaining indices when mapping the points to the image. Second, I rotated the x and y axis by 180°, so that the x axis points to the left, the y axis to the top, and the z axis points forward. All directions are given from the point of view of the recording vehicle. Third, points further away than 35 m away and points previously classified as ground by the ground segmentation model were removed. All these pre-processing steps result in an average of about 8,000 points remaining from the initial 120,000 points. Figure 3.8 shows an example of the same scene with the corresponding flow estimated by FlowStep3D, once from the KITTI-SF dataset (top image), and once from the KITTI raw dataset (bottom image). The point cloud at time $t+1$ is shown in red, and the point cloud at time t predicted into the time $t+1$ with the flow in blue. Based on the overlapping points in the lower image and the similarity with the upper image, it can be seen that the pre-trained model of FlowStep3D can be well applied to raw Velodyne scans when the introduced pre-processing steps are performed.

3.2.2.2 Self-Supervised Odometry Model

To determine the ego-motion of the vehicle and to compensate the flow for the ego-motion, I used DeLORA [75], a SSV odometry model working with lidar data. The architecture of DeLORA is shown in Figure 3.9. The model takes two consecutive point clouds represented as range view

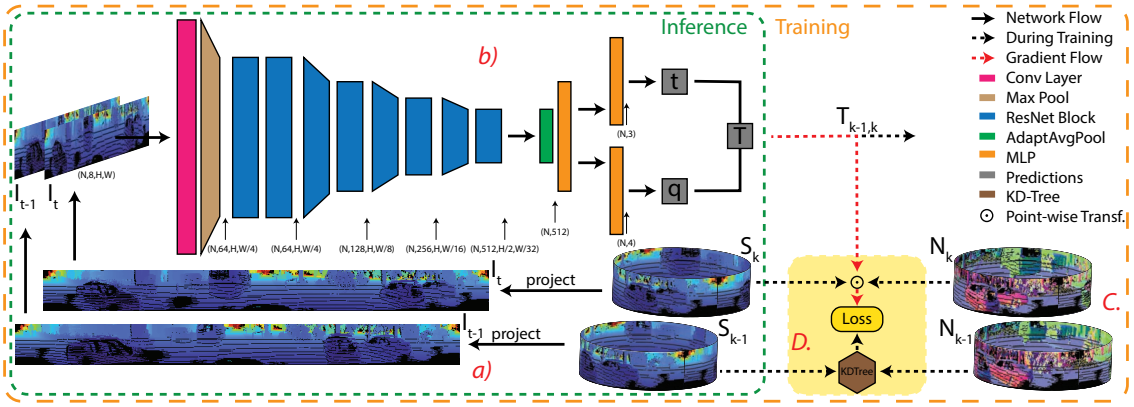


Figure 3.9: Architectural overview of the DeLORA model, where a), b), C. and D. refer to the range view image representation, the network, the normal vectors and the loss, respectively [75].

images as input using the Equation 3.1. Due to the 2D representation of the 3D point cloud, standard 2D convolutions are applied up to a point where the feature vector is split into two MLPs that are used to estimate translation and rotation separately. During training, the estimated rigid body transformation is applied to the source scan. Then, a correspondence search is performed in 3D using a KD-Tree to find pairs between the transformed source scan and the target scan. These pairs are used along with precomputed normal vectors for the loss, which consists of the point-to-plane and the plane-to-plane loss. The point-to-plane loss computes the residual of the matched pairs and projects the residual onto the target plane, while the plane-to-plane loss compares the

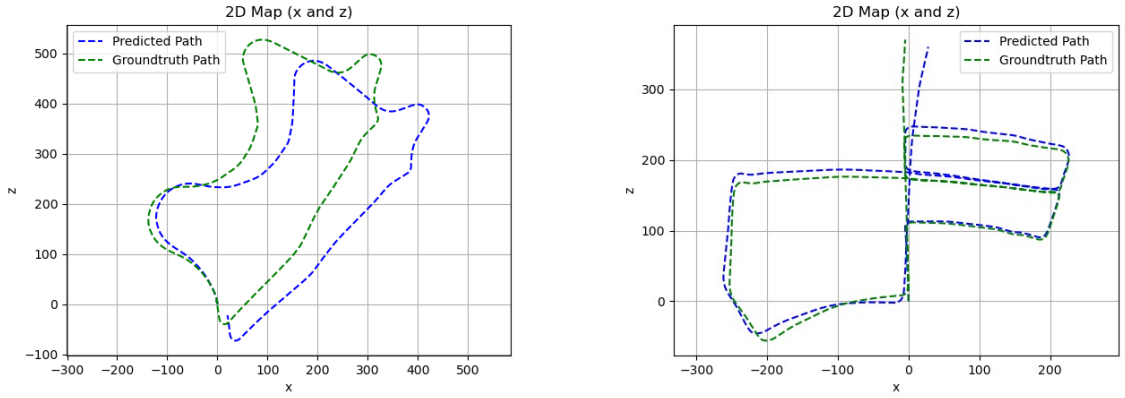


Figure 3.10: Qualitative results of sequences 05 and 09 of the KITTI odometry dataset. Ground truth data are available for sequence 05 and 09. Sequence 05 was used for training.

surface direction of the pairs [76].
For my approach, I used the pre-trained model trained on sequences 00-08 of the KITTI odometry dataset [49]. Figure 3.10 shows qualitative results of the DeLORA model with the KITTI odometry dataset.

3.2.2.3 Motion Labels by Combining Scene Flow and Odometry

To classify each point in the SSV part as dynamic or static, I combined the flow vectors and the relative transformations from the odometry model. The idea behind the combination of the two models is that the flow vectors of dynamic objects stands out compared to static objects after compensating for the ego-motion by the rigid body transformation. An example scene for a better understanding of the individual steps can be seen in Figure 3.11.

In detail, I used the post-processed point clouds that served as input for the scene flow model and applied the associated predicted scene flow. This results in point cloud $X_{t+1}^{f} = X_t + f_t$, where X_t refers to the original point cloud at time t and f_t to the predicted scene flow of point cloud X_t. Since the estimated odometry information is based on lidar data, I transformed both point clouds X_t and X_{t+1}^{f} back to lidar coordinates. To obtain for each point only the motion induced by itself and to classify a point as static or dynamic based on this, I mapped the point cloud X_{t+1}^{f} at time $t+1$ into time t using the estimated rigid body transformation of the odometry model. The resulting point cloud X_t^{fT} is adjusted for the ego-motion. In the far left image in Figure 3.11, an example of the result of this transformation is shown. The two point clouds X_t and the point cloud X_t^{fT} are visualized together in green and red, respectively. Static objects like parked cars overlap to a large extent, and dynamic objects like the two bicyclist in the center are shifted against each other as they have their own movements.
By subtracting the point cloud X_t^{fT} with the point cloud X_t, I obtained f_t^{T} as the scene flow compensated for the ego-motion. For the cross-frame and cross-sequence analysis of the flow vectors, I converted the magnitude of the flow vectors to speed in $\mathrm{km\,h^{-1}}$ to obtain a more interpretable

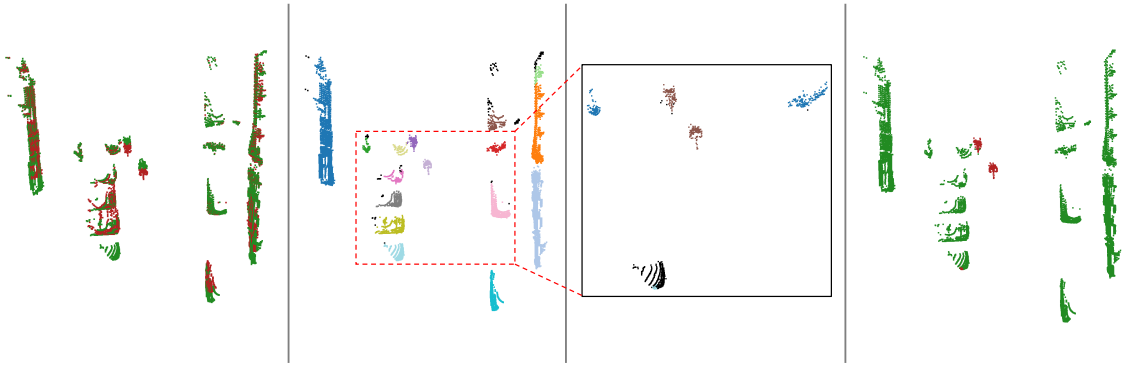


Figure 3.11: Example scene showing the steps of combining scene flow, odometry data and clustering in order to perform motion segmentation. In the first image from the left, the transformation of a point cloud estimated in advance with the flow vector into the next image was transformed back into the original image. The original point cloud is shown in green and the back-transformed point cloud is shown in red. The second image from the left shows the result of the first clustering of the points, with the different clusters represented by different colors. The third image from the left shows the result of the second clustering, this time based on the flow vectors of the potentially dynamic clusters. Again, different colors indicate belonging to different clusters. The last image shows the final result of the process, where the points classified as dynamic are shown in red and the static ones in green. The scene shown is taken from sequence 0, frame 648 of the KITTI odometry dataset.

uniform unit using the timestamps at time t and $t+1$. As it can be seen in Figure 3.12 it seemed difficult to find a rule to classify points into static and dynamic only based on the individual flow vectors. However, Figure 3.13 shows that the normalized standard deviation per instance provides a first distinction between dynamic and static instances. Thereby, the normalized standard deviation was calculated by the ratio between the standard deviation of the speeds and the mean value of the speeds per instance. The boxplot shows that the speed of the points of the dynamic instances has a low standard deviation compared to the mean speed of the instance. For this boxplot, ground truth instance and semantic labels from SemanticKITTI [45] were used.

To obtain instances that are not based on ground truth data, I spatially clustered the point cloud X_t by using DBSCAN [93], a density-based clustering algorithm. This allowed me to draw conclusions about the distribution of the speed per cluster. For the DBSCAN algorithm, I set $\varepsilon = 0.6$ and the minimum size of clusters to 30 points, as these performed best qualitatively based on a few selected sequences. In the second image from the left in Figure 3.11, shows an example of the spatial clustering results, with different clusters represented by different coloring.

Based on the normalized standard deviation of speed per cluster, I selected potentially dynamic clusters. Accordingly, a cluster was classified as potentially dynamic if the normalized standard deviation of the cluster was less than 0.12. This value was determined in a grid search from the values 0.15, 0.17, 0.2, 0.23 as the value with the highest IoU for dynamic points using semantic labels from SemanticKITTI [45]. When calculating the normalized standard deviations per cluster, points whose speed exceeded $80\,\mathrm{km\,h^{-1}}$ were treated as outliers and not included in the calculation. Because the normalized standard deviation threshold is not fully selective, and because of noise and inaccuracies in the scene flow model, some static clusters are categorized as potentially dynamic. To cope with these clusters, I only considered the points of potentially moving

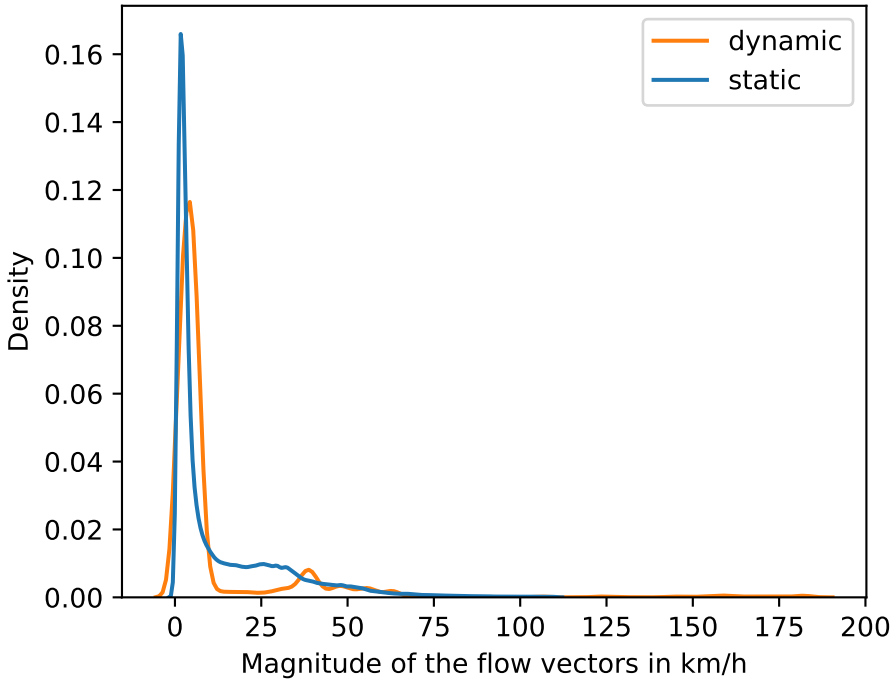


Figure 3.12: Distribution of the magnitudes of the flow vectors of dynamic and static points. All data exceeding the 99% quantile were not included in the plot and were treated as outliers. For this figure, the points and their corresponding flow vectors from sequence 9 and 10 of the KITTI odometry dataset were divided into static and dynamic points using the ground truth semantic labels from SemanticKITTI [45].

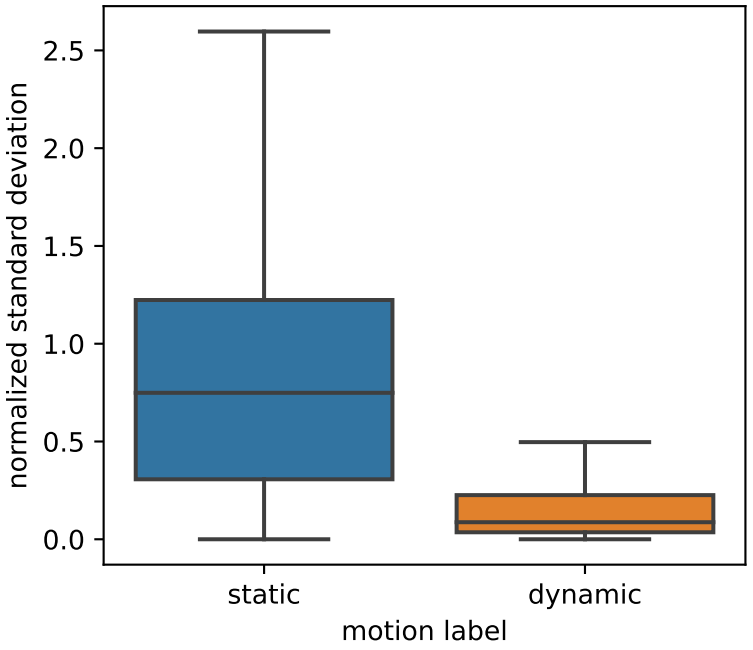


Figure 3.13: This figure shows that the normalized standard deviation of dynamic instances is significantly lower than the normalized standard deviation of static instances. For this figure, the points of sequence 9 and 10 were split into dynamic and static instances by using the ground truth instance and semantic labels of SemanticKITTI [45]. Then, for each instance, the normalized standard deviation was calculated by taking the ratio between the standard deviation of the speeds and the mean of the speeds per instance.

clusters and clustered them again based on the associated flow vectors f_t^T. This means that originally non-clustered, spatially separated points can now be grouped together based on their flow. Static clusters whose normalized standard deviation fell below the value of 0.12 and were thus considered potentially dynamic are now clustered together with other static clusters. Forming new clusters changes the distribution of flow vectors within those clusters, causing less static clusters to be misclassified as dynamic. For the second clustering, I used again DBSCAN with an epsilon of 0.015 and a minimum cluster size of 25 points. The value for epsilon was determined in a grid search from the values 0.015, 0.017, 0.02, 0.023, 0.0026 as the value with the highest IoU for dynamic points using semantic labels from SemanticKITTI [45]. In the third image from the left in Figure 3.11 shows only the potentially dynamic clusters, this time clustered by the corresponding scene flow vectors. As it can be seen from the same coloring, the two bicyclists are clustered together. In addition, a few points on the left and right edges, which belong to static objects, are also clustered together, recognizable by the blue coloring. Black points indicate that these points do not belong to any cluster, as the DBSCAN algorithm treated them as outliers.
When the normalized standard deviation of the newly found clusters is below the value of 0.12 and the median speed of the cluster reaches the threshold value of $4\,\mathrm{km\,h^{-1}}$, the cluster points are labeled as dynamic. All other points are labeled as static. The threshold of $4\,\mathrm{km\,h^{-1}}$ was chosen because this speed could be observed in pedestrian clusters. Furthermore, with a lower threshold, many static clusters are misclassified as dynamic. The last image in Figure 3.11 displays the result, with green representing static points and red representing dynamic points. More examples can be found in the Appendix E.

3.2.3 Comparison and Anomaly Detection

Since my approach is based on finding inconsistencies between the SV models and the SSV models, I compared the SSV motion labels to the SV semantic motion labels point-per-point.
Since my SV part provides labels for an entire scan of the lidar sensor, i.e. for points within 360° of the car, I filtered out only those points for which SSV labels were available. Then, the semantic motion classes were divided into static and dynamic, so that for each point I get a classification into static or dynamic once from the SV and once from the SSV models. When comparing the two labels per point, four scenarios can occur, see Table 3.1.

	SV Part	SSV Part	Consistent	Color
Scenario 1	static	static	✓	green
Scenario 2	dynamic	dynamic	✓	blue
Scenario 3	static	dynamic	✗	red
Scenario 4	dynamic	static	✗	yellow

Table 3.1: All possible scenarios that can occur when comparing the labels between the SV and the SSV part. The column “Color” indicates which color was used to visualize each scenario during the evaluation.

Inconsistencies occur when scenario 3 or 4 appears. Since individual inconsistent points are of

less interest, I clustered points from scenarios 3 and 4 separately per point cloud using DBSCAN with $\varepsilon = 1$ and a minimum cluster size of 30 points.

3.3 Training Dataset

For our methods, we needed a large lidar dataset with semantic labels, motion labels, and odometry information. Since KITTI-360 [46] meets these requirements, we used it for training. KITTI-360, considered to be the successor of the KITTI dataset, "is a suburban driving dataset that comprises richer input modalities, comprehensive semantic instance annotations and accurate localization to facilitate research at the intersection of vision, graphics and robotics" [46]. The dataset consists of over 300k images and 80k laser scans, corresponding to a driving distance of 73.7 km. As the name of the datasets suggests, a 360° view is provided once by the lidar scanner but also by fisheye cameras.

The annotation tool of KITTI-360 allows to annotate static and dynamic points directly in 3D. To reduce data loading traffic and memory consumption during annotation, the authors down-sampled the raw point clouds. For this purpose, the point clouds were sequentially fused and points closer than 5 cm to the nearest neighbor were removed. As a result, about 9.58 billion raw points for training are down-sampled to about 835 million points. The authors of KITTI-360, Liao et al. [46], published raw point clouds and annotated accumulated point clouds, where the accumulated point clouds were further divided into static and dynamic points.

Since semantic labels were only published for the accumulated point clouds, I recovered the labels for the raw points by performing a nearest neighbor search. To this end, I extended an existing script that recovers semantic labels based on the static accumulated point clouds by considering the dynamic accumulated points as well [94]. Since the accumulated points were in world coordinates and the raw point clouds were in Velodyne coordinates, the raw points first had to be transformed into the world coordinate system. The Velodyne coordinate system differs from the world coordinate system in that the origin is located at the Velodyne scanner and thus moves with the vehicle. The world coordinate system has a fixed origin for all scans. Poses are required for the transformation from the Velodyne coordinate system to the world coordinate system, including the transformation from GPU/IMU coordinates to the world coordinate system. KITTI-360 does not provide poses for all scans. The authors only list poses from scans where the distance between the current and the last valid scan is greater than a threshold. For this reason, poses from scans recorded at a very low speed were ignored. Of the total 81,106 scans, poses are available for 64,640 scans, so labels could only be recovered for those 64,640 scans.

After both the accumulated and raw point clouds are in world coordinates, the nearest neighbor in the accumulated point cloud is searched for each raw point. If the distance between the raw point and the nearest neighbor does not exceed a threshold of 0.5 m, the raw point is given the label from the nearest neighbor. Otherwise, the raw point is classified as "unlabeled". By combining the dynamic and static accumulated points in the nearest neighbor search, labels for the whole scene could be recovered.

As a result, I obtained annotations for all sequences of raw point clouds for both static and dynamic

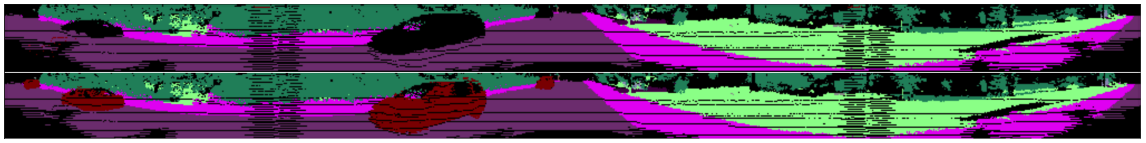


Figure 3.14: The upper image shows the result of semantic label recovery, which was performed only on the basis of the accumulated static points. The bottom image shows the result of semantic label recovery, where the accumulated dynamic and static points were considered together. One can see from the black colored points in the upper image, which refers to unlabeled points, that no labels were recovered for the moving car in the foreground. In contrast, the points of the moving car are colored red in the lower image.

objects. Of the original 9.58 billion raw points, labels could be recovered for 7.65 billion points. For the remaining points, labels could not be recovered due to missing poses. Of the 7.65 billion labels, approximately 9% were classified as "unlabeled", resulting in 6.95 billion semantic labels. A qualitative comparison between the results of the existing label recovery script and the extended script, which considers dynamic points, is shown in Figure3.14.
In addition to semantic labels for training the semantic segmentation model, I needed motion labels for training the motion object segmentation model. Motion labels denote whether a point is dynamic or static. Since the accumulated points are already divided into static and dynamic points, I used the motion label instead of the semantic class of the nearest neighbor during the recovery. In this way, I obtained a dataset of motion labels for each raw point. The accumulated points of KITTI-360 can be divided into 806 million static points and 29 million dynamic points. After recovery, 28.7 million raw points are classified as dynamic and 6.9 billion raw points are classified as static.
I added my changes, including a visualization, to the existing script via a pull request [94].

3.4 Model Training

Of the five models used, the SV semantic segmentation model SalsaNext [38] and the SV motion object segmentation model [79], based on SalsaNext and extended with the residual images, were trained on the KITTI-360 dataset [46]. For training, the KITTI-360 data were divided into training and validation datasets as specified by the authors of KITTI-360. Thus, one part of the data of each sequence was used for training and the other part for validation. The recovered labels, described in Chapter 3.3, were used as ground truth data. The training was performed on an NVIDIA RTX A6000. The hyperparameters were taken from the respective papers [38, 79]. The number of residual images serving as additional input channels of the motion object segmentation model was set to 8 since it has been shown by Chen et al. [79] that each additional residual image brings only a marginal improvement.

4 Evaluation

This chapter contains the evaluation of my method. First, I give a brief evaluation of the results of training the supervised semantic segmentation model and the moving object segmentation model. This is followed by a separate evaluation of the two parts. Finally, the potential of combining the two parts for anomaly detection is evaluated.
To evaluate the performance of the SV and SSV components and the detection of anomalies by combining the two parts, I evaluated the two parts separately in quantitative terms and the combination of the two in qualitative terms. A detailed quantitative analysis of anomaly detection capabilities of the two parts would be possible if semantic motion labels were available for a large number of point clouds. In addition, the training dataset and evaluation dataset should be related to avoid anomalies at the *domain level*. Based on the conducted qualitative analysis, it was determined that inconsistencies between the two parts of my method could potentially be interpreted as indications for an anomaly at the *method level*.

4.1 Evaluation Training Results

As mentioned in Chapter 3.4, I used the entire KITTI-360 dataset for training. The division of the data into training and evaluation datasets followed the authors' division. The semantic segmentation model achieved the best mean IoU across all classes on the validation dataset with 0.516% and the motion object segmentation model with 0.759%. The motion object segmentation model achieved an IoU of 0.519% for dynamic points.

4.2 Evaluation Dataset

Since this approach aims at finding anomalies at the *method level*, the probability that a domain shift leads to anomalies at the *domain level* should be kept as small as possible. Besides changing weather conditions, differences between datasets from various countries can also lead to *domain level* anomalies [1]. Because of that, the datasets used to train the models and to evaluate the anomaly detection capabilities should be closely related. In particular, this includes the environment in which the data was collected, but also the setup, such as the sensor type.

KITTI-360 [46] and KITTI [49] are two strongly related datasets, both containing 3D point clouds acquired by a Velodyne HDL-64E scanner and recorded in Karlsruhe, Germany. The individual components of our method were partially trained on the KITTI-360 and KITTI odometry dataset. The KITTI odometry dataset is a subset of the KITTI dataset, includes 21 sequences, and was used to evaluate the anomaly detection capabilities. SemanticKITTI [45] contains semantic motion la-

bels for sequence 00-10 of the KITTI odometry dataset. SemanticKITTI builds on the KITTI odometry dataset and has semantic motion labels for sequences 00-10. Since we used the pre-trained model of DeLORA and GndNet, only sequences 9 and 10 with semantic motion labels remain, which together contain 2790 frames. Since a detailed quantitative analysis of the anomaly detection capabilities with only 2790 frames is not reasonable, I decided to perform a qualitative analysis on the KITTI odometry test set, which includes sequences 11-21 and thus around 20,350 frames. The selected sequences 11-21 cover a variety of situations, such as sequences recorded on a highway, in suburbs or in a city with many pedestrians and bicyclists. The remaining sequences 9 and 10 of the KITTI odometry dataset, which have ground truth semantic motion labels, were used to evaluate the two parts separately based on their motion segmentation performance in quantitative terms.
The KITTI odometry dataset contains ground truth odometry data for sequences 00-10. Since the SV motion segmentation model requires poses to compute the residual images, I used the poses from SemanticKITTI [45]. To annotate the data, the authors of SemanticKITTI needed poses, which they predicted using a surfel-based SLAM approach [86] and published for all sequences 00-21.

4.3 Quantitative Evaluation of Supervised and Self-Supervised Part

The motion segmentation performance of both parts of my method was evaluated using sequences 09 and 10 of the KITTI odometry dataset. Semantic labels from the SemanticKitti dataset were used as ground truth data. To obtain motion labels from semantic labels, I followed Chen et al. [79] and divided the classes into static and dynamic classes. The SV part assigns each point in a Velodyne scan to a semantic motion class, while the SSV part predicts a motion label only for points in the camera's field of view, with the exception of ground points and points farther than 25 m away. For better comparability, only the points for which both parts predicted a motion label were selected.
The SV part achieved a mean IoU of 0.828, with an IoU for dynamic points of 0.661. In addition, I analyzed the epistemic uncertainty of SalsaNext using the entire KITTI odometry dataset to see how uncertain the network is in predicting different semantic classes. For this purpose, I used MC dropout during inference with a dropout rate of 0.2 and 30 MC samplings. Figure 4.1 compares the mean model uncertainty per semantic class with the frequency of occurrence of the semantic class. It can be seen that classes that occur frequently, such as the class *car*, have comparatively lower uncertainties than classes that occur less frequently, such as the class *bus* or *train*. This plot suggests that model uncertainty depends on the number of points per class in the training dataset. Classes such as road, building, car, sidewalk and terrain occur in almost every point cloud of the KITTI-360 dataset and therefore have a lower uncertainty compared to other less frequent classes. Another plot of the uncertainties per semantic class, represented as a boxplot, can be found in the Appendix F.
The SSV part achieved a mean IoU of 0.578, with an IoU of 0.174 for dynamic points. For comparison, Chen et al. [79] used FlowNet3D [61], a SV scene flow model, to perform motion

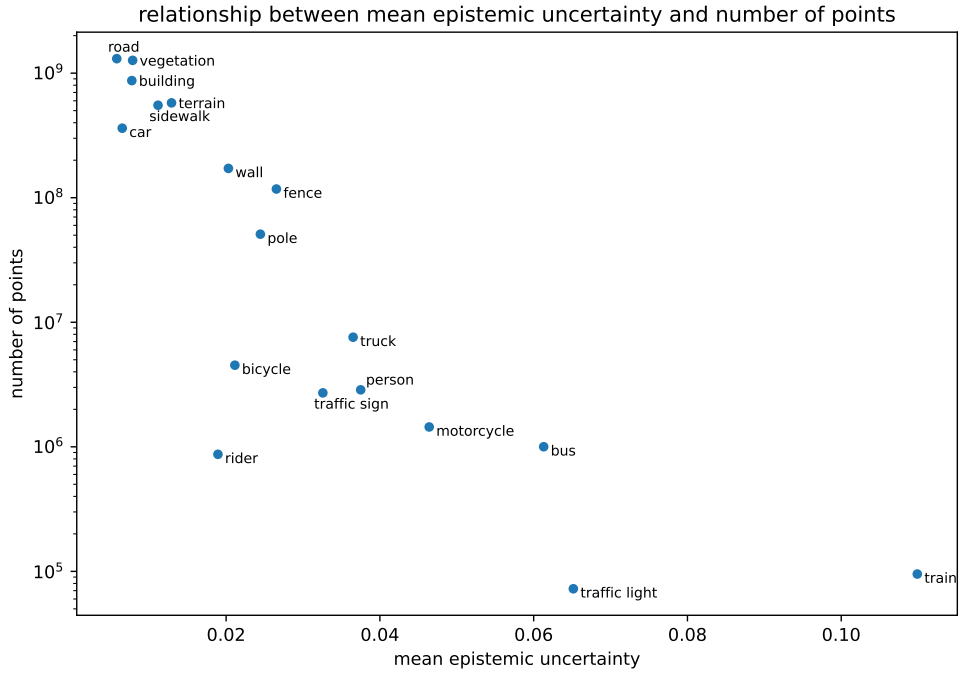


Figure 4.1: SalsaNext's mean epistemic uncertainty per semantic class compared to the number of points. The entire KITTI odometry dataset, sequence 00 to 21, was used for this plot. The y-axis is a logarithmic scale.

segmentation based on the flow of each point using a threshold. By using flow alone, they achieved an IoU for dynamic points of 0.044 on sequences 11-21 of the KITTI odometry dataset. By combining this with SV semantic labels, where a point is labeled as dynamic only if both flow and semantic class indicate a dynamic object, the authors achieved an IoU of 0.287.

4.4 Qualitative Evaluation of Anomaly Detection

For the qualitative analysis, I used sequence 11-21 of the KITTI odometry dataset. The sequences used contain a total of about 20,350 frames, all of which were reviewed. I mapped the lidar points of a frame to the corresponding image. The points of the 4 scenarios were colored differently to directly identify which points were classified as static or dynamic by which part of my method. The assigned colors are shown in Table 3.1. Figure 4.2 shows the number of points per scenario and the number of clusters for the two inconsistent scenarios. From the diagram on the left, it can be seen that the number of points classified as static by both parts is the largest. This is not surprising, since in general most of the points in KITTI are static. The number of points for scenario 3 and 4 where the predictions of the two parts do not agree is significantly higher than the number of points where both parts classify a point as dynamic. During the analysis, I observed a wide variety of cases that can be divided into 4 main categories. I selected representative images for these 4 categories.

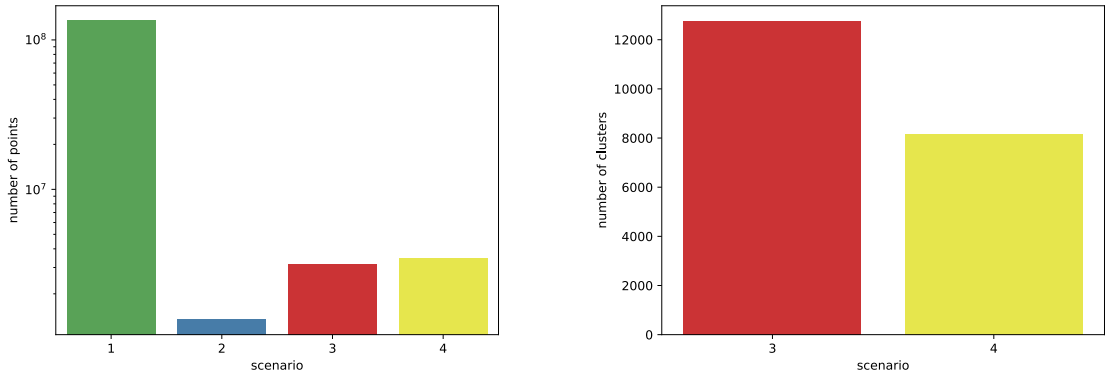


Figure 4.2: The left diagram shows the number of points per scenario. The y-scale is logarithmized. The right diagram shows the number of clusters found for the inconsistent points of the two scenarios 3 and 4.

a) *Correctly consistent*: In this category, I placed images where both the SV and the SSV parts correctly match in their classification. In Figure 4.3, points of dynamic objects, such as a bicyclist, are classified as dynamic by both parts, indicated by the blue coloring. Points of static objects, such as points of parked cars, are classified as static by both parts, recognizable by the green coloring. Only a few points of the one bicyclist are correctly classified as dynamic by the SSV part, shown in red.

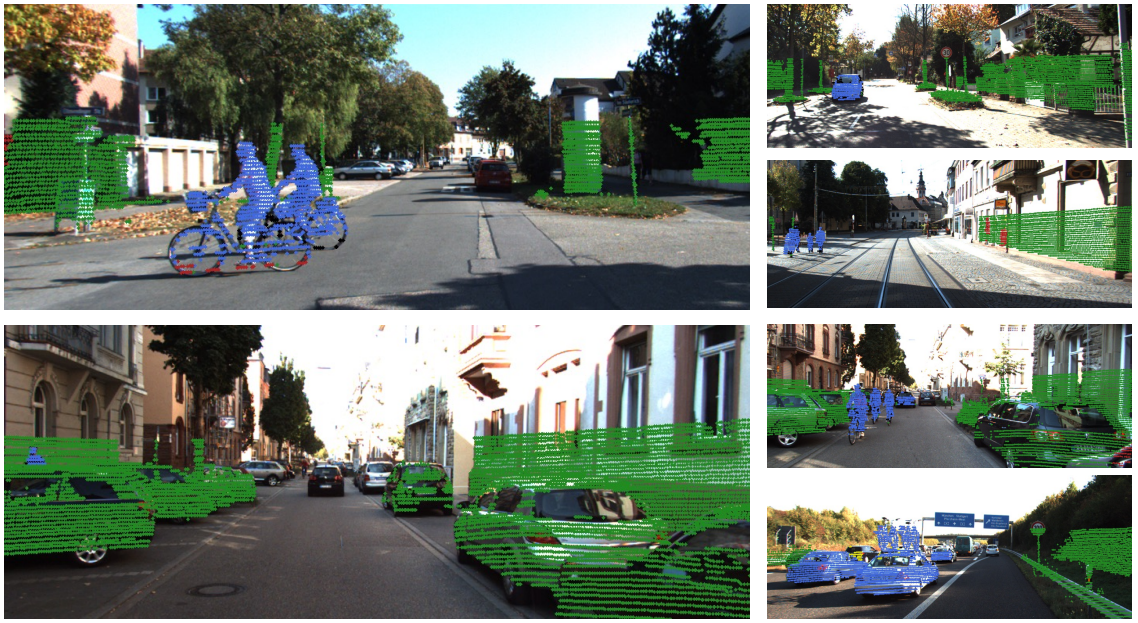


Figure 4.3: Examples where the SV and SSV parts are correctly consistent. Dynamic points are classified as dynamic (blue) and static points as static (green) by both parts.

b) *Incorrectly consistent*: Here, the two parts are incorrectly consistent, as both parts are wrong. For example, in the left image of Figure 4.4, the points of two walking pedestrians are classified as static by both parts. The right image shows a parked car at the bottom left of the image, which was incorrectly classified as dynamic by both parts.

Figure 4.4: These two images show points of objects that were misclassified by both parts of my method. The left image shows two pedestrians classified as static, and the right image shows a parked car classified as dynamic.

c) *Anomalies detected by the SSV part*: Here, the predictions of the two parts are inconsistent, with the SSV part being correct. Example images for this category are shown in Figure 4.5. In image 1, the moving car and one bicyclist were correctly classified as dynamic by both. However, another bicyclist was classified as dynamic only by the SSV part, indicated by the red coloring. In image 2, two walking pedestrians are classified as dynamic only by the SSV part. Image 3 shows a parked car that was incorrectly classified as dynamic by the SV part. In image 4, a slow-moving car and in image 5, a reversing car are detected as dynamic only by the SSV part.

Figure 4.5: Examples of anomalies where the SSV part of my method correctly classified. The anomalies are shown in red and yellow.

d) *Anomalies detected by the SV part*: Here, the predictions of the two parts are inconsistent, while the SV part is correct. The sample images belonging to this category can be found in Figure 4.6. In the first image, two pedestrians located further back in the image are correctly detected as dynamic only by the SV part, recognizable by the yellow coloring. Additionally, points on the right edge are incorrectly labeled as dynamic by the SSV part. Similarly, in image 3, 4, and 5, dynamic objects are labeled as dynamic only by the SV part. In image 2, a parked

car is incorrectly classified as dynamic by the SSV part.

Figure 4.6: Examples of anomalies where the SV part of my method correctly classified. The anomalies are shown in yellow and red.

In addition to the four categories, I identified weak points of the two parts. I defined the weak points of the individual parts as situations that I frequently noticed during evaluation.
Starting with the SSV part, I noticed that there is an above-average number of inconsistencies, especially when the recording vehicle turns around. This is because the SSV part classifies many points as dynamic in such situations. An example is given in Figure 4.7, image 1. I observed a similar behavior when the vehicle drove over a speed bump, see Figure 4.7, image 3. Another weak point of the SSV part are fast oncoming objects, such as cars on a highway, as can be seen in Figure 4.7, image 2. A very common weak point are small clusters that are labeled as dynamic at the right or left edge. An example of this can be seen in Figure 4.7, image 4, where points belonging to a window were incorrectly classified as dynamic.

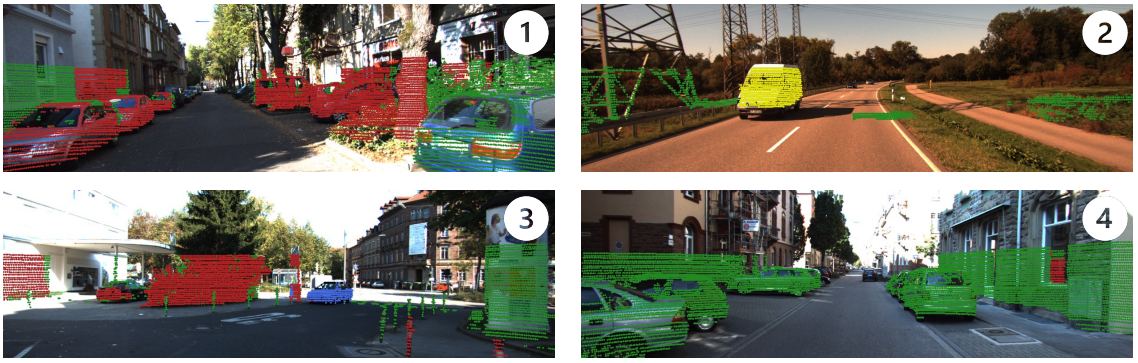


Figure 4.7: Identified weak points of the SSV part.

The SV part seems to have a weakness in distinguishing classes into actually moving or static

in certain situations. As an example, standing cars at traffic lights are very often classified as dynamic, as can be seen in Figure 4.8, image 1. Similarly, parked cars are sometimes misclassified as dynamic, see Figure 4.8, image 3. Another example is standing people at bus stops or at street corners, which are incorrectly classified as dynamic, shown in Figure 4.8, image 2 and 4.

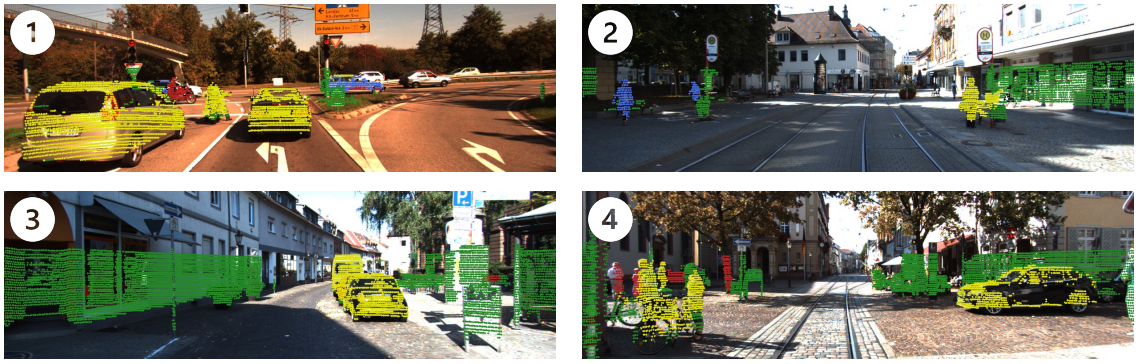


Figure 4.8: Identified weak points of the SV part.

4.5 Evaluation Clustering

In addition to the weaknesses of the SSV models that became apparent when the SSV labels are combined with the SV labels, I identified another weakness related to the first clustering during the combination process of scene flow predictions and odometry information, which is described in Chapter 3.2.2.3. In the first clustering, where points are spatially clustered, dynamic points can be clustered with static points. This can be the case, for example, when a bicyclist passes very close to a parked car. Since potential dynamic objects are selected in the next step based on the normalized standard deviation, such clusters would not be selected. The flow vectors of a dynamic and a static object would differ too much to fall below the normalized standard deviation threshold for potential dynamic objects. As a result, the two static and dynamic objects clustered together will be labeled as static. An example scene is shown in Figure 4.9, where two consecutive frames are shown in separate columns. On the left side in the upper image, a car and a bicyclist can be seen in the foreground, which, recognizable by their different colors, belong to different clusters. Since the two objects were clustered separately and are therefore treated separately, the bicyclist is correctly classified as dynamic, recognizable by the red coloring in the lower left image. However, on the right side, the following image shows that the two objects, the car and the bicyclist in the foreground, have now been clustered together, recognizable by their same colors. As a result, both objects are treated as one object and the distribution of the associated flow vectors is distorted. As a result, the bicyclist is now also classified as static, recognizable by the green coloring in the lower right.

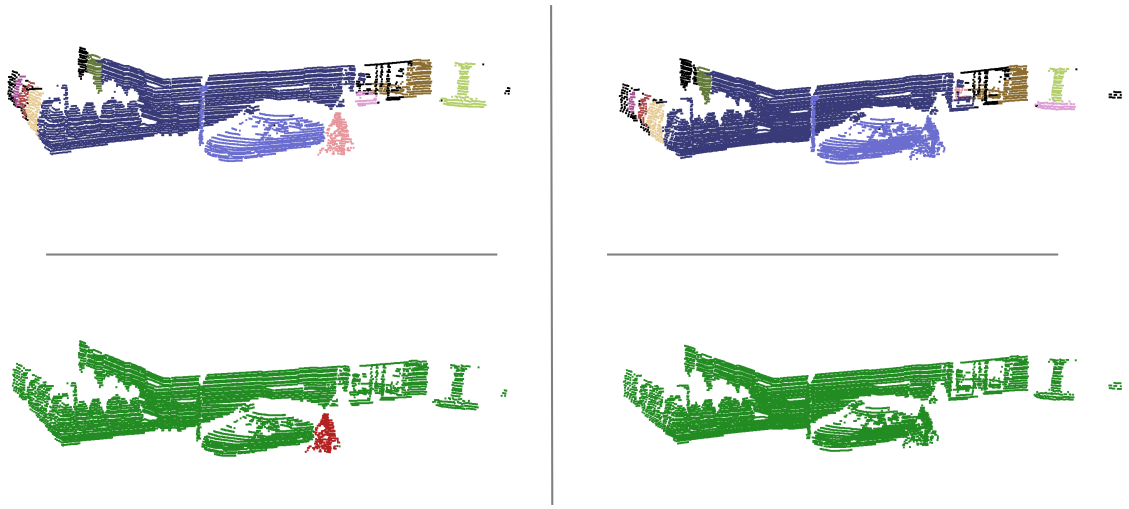


Figure 4.9: Visualization of the disadvantage of the first clustering. Two consecutive frames are shown (column left and column right). The top left image shows a situation where a parked car and moving bicyclist were understood as two separate objects during clustering. Because of that, the bicyclist is classified as dynamic, indicated by the green coloring in the bottom left image. The top right image shows the subsequent frame where the two objects in the foreground have been clustered together, indicated by the same color. As a result, the moving bicyclist and the parked car are considered as one object and classified as static, indicated by the green coloring in the bottom right frame.

5 Conclusion

In this thesis, the potential of SV and SSV models to detect anomalies is demonstrated. In particular, we show that a point-wise check for consistency of two labels predicted in different ways can reveal the limitations of the models used, thereby detecting *method level* anomalies. The knowledge that a model has made an incorrect prediction has many possible use cases: For example, the data where the model is wrong with its prediction can be used to re-train the models, resulting in models that are more robust for similar situations. Another use case would be that the autonomous driving system place itself into a safe state and request remote assistance in response to the detected incorrect prediction [95].
For the detection of *method level* anomalies, the point-wise semantic motion labels of a SV part are compared with the motion labels of a SSV part. The SV part consists of a SV semantic segmentation model and a SV moving object segmentation model, which assign a semantic motion class to each point. Based on the assigned class, the motion state of the point can be inferred. The SSV part is composed of a scene flow model that predicts a displacement vector into the next frame for each point, and an odometry model used to compensate for ego-motion. In a two-step clustering procedure, each point is classified as dynamic or static based on the flow vectors compensated for ego-motion.
Comparing the labels of the two parts per point, four scenarios can occur. Both parts are consistent in that both classify the point as either static or dynamic. Or both parts are inconsistent in that the point is classified once as dynamic and once as static and vice versa. To obtain inconsistent objects, the inconsistent points are clustered. The clusters serve as an indication of an anomaly.
For evaluation, both quantitative and qualitative analyses were performed on the KITTI odometry dataset. In the quantitative evaluation, I analyzed the motion segmentation performance of the SV and SSV part separately. In addition, the uncertainty of the prediction per class of the semantic segmentation model was examined in more detail. It was found that the uncertainty per class depends significantly on the distribution of classes in the training data. For the qualitative analysis, I mapped the point cloud to the corresponding image and colored each scenario differently. The images could be assigned to four different categories, namely *correctly consistent*, *incorrectly consistent*, *anomalies detected by the SSV part*, and *anomalies detected by the SV part*. Representative images were selected for each category. In addition, I identified and analyzed weak points of both parts of my method.

5.1 Outlook

Since this work is the first to address the detection of anomalies in lidar data by combining SV and SSV models, there are many more opportunities to exploit the potential of the two different

training methods.
In a first step, a study of the individual components could be carried out directly after this work. Based on this study, the causes of the weak points, explained in Chapter 4.4, could be identified. For example, the cause of the SSV part for the sometimes many misclassified dynamic points during a speed bump or when turning could be further investigated. The assumption that the many misclassified dynamic points in the sudden vertical motion during a speed bump are due to large displacements and thus to the flow vector could thereby be investigated. In addition, the assumption that sometimes turning is not correctly represented by the odometry model, leading to many misclassified dynamic points, could be examined. This could be tested, for example, by using the ground truth odometry data from sequence 9 or 10 of the KITTI odometry dataset instead of the odometry data predicted by the SSV model. With an otherwise identical setup, the motion segmentation performance of the SSV part could be compared once with ground truth odometry and once with estimated odometry data, especially during turning. It would also be interesting to investigate why fast oncoming cars are not classified as dynamic. It would also be interesting to explore why fast oncoming cars are not classified as dynamic by the SSV part. Is this due to the models or to the parameters I set, such as the threshold for the normalized standard deviation?
Regarding the SV part, the SV semantic segmentation model could be directly trained to distinguish, for example, between moving and parked cars. For this purpose, semantic motion labels would need to be available for training. These can be obtained if both the semantic class and the motion label are determined together for each point during KITTI-360 recovery. Subsequently, a ground truth dataset with semantic motion labels can be derived.
Another interesting direction would be to determine the part that is wrong during inference. One possible idea would be to examine the uncertainty of the SV part. For the quantitative analysis, I examined the uncertainty of the SV semantic segmentation model. If the uncertainty for the segmentation model for moving objects is now also available, it may be possible to derive indications of erroneous predictions during inference. Alternatively, the predicted semantic motion label with associated uncertainty values of the re-trained model mentioned in the previous section can be used during inference.

To detect anomalies at the *object layer*, one could combine my SSV part with a closed-set object detector. An indication of an *object-layer* anomaly would be if the SSV part finds a dynamic cluster that was not detected by the object detector.

A List of Figures

B List of Tables

C Bibliography

[1] Florian Heidecker, Jasmin Breitenstein, Kevin Rösch, Jonas Löhdefink, Maarten Bieshaar, Christoph Stiller, Tim Fingscheidt, and Bernhard Sick. An application-driven conceptualization of corner cases for perception in highly automated driving. In *IEEE Intelligent Vehicles Symposium (IV)*, 2021.

[2] Jasmin Breitenstein, Jan-Aike Termöhlen, Daniel Lipinski, and Tim Fingscheidt. Corner cases for visual perception in automated driving: Some guidance on detection approaches. *arXiv preprint:2102.05897*, 2021.

[3] Jan-Aike Bolte, Andreas Bar, Daniel Lipinski, and Tim Fingscheidt. Towards corner case detection for autonomous driving. In *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[4] Enrique Marti, Miguel Angel de Miguel, Fernando Garcia, and Joshue Perez. A review of sensor technologies for perception in automated driving. *IEEE Intelligent Transportation Systems Magazine*, 11, 2019.

[5] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21, 2021.

[6] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Gläser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 22, 2021.

[7] Gaby Hayon. Autonomous Vehicle Sensing Tech and Algorithms with Gaby Hayon of Mobileye. `/web/20220527110427/https://www.youtube.com/watch?v=GjmZyMWo7YY`, 2020. Accessed: 2022-27-05.

[8] Jasmin Breitenstein, Jan-Aike Termohlen, Daniel Lipinski, and Tim Fingscheidt. Systematization of Corner Cases for Visual Perception in Automated Driving. In *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[9] Vijay Badrinarayanan Alex Kendall and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *British Machine Vision Conference*, 2017.

[10] Giancarlo Di Biase, Hermann Blum, Roland Y. Siegwart, and César Cadena. Pixel-wise anomaly detection in complex driving scenes. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[11] Julia Nitsch, Masha Itkina, Ransalu Senanayake, Juan I. Nieto, Max Theo Schmidt, Roland Y. Siegwart, Mykel J. Kochenderfer, and César Cadena. Out-of-distribution detection for automotive perception. *IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021.

[12] K. J. Joseph, S. Khan, F. Khan, and V. N. Balasubramanian. Towards open world object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[13] Daniel Bogdoll, Maximilian Nitsche, and J. Marius Zöllner. Anomaly Detection in Autonomous Driving: A Survey. *arXiv preprint:2204.07974*, 2022.

[14] Thomas Goelles, Birgit Schlager, and Stefan Muckenhuber. Fault detection, isolation, identification and recovery (fdiir) methods for automotive perception sensors including a detailed literature survey for lidar. *Sensors*, 20, 2020.

[15] Kelvin Wong, Shenlong Wang, Mengye Ren, Ming Liang, and Raquel Urtasun. Identifying Unknown Instances for Autonomous Driving. In *Proceedings of the Conference on Robot Learning*, 2020.

[16] J. Cen, P. Yun, J. Cai, M. Wang, and M. Liu. Open-set 3d object detection. *International Conference on 3D Vision (3DV)*, 2021.

[17] Mana Masuda, Ryo Hachiuma, Ryo Fujii, Hideo Saito, and Yusuke Sekikawa. Toward unsupervised 3d point cloud anomaly detection using variational autoencoder. In *IEEE International Conference on Image Processing (ICIP)*, 2021.

[18] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *arXiv preprint:1512.03012*, 2015.

[19] Hafsa Iqbal, Abdulla Al-Kaff, Pablo Marin, Lucio Marcenaro, David Martin Gomez, and Carlo Regazzoni. Detection of abnormal motion by estimating scene flows of point clouds for autonomous driving. In *IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021.

[20] V. E. Zuev. Laser-light transmission through the atmosphere. In *Laser Monitoring of the Atmosphere*. Springer Berlin Heidelberg, 1976.

[21] J Wojtanowski, M Zygmunt, M Kaszczuk, Z Mierczyk, and M Muzal. Comparison of 905 nm and 1550 nm semiconductor laser rangefinders' performance deterioration due to adverse environmental conditions. 2014.

[22] Sinan Hasirlioglu, Igor Doric, Christian Lauerer, and Thomas Brandmeier. Modeling and simulation of rain for the test of automotive sensor systems. In *IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[23] Chen Zhang, Zefan Huang, Marcelo H. Ang, and Daniela Rus. LiDAR Degradation Quantification for Autonomous Driving in Rain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, 2021.

[24] Lukas Ruff, Robert A. Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep Semi-Supervised Anomaly Detection. *arXiv preprint:1906.02694*, 2020.

[25] Peter Pinggera, Sebastian Ramos, Stefan Gehrig, Uwe Franke, Carsten Rother, and Rudolf Mester. Lost and Found: detecting small road hazards for self-driving vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea, 2016.

[26] Krzysztof Lis, Krishna Nakka, Pascal Fua, and Mathieu Salzmann. Detecting the unexpected via image resynthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[27] Dan Hendrycks, Steven Basart, Mantas Mazeika, Mohammadreza Mostajabi, Jacob Steinhardt, and Dawn Song. Scaling out-of-distribution detection for real-world settings. *arXiv preprint:1911.11132*, 2019.

[28] Hermann Blum, Paul-Edouard Sarlin, Juan Nieto, Roland Siegwart, and Cesar Cadena. The Fishyscapes Benchmark: Measuring Blind Spots in Semantic Segmentation. *International Journal of Computer Vision*, 129, 2021.

[29] Mahesh M Dhananjaya, Varun Ravi Kumar, and Senthil Yogamani. Weather and light level classification for autonomous driving: Dataset, baseline and active learning. In *IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021.

[30] Robin Chan, Krzysztof Lis, Svenja Uhlemeyer, Hermann Blum, Sina Honari, Roland Siegwart, Mathieu Salzmann, Pascal Fua, and Matthias Rottmann. SegmentMeIfYouCan: A Benchmark for Anomaly Segmentation. *arXiv preprint:2104.14812*, 2021.

[31] Daniel Bogdoll, Felix Schreyer, and J. Marius Zöllner. ad-datasets: a meta-collection of data sets for autonomous driving. In *Proceedings of the 8th International Conference on Vehicle Technology and Intelligent Transport Systems*, 2022.

[32] Bogdoll, Daniel and Schreyer, Felix. ad datasets. `https://web.archive.org/web/20220526144859/https://ad-datasets.com/`, 2022. Accessed: 2022-26-05.

[33] Kaican Li, Kai Chen, Haoyu Wang, Lanqing Hong, Chaoqiang Ye, Jianhua Han, Yukuai Chen, Wei Zhang, Chunjing Xu, Dit-Yan Yeung, Xiaodan Liang, Zhenguo Li, and Hang Xu. CODA: A Real-World Road Corner Case Dataset for Object Detection in Autonomous Driving. *arXiv preprint: 2203.07724*, 2022.

[34] Tesla. Tesla AI Day. https://web.archive.org/web/20220527085936/https://www.youtube.com/watch?v=j0z4FweCy4M, 2021. Accessed: 2022-27-05.

[35] Rudra P. K. Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-SCNN: Fast Semantic Segmentation Network. *arXiv preprint:1902.04502*, 2019.

[36] Yuhui Yuan, Xiaokang Chen, Xilin Chen, and Jingdong Wang. Segmentation Transformer: Object-Contextual Representations for Semantic Segmentation. *arXiv preprint:1909.11065*, 2021.

[37] Haotian Yan, Chuang Zhang, and Ming Wu. Lawin Transformer: Improving Semantic Segmentation Transformer with Multi-Scale Representations via Large Window Attention. *arXiv preprint:2201.01615*, 2022.

[38] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. Salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds for autonomous driving. In *Advances in Visual Computing*, 2020.

[39] Ran Cheng, Ryan Razani, Ehsan Taghavi, Enxu Li, and Bingbing Liu. (af)2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[40] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43, 2021.

[41] Deyvid Kochanov, Fatemeh Karimi Nejadasl, and Olaf Booij. KPRNet: Improving projection-based LiDAR semantic segmentation. *arXiv preprint:2007.12668*, 2020.

[42] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Wei Li, Yuexin Ma, Hongsheng Li, Ruigang Yang, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar-based perception. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[43] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. *Computer Vision – ECCV 2020*, 2020.

[44] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. In *Advances in Neural Information Processing Systems*, 2019.

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

[46] Yiyi Liao, Jun Xie, and Andreas Geiger. KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv preprint:2109.13410*, 2021.

[47] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: Audi Autonomous Driving Dataset. *arXiv preprint:2004.06320*, 2020.

[48] Yancheng Pan, Biao Gao, Jilin Mei, Sibo Geng, Chengkun Li, and Huijing Zhao. Semanticposs: A point cloud dataset with large quantity of dynamic instances. In *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[49] 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*, 2012.

[50] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[51] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *Advances in Neural Information Processing Systems*, 2017.

[52] Dimity Miller. *Epistemic uncertainty estimation for object detection in open-set conditions*. PhD thesis, Queensland University of Technology, 2021.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 2014.

[54] Jochen Gast and Stefan Roth. Lightweight probabilistic deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[55] Larissa T. Triess, David Peter, Christoph B. Rist, and J. Marius Zöllner. Scan-based Semantic Segmentation of LiDAR Point Clouds: An Experimental Study. *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[56] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

[57] Stefan Baur, David Emmerichs, Frank Moosmann, Peter Pinggera, Bjorn Ommer, and Andreas Geiger. SLIM: Self-Supervised LiDAR Scene Flow and Motion Segmentation. In *International Conference on Computer Vision (ICCV)*, 2021.

[58] Mingliang Zhai, Xuezhi Xiang, Ning Lv, and Xiangdong Kong. Optical flow and scene flow estimation: A survey. *Pattern Recognition*, 114, 2021.

[59] Himangi Mittal, Brian Okorn, and David Held. Just go with the flow: Self-supervised scene flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[60] Yair Kittenplon, Yonina C. Eldar, and Dan Raviv. Flowstep3d: Model unrolling for self-supervised scene flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[61] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[62] Wenxuan Wu, Zhiyuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. Pointpwc-net: A coarse-to-fine network for supervised and self-supervised scene flow estimation on 3d point clouds. *arXiv preprint:1911.12408*, 2019.

[63] Ivan Tishchenko, Sandro Lombardi, Martin R. Oswald, and Marc Pollefeys. Self-supervised learning of non-rigid residual flow and ego-motion. In *International Conference on 3D Vision (3DV)*, 2020.

[64] Bojun Ouyang and Dan Raviv. Occlusion guided self-supervised scene flow estimation on 3d point clouds. In *International Conference on 3D Vision (3DV)*, 2021.

[65] Li Zhiqi. A Review on Deep Learning Methods for Scene Flow Estimation. *Bournemouth University Postgraduate Research Conference*, 2021.

[66] V. Zuanazzi, J. Vugt, O. Booij, and P. Mettes. Adversarial self-supervised scene flow estimation. In *International Conference on 3D Vision (3DV)*, 2020.

[67] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[68] M. Menze, C. Heipke, and A. Geiger. JOINT 3D ESTIMATION OF VEHICLES AND SCENE FLOW. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2015.

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

[70] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. HPLFlowNet: Hierarchical Permutohedral Lattice FlowNet for Scene Flow Estimation on Large-Scale Point Clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[71] Guangming Wang, Chaokang Jiang, Zehang Shen, Yanzi Miao, and Hesheng Wang. SFGAN: Unsupervised Generative Adversarial Learning of 3D Scene Flow from the 3D Scene Self. *Advanced Intelligent Systems*, 2021.

[72] Nikhil Jonnavithula, Yecheng Lyu, and Ziming Zhang. LiDAR Odometry Methodologies for Autonomous Driving: A Survey. *arXiv preprint:2109.06120*, 2021.

[73] Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Tomi Westerlund, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7, 2019.

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

[75] Julian Nubert, Shehryar Khattak, and Marco Hutter. Self-supervised learning of lidar odometry for robotic applications. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[76] Xu Fu, Cong Liu, Chengjin Zhang, Zihao Sun, Yong Song, Qingyang Xu, and Xianfeng Yuan. Self-supervised learning of LiDAR odometry based on spherical projection. *International Journal of Advanced Robotic Systems*, 19, 2022.

[77] Yan Xu, Zhaoyang Huang, Kwan-Yee Lin, Xinge Zhu, Jianping Shi, Hujun Bao, Guofeng Zhang, and Hongsheng Li. SelfVoxeLO: Self-supervised LiDAR Odometry with Voxel-based Deep Neural Networks. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155, 2021.

[78] Yan Xu, Junyi Lin, Jianping Shi, Guofeng Zhang, Xiaogang Wang, and Hongsheng Li. Robust self-supervised lidar odometry via representative structure discovery and 3d inherent error modeling. *IEEE Robotics and Automation Letters*, 7, 2022.

[79] Xieyuanli Chen, Shijie Li, Benedikt Mersch, Louis Wiesmann, Jürgen Gall, Jens Behley, and Cyrill Stachniss. Moving Object Segmentation in 3D LiDAR Data: A Learning-based Approach Exploiting Sequential Data. *IEEE Robotics and Automation Letters*, 6, 2021.

[80] Eren Erdal Aksoy, Saimir Baci, and Selcuk Cavdar. Salsanet: Fast road and vehicle segmentation in lidar point clouds for autonomous driving. In *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[81] Anirudh Topiwala. Spherical projection for point clouds. `https://web.archive.org/web/20220525095742/https://towardsdatascience.com/spherical-projection-for-point-clouds-56a2fc258e6c?gi=ab4d813b3646`, 2021. Accessed: 2022-05-25.

[82] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[83] Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6, 2019.

[84] Maxim Berman, Amal Rannen Triki, and Matthew B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[85] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. Github salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds for autonomous driving. `/web/20220530143050/https://github.com/slawomir-nowaczyk/SalsaNext`, 2020. Accessed: 2022-30-05.

[86] Jens Behley and Cyrill Stachniss. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In *Robotics: Science and Systems XIV*, 2018.

[87] Anshul Paigwar, Ozgur Erkent, David Sierra-Gonzalez, and Christian Laugier. GndNet: Fast Ground Plane Estimation and Point Cloud Segmentation for Autonomous Vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[88] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[89] R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[90] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 2017.

[91] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *Computer Vision – ECCV 2020*, 2020.

[92] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259*, 2014.

[93] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[94] Jules Sanchez. recoverkitti360label. `https://web.archive.org/web/20220525101943/https://github.com/JulesSanchez/recoverKITTI360label`, 2022. Accessed: 2022-05-24.

[95] Daniel Bogdoll, Patrick Matalla, Christoph Füllner, Christian Raack, Shi Li, Tobias Käfer, Stefan Orf, Marc René Zofka, Finn Sartoris, Christoph Schweikert, Thomas Pfeiffer, André Richter, Sebastian Randel, and Rene Bonk. KIGLIS: Smart Networks for Smart Cities. In *IEEE International Smart Cities Conference (ISC2)*, 2021.

D Visualizations Semantic Segmentation on Lidar

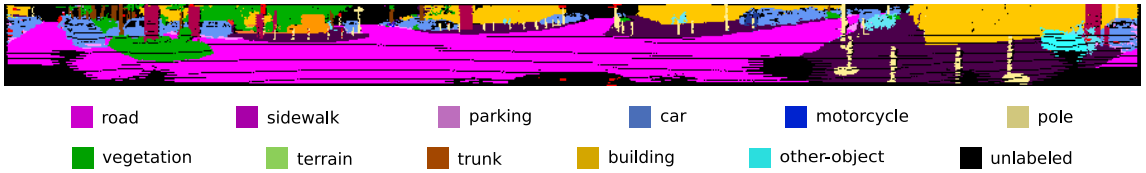


Figure D.1: Visualization of a 3D point cloud with semantic labels projected onto a 2D range view image [45].

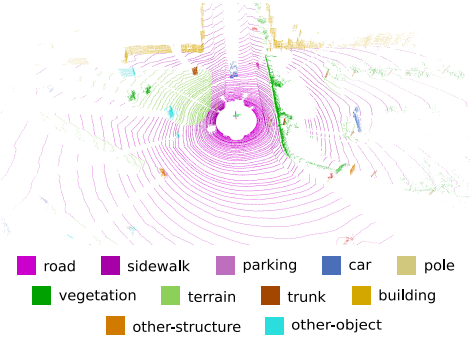


Figure D.2: Visualization of a 3D lidar point cloud with semantic labels [45].

E Examples for Motion Labels by Combining Scene Flow and Odometry

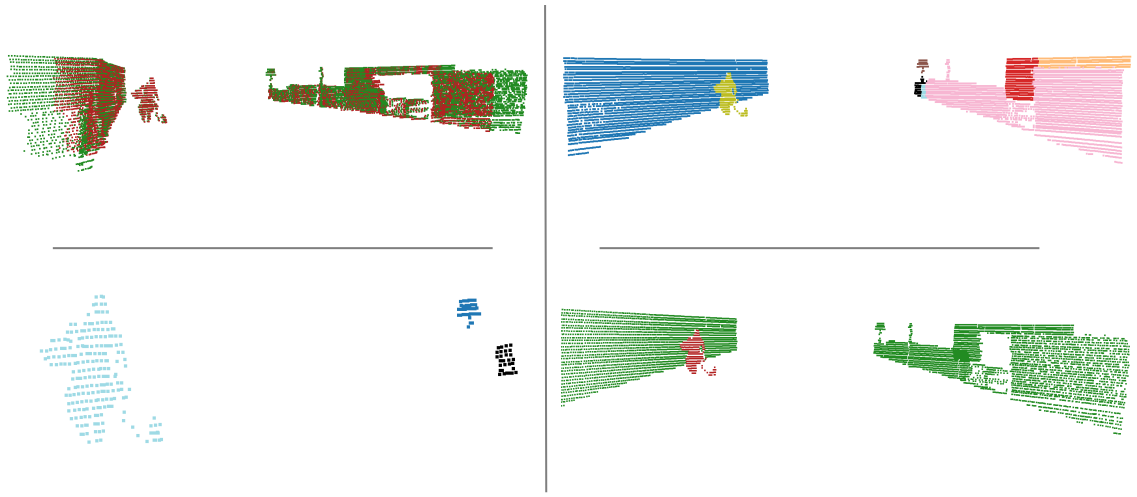


Figure E.1: Example scene showing the steps of combining scene flow, odometry data and clustering in order to perform motion segmentation. In the top left image, the transformation of a point cloud estimated in advance with the flow vector into the next image was transformed back into the original image. The original point cloud is shown in green and the back-transformed point cloud is shown in red. The top right image shows the result of the first clustering of the points, with the different clusters represented by different colors. The bottom left image shows the result of the second clustering, this time based on the flow vectors of the potentially dynamic clusters. Again, different colors indicate belonging to different clusters. The bottom right image shows the final result of the process, where the points classified as dynamic are shown in red and the static ones in green.The scene shown is taken from sequence 0 frame 814 of the KITTI odometry dataset.

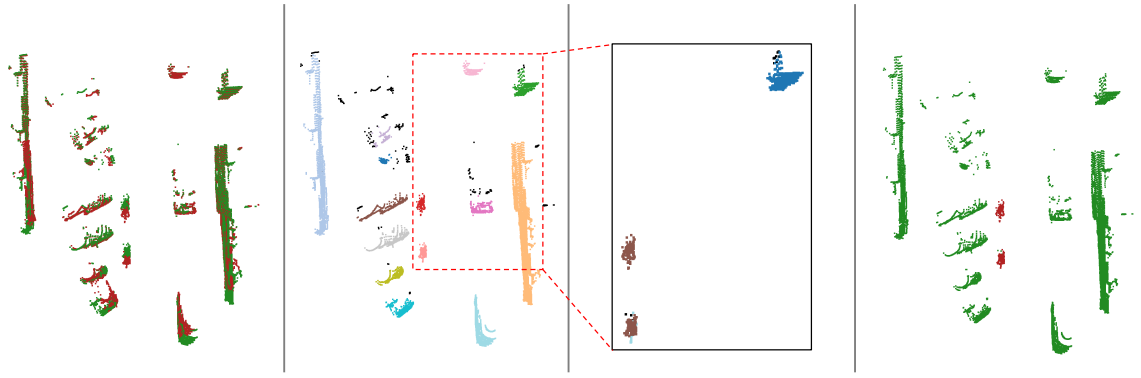


Figure E.2: Example scene showing the steps of combining scene flow, odometry data and clustering in order to perform motion segmentation. In the first image from the left, the transformation of a point cloud estimated in advance with the flow vector into the next image was transformed back into the original image. The original point cloud is shown in green and the back-transformed point cloud is shown in red. The second image from the left shows the result of the first clustering of the points, with the different clusters represented by different colors. The third image from the left shows the result of the second clustering, this time based on the flow vectors of the potentially dynamic clusters. Again, different colors indicate belonging to different clusters. The last image shows the final result of the process, where the points classified as dynamic are shown in red and the static ones in green.The scene shown is taken from sequence 13 frame 23 of the KITTI odometry dataset.

F Epistemic Uncertainty

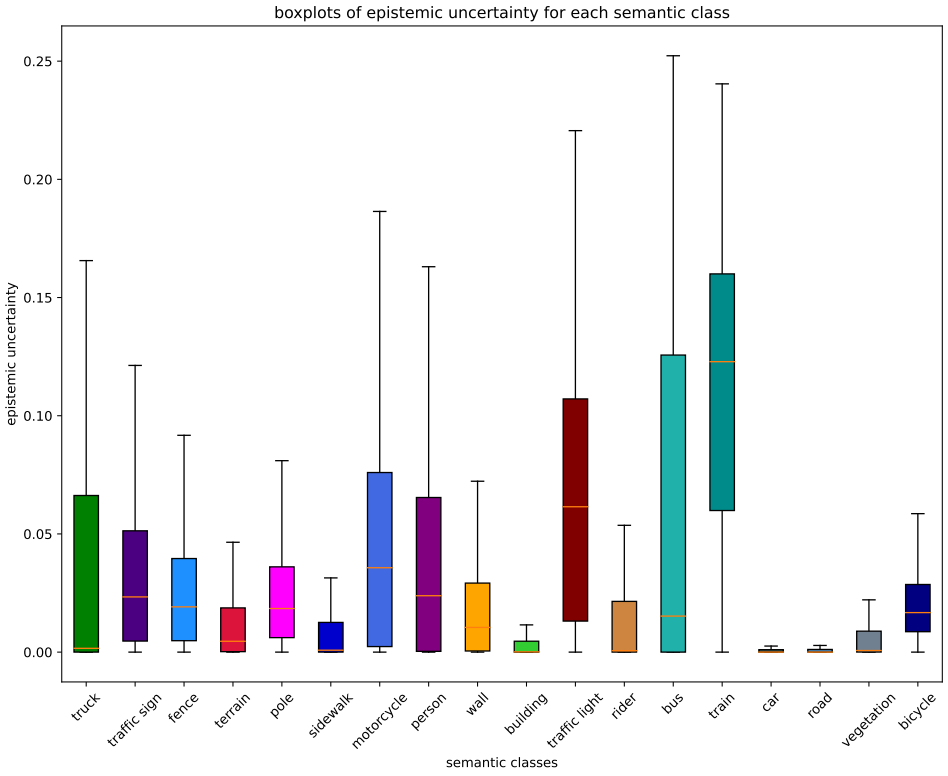


Figure F.1: Boxplots of epistemic uncertainty of SalsaNext per semantic class. The entire KITTI odometry dataset, sequence 00 to 21, was used for this plot.